SIMULIA

3DEXPERIENCE®

ABAQUS 2025 FD04
SCRIPTING REFERENCE GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....28
What's New....29
Abaqus Scripting Reference Guide....35
Python commands....36
Abaqus/CAE Display Preferences commands....38
caePrefsAccess module....39
CaeGuiPrefs object....42
CaeKerPrefs object....44
Adaptivity commands....46
AdaptiveMeshConstraint object....47
AdaptiveMeshConstraintState object....50
AdaptiveMeshControl object....51
AdaptiveMeshDomain object....54
AdaptivityIteration object....56
AdaptivityProcess object....58
DisplacementAdaptiveMeshConstraint object....60
DisplacementAdaptiveMeshConstraintState object....64
ErrorIndicatorResult object....66
Model object....67
RemeshingRule object....68
RuleResult object....72
VelocityAdaptiveMeshConstraint object....73
VelocityAdaptiveMeshConstraintState object....76
Amplitude commands....78
Amplitude object....79
ActuatorAmplitude object....80
BaselineCorrection object....82
Correlation object....84
DecayAmplitude object....85
EquallySpacedAmplitude object....87
ModulatedAmplitude object....89
PeriodicAmplitude object....91
PsdDefinition object....93
SmoothStepAmplitude object....95
SolutionDependentAmplitude object....97
SpectrumAmplitude object....99
TabularAmplitude object....101
UserAmplitude object....103
Animation commands....105
AnimationController object....106
AnimationOptions object....110
AVIOptions object....115
GIFOptions object....117

ImageAnimationOptions object....118
ImageAnimation object....119
QuickTimeOptions object....121
MP4Options object....123
Movie object....125
Session object....128
Annotation commands....129
Annotation object....130
AnnotationsToPlotArray object....133
Arrow object....136
Text object....140
Viewport object....143
Assembly commands....144
Assembly object....145
ConnectorOrientation object....167
Feature object....169
PartInstance object....175
ModelInstance object....185
Model object....188
Basic geometry commands....190
Cell object....191
CellArray object....195
Edge object....200
EdgeArray object....205
Face object....211
FaceArray object....218
IgnoredEdge object....224
IgnoredEdgeArray object....226
IgnoredVertex object....229
IgnoredVertexArray object....230
InterestingPoint object....233
ModelDot object....234
ReferencePoint object....235
Transform object....236
Vertex object....237
VertexArray object....239
Beam Section profile commands....244
Profile object....245
ArbitraryProfile object....246
BoxProfile object....248
ChannelProfile object....250
CircularProfile object....252
GeneralizedProfile object....254
HatProfile object....256
HexagonalProfile object....258
IProfile object....260
LProfile object....262
PipeProfile object....264

RectangularProfile object....266
TProfile object....268
TrapezoidalProfile object....270
Boundary Condition commands....272
BoundaryCondition object....274
AccelerationBaseMotionBC object....277
AccelerationBaseMotionBCState object....280
AccelerationBC object....281
AccelerationBCState object....285
AcousticPressureBC object....287
AcousticPressureBCState object....290
BoundaryConditionState object....292
ConcentrationBC object....293
ConcentrationBCState object....296
ConnAccelerationBC object....298
ConnAccelerationBCState object....302
ConnDisplacementBC object....304
ConnDisplacementBCState object....308
ConnVelocityBC object....310
ConnVelocityBCState object....314
DisplacementBaseMotionBC object....316
DisplacementBaseMotionBCState object....319
DisplacementBC object....320
DisplacementBCState object....324
ElectricPotentialBC object....326
ElectricPotentialBCState object....329
ElectricPotentialElectrolyteBC object....331
ElectricPotentialElectrolyteBCState object....334
EulerianBC object....336
EulerianBCState object....339
EulerianMotionBC object....341
EulerianMotionBCState object....349
FluidCavityPressureBC object....352
FluidCavityPressureBCState object....355
IonConcentrationBC object....357
IonConcentrationBCState object....360
MagneticVectorPotentialBC object....362
MaterialFlowBC object....365
MaterialFlowBCState object....368
PorePressureBC object....370
PorePressureBCState object....373
RetainedNodalDofsBC object....375
SecondaryBaseBC object....378
SecondaryBaseBCState object....380
SubmodelBC object....381
SubmodelBCState object....385
TemperatureBC object....387
TemperatureBCState object....390

TypeBC object....392
TypeBCState object....400
VelocityBaseMotionBC object....402
VelocityBaseMotionBCState object....405
VelocityBC object....406
VelocityBCState object....410
Calibration commands....412
Calibration object....413
Behavior object....414
DataSet object....421
Canvas commands....423
AttributeColorMap object....424
DrawingArea object....427
Layer object....428
Viewport object....430
ImageOptions object....446
MovieOptions object....448
Highlight commands....450
Connector commands....452
ConnectorBehaviorOption object....453
CDCTerm object....454
ConnectorDamage object....457
ConnectorDamping object....462
ConnectorElasticity object....466
ConnectorFailure object....470
ConnectorFriction object....472
ConnectorLock object....476
ConnectorOptions object....478
ConnectorPlasticity object....481
ConnectorPotential object....485
ConnectorStop object....488
DerivedComponent object....490
TangentialBehavior object....492
Constraint commands....495
Constraint object....496
AdjustPoints object....498
Coupling object....500
DisplayBody object....503
EmbeddedRegion object....505
Equation object....507
MultipointConstraint object....509
RigidBody object....511
ShellSolidCoupling object....513
Tie object....515
CustomKernel module....518
CommandRegister object....519
RepositorySupport object....520
RegisteredDictionary object....522

RegisteredList object....524
RegisteredTuple object....525
Datum commands....526
Datum object....527
DatumAxis object....528
DatumCsys object....529
DatumPlane object....531
DatumPoint object....532
Display Group commands....533
DisplayGroup object....534
DisplayGroupInstance object....538
DisplayGroupInstanceRepository object....540
Leaf object....541
LeafFromConstraintNames object....542
LeafFromDatums object....543
LeafFromDisplayGroup object....544
LeafFromElementLabels object....545
LeafFromElementSets object....546
LeafFromElementVarRange object....547
LeafFromGeometry object....548
LeafFromInstance object....549
LeafFromInstanceElementLabels object....550
LeafFromInstanceNodeLabels object....551
LeafFromMeshElementLabels object....552
LeafFromMeshNodeLabels object....553
LeafFromMeshSurfaceSets object....554
LeafFromModelElemLabels object....555
LeafFromModelNodeLabels object....557
LeafFromNodeLabels object....558
LeafFromNodeSets object....559
LeafFromNodeVarRange object....560
LeafFromOdbElementLayups object....561
LeafFromOdbElementMaterials object....562
LeafFromOdbElementPick object....563
LeafFromOdbElementPlies object....564
LeafFromOdbElementSections object....565
LeafFromOdbElementTypes object....566
LeafFromOdbNodePick object....567
LeafFromOdbEdgePick object....568
LeafFromPartElementLabels object....569
LeafFromPartInstance object....570
LeafFromPartNodeLabels object....571
LeafFromReferencePoint object....572
LeafFromSets object....573
LeafFromSurfaceSets object....574
LeafFromSurfaceVarRange object....575
Display Options commands....576
AssemblyDisplayOptions object....577

BCDisplayOptions object....581
ConstraintDisplayOptions object....583
EngineeringFeatureDisplayOptions object....584
PredefinedFieldDisplayOptions object....586
GeometryDisplayOptions object....587
GeometricRestrictionDisplayOptions object....589
GraphicsInfo object....591
GraphicsOptions object....592
InteractionDisplayOptions object....602
Light object....604
LightOptions object....606
LoadDisplayOptions object....609
MeshDisplayOptions object....612
OptimizationTaskDisplayOptions object....614
PartDisplayOptions object....615
StopConditionDisplayOptions object....617
SymbolDisplayOptions object....618
ViewportAnnotationOptions object....620
Edit mesh commands....623
Assembly object....624
Part object....631
MeshEditOptions object....645
Engineering Feature commands....646
EngineeringFeature object....647
AssembledFastener object....649
ContourIntegral object....651
Crack object....654
DebondVCCT object....655
DiscreteFastener object....657
Fastener object....659
HeatCapacitance object....660
Inertia object....662
NonstructuralMass object....663
PointFastener object....665
PointMassInertia object....669
SpringDashpot object....672
SpringDashpotToGround object....673
TwoPointSpringDashpot object....675
XFEMCrack object....678
Imperfection object....681
FileImperfection object....682
InputImperfection object....684
DataImperfection object....686
Feature commands....688
Feature object....689
FeatureOptions object....731
Field commands....732
Field object....733

AnalyticalField object....734
DataTable object....735
DiscreteField object....736
ExpressionField object....739
MappedField object....741
OdbMeshRegionData object....745
Field Report commands....751
FieldReportOptions object....752
FreeBodyReportOptions object....754
OdbFieldVarList object....755
OdbModelFieldVarList object....756
writeFieldReport....757
writeFreeBodyReport....759
Filter commands....760
Filter object....761
ButterworthFilter object....762
Chebyshev1Filter object....764
Chebyshev2Filter object....766
OperatorFilter object....768
Input File Parser commands....770
InputFile object....771
Keyword object....773
AbaqusNDarray object....774
Interaction commands....775
Interaction object....778
AcousticImpedance object....781
AcousticImpedanceProp object....784
AcousticImpedanceState object....786
ActuatorSensor object....787
ActuatorSensorProp object....789
ActuatorSensorState object....791
CavityRadiation object....792
CavityRadiationProp object....799
CavityRadiationState object....801
CohesiveBehavior object....803
ConcentratedFilmCondition object....806
ConcentratedFilmConditionState object....809
ConcentratedRadiationToAmbient object....811
ConcentratedRadiationToAmbientState object....814
ContactControl object....816
ContactDamage object....817
ContactDamping object....823
ContactExp object....826
ContactInitialization object....831
ContactMassScalingExp object....832
ContactProperty object....834
ContactPropertyAssignment object....836
ContactStabilization object....839

ContactStd object....840
ContactTangentialBehavior object....846
CyclicSymmetry object....850
CyclicSymmetryState object....854
ElasticFoundation object....856
ElasticFoundationState object....858
ExpContactControl object....859
ExpInitialization object....861
FilmCondition object....863
FilmConditionProp object....866
FilmConditionState object....868
FluidCavity object....870
FluidCavityProperty object....872
FluidCavityState object....876
FluidExchangeActivation object....877
FluidExchangeActivationState object....879
FluidExchange object....881
FluidExchangeProperty object....883
FluidExchangeState object....886
FluidInflatorActivation object....887
FluidInflatorActivationState object....889
FluidInflator object....890
FluidInflatorProperty object....892
FluidInflatorState object....895
FractureCriterion object....896
GapElectricalConductance object....899
GapHeatGeneration object....902
GeometricProperties object....905
IncidentWave object....907
IncidentWaveProperty object....910
IncidentWaveState object....915
InitializationAssignment object....916
InteractionProperty object....918
InteractionState object....919
MainSecondaryAssignment object....920
Model object....922
ModelChange object....925
NormalBehavior object....928
PolarityAssignments object....931
PressurePenetration object....933
PressurePenetrationState object....936
Radiation object....938
RadiationToAmbient object....940
RadiationToAmbientState object....943
RegionPairs object....945
SelfContactExp object....946
SelfContactExpState object....948
SelfContactStd object....950

SelfContactStdState object....953
SlidingFormulationAssignment object....954
slidingTransitionAssignments object....956
SmoothingAssignment object....958
StabilizationAssignment object....960
StdContactControl object....962
StdInitialization object....965
StdStabilization object....967
StdXplCosimulation object....969
StdXplCosimulationState object....971
SurfaceBeamSmoothingAssignment object....972
SurfaceCrushTriggerAssignment object....974
SurfaceFeatureAssignment object....976
SurfaceFrictionAssignment object....979
SurfaceOffsetAssignment object....981
SurfaceThicknessAssignment object....983
SurfaceToSurfaceContactExp object....985
SurfaceToSurfaceContactStd object....988
SurfaceToSurfaceExpState object....994
SurfaceToSurfaceStdState object....995
SurfaceVertexCriteriaAssignment object....997
ThermalConductance object....999
WearProperty object....1002
WearSurfacePropertyAssignment object....1004
XFEMCrackGrowth object....1006
XFEMCrackGrowthState object....1008
Job commands....1009
Job object....1010
JobFromInputFile object....1014
Message object....1018
ModelJob object....1019
Queue object....1025
Coexecution object....1027
OptimizationProcess object....1031
Load and Load Case commands....1035
Load object....1037
LoadCase object....1040
LoadState object....1042
BodyCharge object....1043
BodyChargeState object....1046
BodyConcentrationFlux object....1047
BodyConcentrationFluxState object....1050
BodyCurrent object....1051
BodyCurrentDensity object....1054
BodyCurrentState object....1057
BodyForce object....1058
BodyForceState object....1061
BodyHeatFlux object....1063

BodyHeatFluxState object....1066
BoltLoad object....1067
BoltLoadState object....1070
ConcCharge object....1072
ConcConcFlux object....1075
ConcCurrent object....1078
ConcCurrentState object....1081
ConcentratedChargeState object....1082
ConcentratedConcentrationFluxState object....1083
ConcentratedForce object....1084
ConcentratedForceState object....1087
ConcentratedHeatFlux object....1089
ConcentratedHeatFluxState object....1092
ConcentratedPoreFluidState object....1093
ConcPoreFluid object....1094
ConnectorForce object....1097
ConnectorForceState object....1100
ConnectorMoment object....1102
ConnectorMomentState object....1105
CoriolisForce object....1107
CoriolisForceState object....1110
DistIonConcBodyFlux object....1111
DistIonConcBodyFluxState object....1114
FluidPressurePenetration object....1115
FluidPressurePenetrationState object....1118
FluidSurfaceCurrent object....1120
FluidSurfaceCurrentState object....1123
Gravity object....1124
GravityState object....1127
HydrostaticFluidFlowState object....1129
InertiaRelief object....1130
InertiaReliefState object....1133
InwardVolAccel object....1135
InwardVolAccelState object....1138
IonConcentrationFlux object....1139
IonConcentrationFluxState object....1142
LineLoad object....1143
LineLoadState object....1146
Moment object....1148
MomentState object....1151
PEGLoad object....1153
PEGLoadState object....1156
PipePressure object....1158
PipePressureState object....1161
Pressure object....1163
PressureState object....1166
RotationalBodyForce object....1168
RotationalBodyForceState object....1172

ShellEdgeLoad object....1173
ShellEdgeLoadState object....1176
SubmodelSB object....1177
SubmodelSBState object....1180
SubstructureLoad object....1182
SubstructureLoadState object....1184
SurfaceCharge object....1185
SurfaceChargeState object....1188
SurfaceConcentrationFlux object....1189
SurfaceConcentrationFluxState object....1192
SurfaceCurrent object....1193
SurfaceCurrentDensity object....1196
SurfaceCurrentState object....1199
SurfaceHeatFlux object....1200
SurfaceHeatFluxState object....1203
SurfacePoreFluid object....1204
SurfacePoreFluidState object....1207
SurfaceTraction object....1208
SurfaceTractionState object....1212
Material commands....1213
Material object....1216
AcousticMedium object....1221
AnnealTemperature object....1223
BiaxialTestData object....1225
BrittleCracking object....1227
BrittleFailure object....1230
BrittleShear object....1232
CapCreepCohesion object....1234
CapCreepConsolidation object....1237
CapHardening object....1240
CapPlasticity object....1242
CastIronCompressionHardening object....1244
CastIronPlasticity object....1246
CastIronTensionHardening object....1248
ClayHardening object....1250
ClayPlasticity object....1252
CombinedTestData object....1255
Concrete object....1257
ConcreteCompressionDamage object....1259
ConcreteCompressionHardening object....1261
ConcreteDamagedPlasticity object....1263
ConcreteTensionDamage object....1265
ConcreteTensionStiffening object....1267
Conductivity object....1270
ContactArea object....1273
Creep object....1275
CrushableFoam object....1279
CrushableFoamHardening object....1281

CrushStress object....1283
CrushStressVelocityFactor object....1285
CycledPlastic object....1287
CyclicHardening object....1289
DamageEvolution object....1291
DamageInitiation object....1297
DamageStabilization object....1326
DamageStabilizationCohesive object....1330
Damping object....1334
DeformationPlasticity object....1336
Density object....1338
Depvar object....1340
DetonationPoint object....1342
Dielectric object....1344
Diffusivity object....1347
DruckerPrager object....1350
DruckerPragerCreep object....1353
DruckerPragerHardening object....1355
Elastic object....1357
ElectricalConductivity object....1362
Eos object....1365
EosCompaction object....1369
Expansion object....1371
FailStrain object....1374
FailStress object....1376
FailureRatios object....1378
FluidLeakoff object....1380
GapConductance object....1382
GapConvection object....1384
GapFlow object....1386
GapRadiation object....1389
GasketMembraneElastic object....1391
GasketThicknessBehavior object....1393
GasketTransverseShearElastic object....1397
Gel object....1399
HeatGeneration object....1401
Hyperelastic object....1402
Hyperfoam object....1407
Hypoelastic object....1409
Hysteresis object....1411
InelasticHeatFraction object....1413
JouleHeatFraction object....1415
LatentHeat object....1417
LowDensityFoam object....1419
MagneticPermeability object....1421
MeanFieldHomogenization object....1424
MeanFieldInclusion object....1426
MeanFieldMatrix object....1429

MeanFieldVoid object....1431
MohrCoulombHardening object....1434
MohrCoulombPlasticity object....1436
MoistureSwelling object....1438
MullinsEffect object....1440
Ornl object....1441
Permeability object....1443
Piezoelectric object....1446
PlanarTestData object....1449
Plastic object....1451
PlasticityCorrection object....1455
PoreFluidExpansion object....1457
PorousBulkModuli object....1459
PorousElastic object....1461
PorousFailureCriteria object....1463
PorousMetalPlasticity object....1465
Potential object....1467
PressureEffect object....1469
RateDependent object....1471
Ratios object....1474
Regularization object....1476
SaturationDependence object....1478
ShearRetention object....1480
ShearTestData object....1482
SimpleShearTestData object....1484
Solubility object....1486
SoretEffect object....1488
Sorption object....1490
SpecificHeat object....1492
SuperElasticHardening object....1494
SuperElasticHardeningModifications object....1496
SuperElasticity object....1498
Swelling object....1500
TensileFailure object....1502
TensionCutOff object....1504
TensionStiffening object....1506
TriaxialTestData object....1508
Trs object....1510
UniaxialTestData object....1512
UserDefinedField object....1515
UserMaterial object....1516
UserOutputVariables object....1518
VelocityDependence object....1520
Viscoelastic object....1522
Viscosity object....1526
Viscous object....1528
VoidNucleation object....1532
VolumetricTestData object....1534

evaluateMaterial....1537
Mdb commands....1540
Mdb object....1541
Mdb commands....1552
Mesh commands....1554
Assembly object....1555
Part object....1580
ElemType object....1605
MeshEdge object....1609
MeshEdgeArray object....1611
MeshElement object....1613
MeshElementArray object....1617
MesherOptions object....1623
MeshFace object....1625
MeshFaceArray object....1630
MeshNode object....1632
MeshNodeArray object....1636
MeshStats object....1642
Messaging commands....1643
DataObject object....1644
MonitorMgr object....1647
Model commands....1651
Model object....1652
KeywordBlock object....1660
Odb commands....1663
Odb object....1664
AnalyticSurface object....1668
AnalyticSurfaceSegment object....1669
BeamOrientation object....1670
FieldBulkData object....1671
FieldLocation object....1673
FieldOutput object....1674
FieldValue object....1688
HistoryOutput object....1691
HistoryPoint object....1694
HistoryRegion object....1699
JobData object....1702
OdbAssembly object....1703
OdbDatumCsys object....1710
OdbFrame object....1715
OdbInstance object....1718
OdbLoadCase object....1727
OdbMeshElementArray object....1728
OdbMeshElement object....1730
OdbMeshNodeArray object....1732
OdbMeshNode object....1733
OdbPart object....1734
OdbPretensionSection object....1743

OdbRigidBody object....1744
OdbSequenceAnalyticSurfaceSegment object....1746
OdbSet object....1749
OdbStep object....1755
RebarOrientation object....1761
ScratchOdb object....1762
SectionCategory object....1763
SectionPoint object....1765
SectorDefinition object....1767
UserData object....1768
Odb commands....1770
Odb Display commands....1775
OdbDisplay object....1776
CommonOptions object....1787
ContourOptions object....1794
DefaultOdbDisplay object....1801
DisplayBodyOptions object....1802
OrientationOptions object....1805
SuperimposeOptions object....1808
SymbolOptions object....1815
ViewCut object....1821
ViewerOptions object....1826
Optimization commands....1827
OptimizationTask object....1829
BeadTask object....1830
CombinedTermDesignResponse object....1833
DesignDirection object....1836
DesignResponse object....1838
DrillControl object....1839
FixedRegion object....1841
FrozenArea object....1843
GeometricRestriction object....1845
Growth object....1846
LocalStopCondition object....1848
ObjectiveFunction object....1850
OptimizationConstraint object....1852
OptimizationObjective object....1854
PenetrationCheck object....1855
ShapeDemoldControl object....1857
ShapeMemberSize object....1859
ShapePlanarSymmetry object....1861
ShapePointSymmetry object....1863
ShapeRotationalSymmetry object....1865
ShapeTask object....1867
SingleTermDesignResponse object....1872
SizingClusterAreas object....1874
SizingCyclicSymmetry object....1876
SizingFrozenArea object....1878

SizingMemberSize object....1880
SizingPlanarSymmetry object....1882
SizingPointSymmetry object....1884
SizingRotationalSymmetry object....1886
SizingTask object....1888
SlideRegionControl object....1891
StampControl object....1893
StepOption object....1895
StopCondition object....1896
TopologyCyclicSymmetry object....1897
TopologyDemoldControl object....1899
TopologyMemberSize object....1901
TopologyOverhangControl object....1903
TopologyPlanarSymmetry object....1905
TopologyPointSymmetry object....1907
TopologyRibDesign object....1909
TopologyRotationalSymmetry object....1911
TopologyMillingControl object....1913
TopologyTask object....1915
TurnControl object....1919
BeadFilter object....1921
BeadFixedRegion object....1923
BeadGrowth object....1925
BeadPenetrationCheck object....1927
BeadPlanarSymmetry object....1929
BeadPointSymmetry object....1931
BeadRotationalSymmetry object....1933
Part commands....1935
Part object....1936
Feature object....1975
AcisFile object....2019
Partition commands....2025
Path and Probe commands....2026
Path object....2027
CurrentProbeValues object....2029
FreeBody object....2030
NodeQuery object....2033
ProbeOptions object....2034
ProbeReport object....2038
SelectedProbeValues object....2040
Spectrum object....2041
Stream object....2042
Plot Options commands....2043
BasicOptions object....2045
DGCommonOptions object....2058
DGContourOptions object....2061
DisplayOptions object....2063
DGDisplayBodyOptions object....2064

FreeBodyOptions object....2066
HistoryVariable object....2069
OdbData object....2070
OdbDataDatumCsys object....2073
OdbDataElementSet object....2074
OdbDataFrame object....2075
OdbDataInstance object....2076
OdbDataMaterial object....2077
OdbDataNodeSet object....2078
OdbDataSection object....2079
OdbDataStep object....2080
OdbDataSurfaceSet object....2081
CouplingConstraint object....2082
RigidBodyConstraint object....2083
TieConstraint object....2084
MpcConstraint object....2085
OdbDiagnosticData object....2086
OdbAnalysisError object....2087
OdbAnalysisWarning object....2088
OdbJobTime object....2089
OdbNumericalProblemSummary object....2090
OdbDiagnosticStep object....2091
OdbAuxiliaryData object....2094
OdbDiagnosticIncrement object....2095
OdbDiagnosticAttempt object....2096
OdbContactDiagnostics object....2097
OdbDisplayOptions object....2098
OptionArg object....2099
DGOrientationOptions object....2101
StreamOptions object....2103
DGSuperimposeOptions object....2105
DGSymbolOptions object....2108
ViewCutOptions object....2111
DetailPlotOptions object....2115
PlyStackPlotOptions object....2116
MdbData object....2119
MdbDataFrame object....2120
MdbDataInstance object....2121
MdbDataStep object....2122
Plug-in registration commands....2123
Predefined field commands....2124
PredefinedField object....2125
Field object....2127
FieldState object....2131
FluidCavityPressure object....2134
FluidElectricPotential object....2136
IMAField object....2138
IMARegion object....2139

InitialState object....2140
IonConcentration object....2142
KinematicHardening object....2144
MaterialAssignment object....2146
PredefinedFieldState object....2148
SolidElectricPotential object....2149
Stress object....2151
Temperature object....2153
TemperatureState object....2157
Velocity object....2160
Print commands....2162
PrintOptions object....2163
EpsOptions object....2165
PageSetupOptions object....2167
PngOptions object....2169
PsOptions object....2170
SvgOptions object....2172
TiffOptions object....2173
Property commands....2174
SectionAssignment object....2175
CompositeLayup object....2178
CompositePly object....2181
MaterialOrientation object....2183
PlyStackPlot object....2188
Property assignment commands....2190
Region commands....2195
Assembly object....2196
Part object....2200
Region object....2204
Set object....2208
Surface object....2215
Skin object....2221
Stringer object....2223
Section commands....2225
Section object....2226
AcousticInfiniteSection object....2227
AcousticInterfaceSection object....2229
BeamSection object....2231
CohesiveSection object....2235
CompositeShellSection object....2237
CompositeSolidSection object....2240
ConnectorSection object....2242
EulerianSection object....2246
GasketSection object....2248
GeneralStiffnessSection object....2250
GeometryShellSection object....2253
HomogeneousShellSection object....2257
HomogeneousSolidSection object....2260

LayerProperties object....2262
MembraneSection object....2264
MPCSection object....2266
PEGSection object....2268
RebarLayers object....2270
SectionLayer object....2272
ShellSection object....2274
SolidSection object....2275
SurfaceSection object....2276
TransverseShearBeam object....2278
TransverseShearShell object....2280
TrussSection object....2282
Session commands....2284
Session object....2285
AutoColors object....2297
Color object....2298
Drawing object....2299
Image object....2308
JournalOptions object....2310
MemoryReductionOptions object....2311
NetworkDatabaseConnector object....2312
NumberFormat object....2315
Sketcher commands....2317
ConstrainedSketch object....2318
ConstrainedSketchConstraint object....2335
ConstrainedSketchDimension object....2341
ConstrainedSketcherOptions object....2346
ConstrainedSketchGeometry object....2348
ConstrainedSketchGeometryArray object....2356
ConstrainedSketchImageOptions object....2357
ConstrainedSketchOptions object....2359
ConstrainedSketchParameter object....2362
ConstrainedSketchVertex object....2364
ConstrainedSketchVertexArray object....2365
Step commands (step)....2366
Step object....2367
AnalysisStep object....2370
AnnealStep object....2372
BuckleStep object....2376
ComplexFrequencyStep object....2381
CoupledTempDisplacementStep object....2386
CoupledThermalElectricalStructuralStep object....2392
CoupledThermalElectricStep object....2398
CoupledThermalElectrochemicalStep object....2403
DirectCyclicStep object....2408
EmagTimeHarmonicStep object....2415
ExplicitDynamicsStep object....2419
FrequencyStep object....2424

GeostaticStep object....2431
HeatTransferStep object....2437
ImplicitDynamicsStep object....2443
InitialStep object....2450
MassDiffusionStep object....2452
ModalDynamicsStep object....2457
RandomResponseStep object....2462
ResponseSpectrumStep object....2466
SoilsStep object....2471
StaticLinearPerturbationStep object....2478
StaticRiksStep object....2482
StaticStep object....2488
SteadyStateDirectStep object....2495
SteadyStateModalStep object....2499
SteadyStateSubspaceStep object....2504
SubspaceDynamicsStep object....2509
SubstructureGenerateStep object....2513
TempDisplacementDynamicsStep object....2519
ViscoStep object....2524
Step commands (miscellaneous)....2530
CompositeDamping object....2531
CompositeDampingComponent object....2532
Control object....2533
DirectDamping object....2536
DirectDampingByFrequency object....2537
DirectDampingByFrequencyComponent object....2538
DirectDampingComponent object....2539
EmagTimeHarmonicFrequency object....2540
MassScaling object....2541
RandomResponseFrequency object....2542
RayleighDamping object....2543
RayleighDampingByFrequency object....2544
RayleighDampingByFrequencyComponent object....2545
RayleighDampingComponent object....2546
ResponseSpectrumComponent object....2547
SolverControl object....2548
SteadyStateDirectFrequency object....2549
SteadyStateModalFrequency object....2550
SteadyStateSubspaceFrequency object....2551
StepControl object....2552
StructuralDamping object....2554
StructuralDampingByFrequency object....2555
StructuralDampingByFrequencyComponent object....2556
StructuralDampingComponent object....2557
SubstructureGenerateFrequency object....2558
SubstructureGenerateModes object....2559
Step commands (output)....2560
DiagnosticPrint object....2561

FieldOutputRequest object....2563
FieldOutputRequestState object....2569
HistoryOutputRequest object....2571
HistoryOutputRequestState object....2577
IntegratedOutputSection object....2579
Monitor object....2581
Restart object....2583
TimePoint object....2585
Table Collection commands....2587
ActivateElements object....2588
DataTable object....2590
ElementProgressiveActivation object....2591
EventSeries object....2593
EventSeriesType object....2595
ParameterColumn object....2597
ParameterTable object....2599
PropertyTable object....2600
PropertyTableData object....2602
TableCollection object....2604
Text Representation commands....2605
redentABQ module....2606
textRepr module....2607
TextReprOptions object....2612
Utility and View commands....2614
SymbolicConstant object....2615
AbaqusBoolean object....2616
Repository object....2617
BackwardCompatibility object....2618
User input commands....2622
Status commands....2625
Callback commands....2627
Method callback command....2629
Delete object callback command....2632
Upgrade script commands....2634
Customization commands....2636
View object....2637
XY commands....2647
XYData object....2648
Area object....2661
AreaStyle object....2664
Axis object....2666
AxisData object....2669
Chart object....2673
CurveOptions object....2679
DefaultChartOptions object....2682
DefaultPlot object....2684
Legend object....2685
LineStyle object....2687

QuantityType object....2690
SymbolStyle object....2694
TextStyle object....2697
Title object....2699
XYCurve object....2701
XYPlot object....2703
XYReportOptions object....2707
Session object....2709
writeXYReport....2713
C++ commands....2714
Amplitude commands....2715
Amplitude object....2716
ActuatorAmplitude object....2717
BaselineCorrection object....2719
DecayAmplitude object....2721
EquallySpacedAmplitude object....2723
ModulatedAmplitude object....2725
PeriodicAmplitude object....2727
PsdDefinition object....2729
SmoothStepAmplitude object....2731
SolutionDependentAmplitude object....2733
SpectrumAmplitude object....2735
TabularAmplitude object....2738
UserAmplitude object....2740
Beam Section profile commands....2742
Profile object....2743
ArbitraryProfile object....2744
BoxProfile object....2746
CircularProfile object....2748
GeneralizedProfile object....2749
HexagonalProfile object....2751
IProfile object....2753
LProfile object....2755
PipeProfile object....2757
RectangularProfile object....2759
TProfile object....2761
TrapezoidalProfile object....2763
Connector commands....2765
ConnectorBehaviorOption object....2766
CDCTerm object....2767
ConnectorDamage object....2770
ConnectorDamping object....2776
ConnectorElasticity object....2780
ConnectorFailure object....2784
ConnectorFriction object....2786
ConnectorLock object....2791
ConnectorOptions object....2793
ConnectorPlasticity object....2796

ConnectorPotential object....2801
ConnectorStop object....2804
DerivedComponent object....2806
TangentialBehavior object....2808
Filter commands....2811
Filter object....2812
ButterworthFilter object....2813
Chebyshev1Filter object....2815
Chebyshev2Filter object....2817
OperatorFilter object....2819
Material commands....2821
Material object....2824
AcousticMedium object....2830
AnnealTemperature object....2832
BiaxialTestData object....2834
BrittleCracking object....2836
BrittleFailure object....2838
BrittleShear object....2840
CapCreepCohesion object....2842
CapCreepConsolidation object....2844
CapHardening object....2846
CapPlasticity object....2848
CastIronCompressionHardening object....2850
CastIronPlasticity object....2852
CastIronTensionHardening object....2854
ClayHardening object....2856
ClayPlasticity object....2858
CombinedTestData object....2860
Concrete object....2862
ConcreteCompressionDamage object....2864
ConcreteCompressionHardening object....2866
ConcreteDamagedPlasticity object....2868
ConcreteTensionDamage object....2870
ConcreteTensionStiffening object....2872
Conductivity object....2874
ContactArea object....2876
Creep object....2878
CrushableFoam object....2880
CrushableFoamHardening object....2882
CycledPlastic object....2884
CyclicHardening object....2886
DamageEvolution object....2888
DamageInitiation object....2893
DamageStabilization object....2919
DamageStabilizationCohesive object....2921
Damping object....2923
DeformationPlasticity object....2925
Density object....2927

Depvar object....2929
DetonationPoint object....2930
Dielectric object....2932
Diffusivity object....2935
DruckerPrager object....2938
DruckerPragerCreep object....2941
DruckerPragerHardening object....2943
Elastic object....2945
ElectricalConductivity object....2951
Eos object....2954
EosCompaction object....2958
Expansion object....2960
FailStrain object....2963
FailStress object....2965
FailureRatios object....2967
FluidLeakoff object....2969
GapConductance object....2971
GapConvection object....2973
GapFlow object....2975
GapRadiation object....2977
GasketMembraneElastic object....2979
GasketThicknessBehavior object....2981
GasketTransverseShearElastic object....2985
Gel object....2987
HeatGeneration object....2989
Hyperelastic object....2990
Hyperfoam object....2995
Hypoelastic object....2997
Hysteresis object....2999
InelasticHeatFraction object....3000
JouleHeatFraction object....3001
LatentHeat object....3002
LowDensityFoam object....3003
MagneticPermeability object....3005
MohrCoulombHardening object....3008
MohrCoulombPlasticity object....3010
MoistureSwelling object....3012
MullinsEffect object....3014
Ornl object....3015
Permeability object....3017
Piezoelectric object....3020
PlanarTestData object....3023
Plastic object....3025
PoreFluidExpansion object....3029
PorousBulkModuli object....3031
PorousElastic object....3033
PorousFailureCriteria object....3035
PorousMetalPlasticity object....3036

Potential object....3038
PressureEffect object....3040
RateDependent object....3042
Ratios object....3044
Regularization object....3046
SaturationDependence object....3047
ShearRetention object....3048
ShearTestData object....3050
SimpleShearTestData object....3052
Solubility object....3053
SoretEffect object....3055
Sorption object....3057
SpecificHeat object....3059
SuperElasticHardening object....3061
SuperElasticHardeningModifications object....3062
SuperElasticity object....3064
Swelling object....3066
TensileFailure object....3068
TensionCutOff object....3070
TensionStiffening object....3072
TriaxialTestData object....3074
Trs object....3076
UniaxialTestData object....3078
UserDefinedField object....3080
UserMaterial object....3081
UserOutputVariables object....3083
VelocityDependence object....3084
Viscoelastic object....3085
Viscosity object....3089
Viscous object....3091
VoidNucleation object....3093
VolumetricTestData object....3095
Odb commands....3097
Odb object....3098
AnalyticSurface object....3102
AnalyticSurfaceSegment object....3103
BeamOrientation object....3104
FieldBulkData object....3105
FieldLocation object....3108
FieldOutput object....3109
FieldValue object....3130
HistoryOutput object....3134
HistoryPoint object....3137
HistoryRegion object....3142
JobData object....3145
OdbAssembly object....3146
OdbDatumCsys object....3153
OdbFrame object....3157

OdbInstance object....3161
OdbLoadCase object....3172
OdbMeshElement object....3173
OdbMeshNode object....3175
OdbPart object....3176
OdbPretensionSection object....3185
OdbRigidBody object....3186
OdbSequenceAnalyticSurfaceSegment object....3187
OdbSet object....3190
OdbStep object....3198
RebarOrientation object....3204
SectionCategory object....3205
SectionPoint object....3207
SectorDefinition object....3209
UserData object....3210
UserXYData object....3213
Top level commands....3215
Property and Section commands....3222
SectionAssignment object....3223
Section object....3224
AcousticInfiniteSection object....3225
AcousticInterfaceSection object....3227
BeamSection object....3228
CohesiveSection object....3232
CompositeShellSection object....3234
CompositeSolidSection object....3237
ConnectorSection object....3239
EulerianSection object....3244
GasketSection object....3245
GeneralStiffnessSection object....3247
GeometryShellSection object....3250
HomogeneousShellSection object....3252
HomogeneousSolidSection object....3255
LayerProperties object....3256
MembraneSection object....3258
MPCSection object....3260
PEGSection object....3262
RebarLayers object....3264
SectionLayer object....3265
ShellSection object....3267
SolidSection object....3268
SurfaceSection object....3269
TransverseShearBeam object....3271
TransverseShearShell object....3273
TrussSection object....3274
Infrastructure commands....3275
BaseException object....3276
Repository object....3277

RepositoryIterator object....3279
Sequence object....3282
String object....3286
Union object....3288

Trademarks and Legal Notices

Trademarks

Abaqus, **3D**EXPERIENCE®, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

Legal Notices

Abaqus and this documentation may be used or reproduced only in accordance with the terms of the software license agreement signed by the customer, or, absent such agreement, the then current software license agreement to which the documentation relates.

This documentation and the software described in this documentation are subject to change without prior notice.

Dassault Systèmes or its Affiliates shall not be responsible for the consequences of any errors or omissions that may appear in this documentation.

© Dassault Systèmes Americas Corp., 2025.

For a full list of the third-party software contained in this release, please go to the Legal Notices in the Abaqus 2025 HTML documentation, which can be obtained from a documentation installation, or in the SIMULIA Established Products 2025 Program Directory, which is available on *www.3ds.com*.

What's New

This page describes recent changes in Abaqus Scripting Reference.

2025 FD02

Python Commands

New and revised Python objects are available.

Benefits: The features extend the capabilities of the Python commands.

Table 1: New Python Command Objects

Command	Object
Boundary Condition	*ElectricPotentialElectrolyteBC object*
Boundary Condition	*ElectricPotentialElectrolyteBCState object*
Boundary Condition	*IonConcentrationBC object*
Boundary Condition	*IonConcentrationBCState object*
Load and Load Case	*DistIonConcBodyFlux object*
Load and Load Case	*DistIonConcBodyFluxState object*
Load and Load Case	*FluidSurfaceCurrent object*
Load and Load Case	*FluidSurfaceCurrentState object*
Load and Load Case	*IonConcentrationFluxState object*
Predefined field	*FluidElectricPotential object*
Predefined field	*IonConcentration object*
Step Commands (step)	*CoupledThermalElectrochemicalStep object*
Step Commands (miscellaneous)	*StepControl object*
Step Commands (miscellaneous)	*CoupledThermalElectrochemicalStep object*

Table 2: Revisions to Python Command Objects

Command	Object	Revisions
Constraint	*Coupling object*	The required argument **coupling*Type*** can now take the value of UNIFORM, and the new optional arguments **uniformConstraint** and **temperatureDOFType** are added.
Constraint	*Tie object*	Optional arguments **tieTemperature**, **tiePore**, **electricPotential**,

Command	Object	Revisions
		fluidElectricPotential, and **ionConcentration** are now available.
Interaction	*FilmCondition object*	Optional argument **dynamicSurfaceActivation** is now available.
Interaction	*GapElectricalConductance object*	Optional arguments **Type**, **ionConcentrationDependencyC**, and **ionConcentrationDependencyP** are now available.
Interaction	*GapHeatGeneration object*	Optional arguments **mechanicalInteractionFraction**, **mechanicalInteractionWeightingFactor**, **electricalConductionSolidsFraction**, **electricalConductionSolidsWeightingFactor**, **electricalConductionLiquidsFraction**, and **electricalConductionLiquidsWeightingFactor** are now available.

2025 FD01

Animation File Formats

You can use the new MP4Options and GIFOptions objects in the Animation commands to save and load animations in MP4 and animated GIF formats.

Benefits: New file formats offer improved compatibility with modern media players and editing software, as well as HTML5 and third-party applications. File size can be significantly smaller, with no appreciable loss of quality.
You can save MP4 animations with the MPEG-4 codec. If you have an installation of the FFmpeg program that supports H.264 encoding on your system, you can save MP4 animations using the H.264 codec.

For more information, see *GIFOptions object*, *MP4Options object*, *Choosing the animation file format*, and *Using background images and movies*.

PDF Guides Available

You can download PDF versions of the Abaqus guides from each guide overview.

Benefits: You can easily access the PDF versions of the guides.
In earlier releases, PDF versions of the guides were updated only for the GA (General Availability) release and could be downloaded from the Dassault Systèmes Knowledge Base. As of 2025 FD01, the PDF guides will be updated with each release of the HTML versions of the guides.

For more information, see *Abaqus Scripting Reference Guide*.

2025 GA

Python Interactions Commands: FluidExchangeActivation Object

The FluidExchangeActivation object is now available in the Interactions profile commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *FluidExchangeActivation object*.

Python Interactions Commands: FluidExchangeActivationState Object

The FluidExchangeActivationState object is now available in the Interactions profile commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *FluidExchangeActivationState object*.

Python Interactions Commands: FluidInflator Object

The required arguments inflationTimeAmplitude and massFlowAmplitude are no longer available in the FluidInflator object.

Benefits: The functionality was extended, with these arguments appropriately available in the FluidInflatorActivation object instead.

For more information, see *FluidInflator object*.

Python Interactions Commands: FluidInflatorActivation Object

The FluidInflatorActivation object is now available in the Interactions profile commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *FluidInflatorActivation object*.

Python Interactions Commands: FluidInflatorActivationState Object

The FluidInflatorActivationState object is now available in the Interactions commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *FluidInflatorActivationState object*.

Python Interactions Commands: WearProperty Object

The WearProperty object is now available in the Interactions commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *WearProperty object*.

Python Interactions Commands: WearSurfacePropertyAssignment Object

The WearSurfacePropertyAssignment object is now available in the Interactions commands.

Benefits: The feature extends the capabilities of the Interactions commands.

For more information, see *WearSurfacePropertyAssignment object*.

Python Load and Load Case Commands: RotationalBodyForce Object

The new optional argument rotorDynamicloads is available in the RotationalBodyForce object.

Benefits: The feature extends the capabilities of the Load and Load Case commands.

For more information, see *RotationalBodyForce object*.

Python Path and Probe Commands: FreeBody Objects

The method getFreeBodyData() is now available in the FreeBody objects.

Benefits: The feature extends the capabilities of the Path and Probe commands.

For more information, see *FreeBody object*.

2024 FD03

New Method getMemberFromAll for Node and Element Sequences

There is a new Python API on node and element sequences to retrieve a member value for each item in the sequence. This applies to MeshNodeArray, MeshElementArray, MeshFaceArray, ODBMeshNodeArray, and ODBMeshElementArray.

Benefits: Retrieving a member from a large sequence (for example, the node label for every node) is much faster using this method than iterating over the sequence.

For more information, see *MeshNodeArray object*, *MeshElementArray object*, *MeshFaceArray object*, *OdbMeshNode object*, and *OdbMeshElement object*.

2024 GA

Python 3

Abaqus Python and Abaqus/CAE are now based on Python 3.10 instead of Python 2.7.

Benefits: This update brings Abaqus Python into conformance with a version of Python that is supported in the open-source community.

For more information, see *https://go.3ds.com/AbaqusPython*.

Python Beam Section Profile Commands: ChannelProfile Object

The ChannelProfile object is now available in the Beam Section profile commands.

Benefits: The feature extends the capabilities of the Beam Section profile commands.

For more information, see *ChannelProfile object*.

Python Beam Section Profile Commands: HatProfile Object

The HatProfile object is now available in the Beam Section profile commands.

Benefits: The feature extends the capabilities of the Beam Section profile commands.

For more information, see *HatProfile object*.

Python Constraint Commands: Coupling Object

The Coupling object is now available in the Constraint commands.

Benefits: The feature extends the capabilities of the Constraint commands.

For more information, see *Coupling object*.

Python Engineering Feature Commands: DiscreteFastener Object

The optional argument *rotatonalCoupling* is now available in the DiscreteFastener object.

Benefits: The feature extends the capabilities of the Engineering Feature commands.

For more information, see *DiscreteFastener object*.

Python Interaction Commands: ContactMassScalingExp Object

The ContactMassScalingExp object is now available in the Interaction commands.

Benefits: The feature extends the capabilities of the Interaction commands.

For more information, see *ContactMassScalingExp object*.

Python Job Commands: OptimizationProcess Object

The optional argument licenseType is now available in the OptimizationProcess object.

Benefits: The feature extends the capabilities of the Job commands.

For more information, see *ContactMassScalingExp object*.

Python Odb Commands: OdbAssembly Object

The method getMassProperties is now available in the OdbAssembly object.

Benefits: The feature extends the capabilities of the Odb commands.

For more information, see *OdbAssembly object*.

Python Optimization Commands: SingleTermDesignResponse Object

The optional argument stepOperation is no longer available in the SingleTermDesignResponse object.

Benefits: The functionality was removed from the Optimization commands: SingleTermDesignResponse object.

For more information, see *SingleTermDesignResponse object*.

Python Section Commands: BeamSection Object

The optional argument beamSectionOffset is now available in the BeamSection object.

Benefits: The feature extends the capabilities of the Section commands.

For more information, see *BeamSection object*.

Python Step Commands (Step): StaticLinearPerturbationStep Object

The optional arguments solutionTechnique, gapDistance, and scaleFactor are now available in the StaticLinearPerturbationStep object.

Benefits: The feature extends the capabilities of the Step commands (step).

For more information, see *StaticLinearPerturbationStep object*.

Python XY Commands: Session Object

The optional argument useMaxStressValue is now available in the Session object.

Benefits: The feature extends the capabilities of the XY commands.

For more information, see *Session object*.

Abaqus Scripting Reference Guide

The Abaqus Scripting Reference Guide contains a complete description of each command in the Abaqus Scripting Interface.

This guide is a part of the Abaqus® documentation collection, which describes all the capabilities of the Abaqus finite element analysis technology used in SIMULIA® applications.

Python commands

Python commands are used to read and write data from an output database.

In this section:

- *Abaqus/CAE Display Preferences commands*
- *Adaptivity commands*
- *Amplitude commands*
- *Animation commands*
- *Annotation commands*
- *Assembly commands*
- *Basic geometry commands*
- *Beam Section profile commands*
- *Boundary Condition commands*
- *Calibration commands*
- *Canvas commands*
- *Connector commands*
- *Constraint commands*
- *CustomKernel module*
- *Datum commands*
- *Display Group commands*
- *Display Options commands*
- *Edit mesh commands*
- *Engineering Feature commands*
- *Feature commands*
- *Field commands*
- *Field Report commands*
- *Filter commands*
- *Input File Parser commands*
- *Interaction commands*
- *Job commands*
- *Load and Load Case commands*
- *Material commands*
- *Mdb commands*
- *Mesh commands*
- *Messaging commands*
- *Model commands*
- *Odb commands*
- *Odb Display commands*
- *Optimization commands*
- *Part commands*
- *Partition commands*

- *Path and Probe commands*
- *Plot Options commands*
- *Plug-in registration commands*
- *Predefined field commands*
- *Print commands*
- *Property commands*
- *Region commands*
- *Section commands*
- *Session commands*
- *Sketcher commands*
- *Step commands (step)*
- *Step commands (miscellaneous)*
- *Step commands (output)*
- *Table Collection commands*
- *Text Representation commands*
- *Utility and View commands*
- *XY commands*

Abaqus/CAE Display Preferences commands

Display preferences commands are used to edit the Abaqus/CAE preferences file, `abaqus_2025.gpr`.

In this section:

- *caePrefsAccess module*
- *CaeGuiPrefs object*
- *CaeKerPrefs object*

caePrefsAccess module

The Python module `caePrefsAccess` contains functions that enable you to edit the Abaqus/CAE preferences file, `abaqus_2025.gpr`.

Access

```
import caePrefsAccess
```

getGuiPrefsFileName(...)

This function enables you to retrieve the location of your `abaqus_2025.gpr` file.

Path

```
caePrefsAccess.getGuiPrefsFileName
```

Arguments

None.

Return value

A String with the default file name for the GUI preferences file.

getDisplayNamesInGuiPreferences(...)

The `abaqus_2025.gpr` file stores a separate `guiPreferences` record for each display that you use. This function returns a list of every displayName recorded in the preferences file.

Path

```
caePrefsAccess.getDisplayNamesInGuiPreferences
```

Required arguments

fileName	A String that specifies the path to the preferences file.

Optional arguments

None.

Return value

A list of Strings of displayNames.

Exceptions

None.

printValuesList(...)

This function enables you to print all of the options and their values for a set of `guiPreferences` or `sessionOptions` settings derived from the `abaqus_2025.gpr` file.

Path

```
caePrefsAccess.printValuesList
```

Required arguments

object
The `guiPreferences` object or `sessionOptions` object for which you want to print options and their values.

Optional arguments

maxRecursionDepth
An Int, or SymbolicConstant UNLIMITED, that specifies the depth of recursion when accessing the attributes of *object*.

asString
A Boolean specifying how the string representation of each option is printed. If *asString* is True, printValuesList prints the `str` of each option; otherwise printValuesList prints the `repr` of the options. The default value is False.

Return value

A String displaying the path, name, and value for all of the options in the object that you select.

Exceptions

None.

openGuiPreferences(...)

This function enables you to examine and change many default behaviors in the Abaqus/CAE graphical user interface. Abaqus stores preferences for each display you use in a separate `guiPreferences` section of the `abaqus_2025.gpr` file.

Path

```
caePrefsAccess.openGuiPreferences
```

Required arguments

displayName
A String that specifies the display for which you want to investigate GUI preferences from the `abaqus_2025.gpr` file. You can retrieve the available display names in the file by using the `getDisplayNamesInGuiPreferences` method.

Optional arguments

fileName

A String specifying the path to the preferences file. The `openGuiPreferences` method uses this argument if you are working with a preferences file that is not at the default location.

If this argument is omitted, the `abaqus_2025.gpr` file in your home directory is opened.

Return value

A CaeGuiPrefs object.

Exceptions

None.

openSessionOptions(...)

This function enables you to examine and change the default behavior for many session options Abaqus/CAE; that is, the settings that you can save in Abaqus/CAE from the **File->Save Display Options** menu option. Abaqus stores default session options in the `sessionOptions` section of the `abaqus_2025.gpr` file.

Path

```
caePrefsAccess.openSessionOptions
```

Required arguments

None.

Optional arguments

fileName

A String specifying the path to the preferences file. The `openSessionOptions` method uses this argument if you are working with a preferences file that is not at the default location.

If this argument is omitted, the `abaqus_2025.gpr` file in your home directory is opened.

directory

A SymbolicConstant specifying the location of the preferences file. Possible values are:

- CURRENT to open the preferences file in the current directory (`caePrefsAccess`.CURRENT)
- HOME to open the preferences file in your home directory (`caePrefsAccess`.HOME)

The default value is HOME. Either *fileName* or *directory* must be supplied. The *fileName* or *directory* arguments are mutually exclusive.

Return value

A CaeKerPrefs object.

Exceptions

None.

CaeGuiPrefs object

The CaeGuiPrefs object contains the details of the graphical preferences in a `guiPreferences` section of the `abaqus_2025.gpr` file.

Access

```
import caePrefsAccess
caePrefsAccess.openGuiPreferences(...)
```

`save(...)`

This method saves the `guiPreferences` section specified in the current *fileName*.

Required arguments

None.

Optional arguments

backupFile A Boolean specifying whether Abaqus should save a numbered backup copy of the preferences file, *fileName*. Default is True.

Return value

`None.`

Exceptions

None.

`saveAs(...)`

This method saves the `guiPreferences` settings to the specified location.

Required arguments

fileName A String specifying the path to the preferences file.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The CaeGuiPrefs object has the following member:

fileName
A String specifying the path to the preferences file.

CaeKerPrefs object

The CaeKerPrefs object contains the details of the `sessionOptions`.

Access

```
import caePrefsAccess
caePrefsAccess.openSessionOptions(...)
```

`save(...)`

This method saves the `sessionOptions` in the current *fileName*.

Required arguments

None.

Optional arguments

backupFile A Boolean specifying whether save a numbered backup copy of the preferences file, *fileName*. Default is True.

Return value

`None.`

Exceptions

None.

`saveAs(...)`

This method saves the `sessionOptions` to the specified location.

Required arguments

fileName A String specifying the path to the preferences file.

directory A SymbolicConstant specifying the location of the preferences file. Possible values are:

- CURRENT to open the preferences file in the current directory (`caePrefsAccess`.CURRENT)
- HOME to open the preferences file in your home directory (`caePrefsAccess`.HOME)

The default value is HOME. Either *fileName* or *directory* must be supplied. The *fileName* or *directory* arguments are mutually exclusive.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The CaeKerPrefs object has the following member:

fileName A String specifying the path to the preferences file that the CaeKerPrefs object represents.

Adaptivity commands

The Adaptivity commands are used to define objects, perform analyses, and calculate new meshes for Arbitrary Lagrangian Eularian (ALE) adaptive smoothing (adaptive meshing) and varying topology adaptivity (adaptive remeshing).

In this section:

- *AdaptiveMeshConstraint object*
- *AdaptiveMeshConstraintState object*
- *AdaptiveMeshControl object*
- *AdaptiveMeshDomain object*
- *AdaptivityIteration object*
- *AdaptivityProcess object*
- *DisplacementAdaptiveMeshConstraint object*
- *DisplacementAdaptiveMeshConstraintState object*
- *ErrorIndicatorResult object*
- *Model object*
- *RemeshingRule object*
- *RuleResult object*
- *VelocityAdaptiveMeshConstraint object*
- *VelocityAdaptiveMeshConstraintState object*

AdaptiveMeshConstraint object

The AdaptiveMeshConstraint object is the abstract base type for other Arbitrary Lagrangian Eularian (ALE) style AdaptiveMeshConstraint objects. The AdaptiveMeshConstraint object has no explicit constructor. The methods and members of the AdaptiveMeshConstraint object are common to all objects derived from the AdaptiveMeshConstraint object.

Access

```
import step
mdb.models[name].adaptiveMeshConstraints[name]
```

deactivate(...)

This method deactivates the adaptive mesh constraint in the specified step and all subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the adaptive mesh constraint is deactivated.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

move(...)

This method moves the adaptive mesh constraint state from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the adaptive mesh constraint state is moved.

toStepName A String specifying the name of the step to which the adaptive mesh constraint state is moved.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

reset(...)

This method resets the adaptive mesh constraint state of the specified step to the state of the previous analysis step.

Required arguments

stepName	A String specifying the name of the step in which the adaptive mesh constraint state is reset.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

resume()

This method resumes the adaptive mesh constraint that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

suppress()

This method suppresses the adaptive mesh constraint.

Arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

This method allows you to delete existing adaptive mesh constraints.

Required arguments

indices A sequence of Ints specifying the index of each adaptive mesh constraint to delete.

Optional arguments

None.

Return value

`None`.

Exceptions

None.

Members

The AdaptiveMeshConstraint object can have the following members:

name A String specifying the adaptive mesh constraint repository key.

category A SymbolicConstant specifying the category of the adaptive mesh constraint. Possible values are MECHANICAL and THERMAL.

region A *Region* object specifying the region to which the adaptive mesh constraint is applied.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the adaptive mesh constraint's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

AdaptiveMeshConstraintState object

The AdaptiveMeshConstraintState object is the abstract base type for other Arbitrary Lagrangian Eularian (ALE) style AdaptiveMeshConstraintState objects. The AdaptiveMeshConstraintState object has no explicit constructor or methods. The members of the AdaptiveMeshConstraintState object are common to all objects derived from the AdaptiveMeshConstraintState object.

Access

```
import step
mdb.models[name].steps[name].adaptiveMeshConstraintStates[name]
```

Members

The AdaptiveMeshConstraintState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the AdaptiveMeshConstraintState object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the adaptive mesh constraint has no amplitude reference.

AdaptiveMeshControl object

The AdaptiveMeshControl object is used to control various aspects of Arbitrary Lagrangian Eularian (ALE) style adaptive smoothing and advection algorithms applied to an ALE adaptive mesh domain.

Access

```
import step
mdb.models[name].adaptiveMeshControls[name]
```

AdaptiveMeshControl(...)

This method creates an AdaptiveMeshControl object.

Path

```
mdb.models[name].AdaptiveMeshControl
```

Required arguments

name
A String specifying the name of the object.

Optional arguments

remapping
A SymbolicConstant specifying the remapping algorithm. Possible values are FIRST_ORDER_ADVECTION and SECOND_ORDER_ADVECTION. The default value is SECOND_ORDER_ADVECTION.

smoothingAlgorithm
A SymbolicConstant specifying the type of smoothing algorithm to use. Possible values are STANDARD and GEOMETRY_ENHANCED. The default value is GEOMETRY_ENHANCED.

smoothingPriority
A SymbolicConstant specifying the type of smoothing to perform. Possible values are UNIFORM and GRADED. The default value is UNIFORM.

initialFeatureAngle
A Float specifying the initial geometric feature angle, $\boldsymbol{\theta_I}$, in degrees. Possible values are $0° \leq \boldsymbol{\theta_I} \leq 180°$. The default value is 30.0.

transitionFeatureAngle
A Float specifying the transitional feature angle, $\boldsymbol{\theta_T}$, in degrees. Possible values are $0° \leq \boldsymbol{\theta_T} \leq 180°$. The default value is 30.0.

momentumAdvection
A SymbolicConstant specifying the type of momentum advection algorithm. Possible values are ELEMENT_CENTER_PROJECTION and HALF_INDEX_SHIFT. The default value is ELEMENT_CENTER_PROJECTION.

meshingPredictor
A SymbolicConstant specifying the nodal starting location to use for remeshing. Possible values are CURRENT and PREVIOUS. The default value is CURRENT.

curvatureRefinement
A Float specifying the solution dependence weight, α_C. Possible values are $0.0 \le \alpha_C \le 1.0$. The default value is 1.0.

volumetricSmoothingWeight
A Float specifying the weight used by Abaqus/Explicit for the volumetric smoothing method. The default value is 1.0.

laplacianSmoothingWeight
A Float specifying the weight for the Laplacian smoothing method. The default value is 0.0.

equipotentialSmoothingWeight
A Float specifying the weight for the equipotential smoothing method. The default value is 0.0.

meshConstraintAngle
A Float specifying the initial geometric feature angle, θ_C. Possible values are $0° \le \theta_C \le 180°$. The default value is 60.0.

originalConfigurationProjectionWeight
A Float specifying the weight for the original configuration projection method. The default value is 1.0.

standardVolumetricSmoothingWeight
A Float specifying the weight used by Abaqus/Standard for the volumetric smoothing method. The default value is 0.0.

Return value

An AdaptiveMeshControl object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the AdaptiveMeshControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AdaptiveMeshControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The AdaptiveMeshControl object has members with the same names and descriptions as the arguments to the *AdaptiveMeshControl* method.

AdaptiveMeshDomain object

The AdaptiveMeshDomain object defines the region and controls that govern an Arbitrary Lagrangian Eularian (ALE) style adaptive smoothing mesh domain.

Access

```
import step
mdb.models[name].steps[name].adaptiveMeshDomains[name]
```

AdaptiveMeshDomain(...)

This method creates an AdaptiveMeshDomain object.

Path

```
mdb.models[name].steps[name].AdaptiveMeshDomain
```

Required arguments

region A *Region* object specifying the region to which the adaptive mesh domain is applied.

Optional arguments

controls A String specifying the name of an *AdaptiveMeshControl* object.

frequency An Int specifying the frequency in increments at which adaptive meshing will be performed. The default value is 10.

initialMeshSweeps An Int specifying the number of mesh sweeps to be performed at the beginning of the first step in which this adaptive mesh definition is active. The default value is 5.

meshSweeps An Int specifying the number of mesh sweeps to be performed in each adaptive mesh increment. The default value is 1.

Return value

An AdaptiveMeshDomain object.

Exceptions

RangeError.

setValues(...)

This method modifies the AdaptiveMeshDomain object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AdaptiveMeshDomain* method.

Return value

`None.`

Exceptions

RangeError.

Members

The AdaptiveMeshDomain object has members with the same names and descriptions as the arguments to the *AdaptiveMeshDomain* method.

AdaptivityIteration object

The AdaptivityIteration object contains information about a given iteration of the varying topology adaptivity process (adaptive remeshing).

Access

```
import job
mdb.adaptivityProcesses[name].iterations[i]
```

AdaptivityIteration(...)

This method creates an AdaptivityIteration object.

Path

```
mdb.adaptivityProcesses[name].AdaptivityIteration
```

Required arguments

iteration	An Int specifying the sequence number for this iteration in the adaptivity process.
jobName	A String specifying the name of the job that was run for this iteration.
modelName	A String specifying the name of the model that was analyzed and remeshed in this iteration.
odbPath	A String specifying the path to the ODB file that was created for this iteration.
remeshingErrors	An Int specifying the number of part instances which generated errors while remeshing the model in this iteration of the adaptivity process.

Optional arguments

None.

Return value

An AdaptivityIteration object.

Exceptions

None.

Members

The AdaptivityIteration object has members with the same names and descriptions as the arguments to the *AdaptivityIteration* method.

In addition, the AdaptivityIteration object can have the following member:

ruleResults

A repository of *RuleResult* objects specifying the calculated results from sizing functions corresponding to the *RemeshingRule* objects for this iteration of an adaptivity process.

AdaptivityProcess object

The AdaptivityProcess object defines a series of jobs that will be submitted for analysis. Abaqus performs adaptive remeshing between each job.

Access

```
import job
mdb.adaptivityProcesses[name]
```

AdaptivityProcess(...)

This method creates an AdaptivityProcess object.

Path

```
mdb.AdaptivityProcess
```

Required arguments

name A String specifying the name of the Adaptivity Process.

job A *ModelJob* object specifying a job to be used as the prototype for all analysis jobs run by the adaptivity process.

Optional arguments

maxIterations An Int specifying the maximum number of analysis jobs that will be run by the Adaptivity Process. Abaqus performs adaptive remeshing between each analysis. The default value is 3.

jobPrefix A String specifying the prefix to use for jobs created by the adaptivity process. An empty string indicates that the name of the adaptivity process should be used. The default value is an empty string.

Return value

An AdaptivityProcess object.

Exceptions

AbaqusException.

submit(...)

This method begins the process of analysis and adaptive remeshing.

Required arguments

None.

Optional arguments

waitForCompletion A Boolean specifying whether to interrupt the execution of a script until the end of the adaptive remeshing process is reached.

datacheckJob A Boolean specifying whether to run the adaptivity as a datacheck analysis. The default value is False. The datacheckJob and continueJob arguments cannot both be True.

continueJob A Boolean specifying whether to run the adaptivity as a continuation analysis. The default value is False. The datacheckJob and continueJob arguments cannot both be True.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the AdaptivityProcess object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AdaptivityProcess* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The AdaptivityProcess object has members with the same names and descriptions as the arguments to the *AdaptivityProcess* method.

In addition, the AdaptivityProcess object can have the following members:

status A SymbolicConstant specifying the status of the adaptivity process. Possible values are SUBMITTED, RUNNING, ABORTED, TERMINATED, and COMPLETED.

iterations A repository of *AdaptivityIteration* objects specifying the *AdaptivityIteration* objects received during running the adaptivity process.

DisplacementAdaptiveMeshConstraint object

The DisplacementAdaptiveMeshConstraint object stores the data for an Arbitrary Lagrangian Eularian (ALE) style displacement/rotation adaptive mesh constraint.

The DisplacementAdaptiveMeshConstraint object is derived from the *AdaptiveMeshConstraint* object.

Access

```
import step
mdb.models[name].adaptiveMeshConstraints[name]
```

DisplacementAdaptiveMeshConstraint(...)

This method creates a DisplacementAdaptiveMeshConstraint object.

Path

```
mdb.models[name].DisplacementAdaptiveMeshConstraint
```

Required arguments

name
A String specifying the adaptive mesh constraint repository key.

createStepName
A String specifying the name of the step in which the adaptive mesh constraint is created.

region
A *Region* object specifying the region to which the adaptive mesh constraint is applied.

Optional arguments

u1
A Float or a SymbolicConstant specifying the displacement component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *u1*, *u2*, *u3*, *ur1*, *ur2*, and *ur3* are optional arguments, at least one of them must be specified.

u2
A Float or a SymbolicConstant specifying the displacement component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

u3
A Float or a SymbolicConstant specifying the displacement component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur1
A Float or a SymbolicConstant specifying the rotational displacement component about the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur2
A Float or a SymbolicConstant specifying the rotational displacement component about the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur3
A Float or a SymbolicConstant specifying the rotational displacement component about the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the adaptive mesh constraint has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

motionType
A SymbolicConstant specifying the mesh motion in relation to the underlying material. Possible values are INDEPENDENT, FOLLOW and USER_DEFINED. The default value is INDEPENDENT.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the adaptive mesh constraint's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A DisplacementAdaptiveMeshConstraint object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing DisplacementAdaptiveMeshConstraint object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DisplacementAdaptiveMeshConstraint* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing DisplacementAdaptiveMeshConstraint object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the adaptive mesh constraint is modified.

Optional arguments

u1 A Float or a SymbolicConstant specifying the displacement component in the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u2 A Float or a SymbolicConstant specifying the displacement component in the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u3 A Float or a SymbolicConstant specifying the displacement component in the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur1 A Float or a SymbolicConstant specifying the rotational displacement component about the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur2 A Float or a SymbolicConstant specifying the rotational displacement component about the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur3 A Float or a SymbolicConstant specifying the rotational displacement component about the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the adaptive mesh constraint is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The DisplacementAdaptiveMeshConstraint object can have the following members:

name A String specifying the adaptive mesh constraint repository key.

category A SymbolicConstant specifying the category of the adaptive mesh constraint. Possible values are MECHANICAL and THERMAL.

region A *Region* object specifying the region to which the adaptive mesh constraint is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the adaptive mesh constraint's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

DisplacementAdaptiveMeshConstraintState object

The DisplacementAdaptiveMeshConstraintState object stores the propagating data for an Arbitrary Lagrangian Eularian (ALE) style displacement/rotation adaptive mesh constraint in a step. One instance of this object is created internally by the *DisplacementAdaptiveMeshConstraint* object for each step. The instance is also deleted internally by the *DisplacementAdaptiveMeshConstraint* object.

The DisplacementAdaptiveMeshConstraintState object has no constructor or methods.

The DisplacementAdaptiveMeshConstraintState object is derived from the *AdaptiveMeshConstraintState* object.

Access

```
import step
mdb.models[name].steps[name].adaptiveMeshConstraintStates[name]
```

Members

The DisplacementAdaptiveMeshConstraintState object has the following members:

u1
A Float or a Complex specifying the displacement component in the 1-direction.

u2
A Float or a Complex specifying the displacement component in the 2-direction.

u3
A Float or a Complex specifying the displacement component in the 3-direction.

ur1
A Float or a Complex specifying the rotational displacement component about the 1-direction.

ur2
A Float or a Complex specifying the rotational displacement component about the 2-direction.

ur3
A Float or a Complex specifying the rotational displacement component about the 3-direction.

u1State
A SymbolicConstant specifying the propagation state of the displacement component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u2State
A SymbolicConstant specifying the propagation state of the displacement component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u3State
A SymbolicConstant specifying the propagation state of the displacement component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur1State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur2State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur3State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status A SymbolicConstant specifying the propagation state of the *AdaptiveMeshConstraintState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude A String specifying the name of the amplitude reference. The String is empty if the adaptive mesh constraint has no amplitude reference.

Corresponding analysis keywords

- **ADAPTIVE MESH CONSTRAINT*, TYPE=DISPLACEMENT (degree of freedom: 1, 2, 3, 4, 5, or 6)

ErrorIndicatorResult object

The ErrorIndicatorResult object contains result information corresponding to an error indicator variable in a *RemeshingRule* object for an adaptivity iteration.

Access

```
import job
mdb.adaptivityProcesses[name].iterations[i].ruleResults[name]\
.indicatorResults[name]
```

ErrorIndicatorResult(...)

This method creates an ErrorIndicatorResult with data for an error indicator variable in a *RemeshingRule* for a given adaptivity iteration.

Path

```
mdb.adaptivityProcesses[name].iterations[i].ruleResults[name]\
.ErrorIndicatorResult
```

Required arguments

name A String specifying the name of the error indicator variable to which these results correspond.

results A String-to-Float Dictionary specifying the calculated results from the sizing function corresponding to the error indicator variable represented by this ErrorIndicatorResult.

Optional arguments

None.

Return value

An ErrorIndicatorResult object.

Exceptions

AbaqusException.

Members

The ErrorIndicatorResult object has members with the same names and descriptions as the arguments to the *ErrorIndicatorResult* method.

Model object

The following commands operate on Model objects. For more information about the Model object, see *Model object*.

Access

```
import mesh
```

`adaptiveRemesh(...)`

This method remeshes the model using the active remesh rules in the model and the error indicator results from a previous analysis.

Required arguments

odb An *Odb* object containing error output field results.

Optional arguments

None.

Return value

An *AdaptivityIteration* Object.

Exceptions

None.

RemeshingRule object

The RemeshingRule object controls the adaptive remeshing resizing and the error indicators written to the output database for a specified region of the model.

Access

```
import mesh
mdb.models[name].remeshingRules[name]
```

RemeshingRule(...)

This method creates a RemeshingRule object.

Path

```
mdb.models[name].RemeshingRule
```

Required arguments

name
A String specifying the name of the object.

stepName
A String specifying the name of the step in which resizing should occur for this rule.

variables
A sequence of Strings specifying the output request variables that Abaqus will use as error indicators.

Optional arguments

description
A String specifying a descriptive string for this rule. The default value is an empty string.

region
The SymbolicConstant MODEL or a *Region* object specifying the region in which Abaqus will remesh and generate output. The SymbolicConstant MODEL represents the entire applicable model. The default value is MODEL.

sizingMethod
A SymbolicConstant specifying the method for calculating the new mesh sizes. The SymbolicConstant DEFAULT indicates that Abaqus will use the default calculation method for each individual variable. Possible values are DEFAULT, UNIFORM_ERROR, and MINIMUM_MAXIMUM. The default value is DEFAULT.

errorTarget
A Float specifying the target error percentage for each variable in the region. A value of 0.0 indicates that Abaqus will use automated error target reduction for the region. You use the *errorTarget* argument when *sizingMethod*=UNIFORM_ERROR. The default value is 0.0.

maxSolutionErrorTarget
A Float specifying the target error percentage at the location of the maximum solution value in the region. A value of 0.0 indicates that Abaqus will use automated error target reduction for the region. You use the *maxSolutionErrorTarget* argument when *sizingMethod*=MINIMUM_MAXIMUM. The default value is 0.0.

minSolutionErrorTarget A Float specifying the target error percentage at the location of the minimum solution value in the region. A value of 0.0 indicates that Abaqus will use automated error target reduction for the region. You use the *minSolutionErrorTarget* argument when *sizingMethod*=MINIMUM_MAXIMUM. The default value is 0.0.

meshBias An Int specifying an indication of how much Abaqus will bias the mesh toward the location of the maximum solution value in the region. The higher the value, the more the mesh will bias towards the location of the maximum solution value. You use the *meshBias* argument when *sizingMethod*=MINIMUM_MAXIMUM. The default value is 0.0.

minElementSize A Float specifying the minimum size of any single element. The default value is 0.0.

maxElementSize A Float specifying the maximum size of any single element. The default value is 0.0.

outputFrequency A SymbolicConstant specifying the frequency with which the error indicators are saved to the output database file (`.odb`). Possible values are LAST_INCREMENT and ALL_INCREMENTS. The default value is LAST_INCREMENT.

specifyMinSize A Boolean specifying an indication of whether to use a user-supplied minimum element size or to calculate a characteristic minimum element size. The default value is OFF.

specifyMaxSize A Boolean specifying an indication of whether to use a user-supplied maximum element size or to calculate a characteristic maximum element size. The default value is ON.

coarseningFactor A SymbolicConstant or an Int specifying an indication of the upper limit on the element growth from one remeshing iteration to the next. Possible values are DEFAULT_LIMIT and NOT_ALLOWED. The default value is DEFAULT_LIMIT.

refinementFactor A SymbolicConstant or an Int specifying an indication of the upper limit on element shrinkage from one remeshing iteration to the next. Possible values are DEFAULT_LIMIT and NOT_ALLOWED. The default value is DEFAULT_LIMIT.

elementCountLimit `None` or an Int specifying an approximate limit on the number of elements that will be created during remeshing. Use `None` to indicate there is not upper limit. The default value is `None`.

Return value

A RemeshingRule object.

Exceptions

AbaqusException.

`resume()`

This method resumes the remeshing rule that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the remeshing rule. Abaqus will not remesh regions where the rules are suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the RemeshingRule object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RemeshingRule* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The RemeshingRule object has members with the same names and descriptions as the arguments to the *RemeshingRule* method.

In addition, the RemeshingRule object has the following member:

suppressed

A Boolean specifying whether the remeshing rule is suppressed. Remeshing of the remeshing rule's region will not occur if you suppress a rule. The default value is OFF.

RuleResult object

The RuleResult object contains result information corresponding to a *RemeshingRule* object for an adaptivity iteration.

Access

```
import job
mdb.adaptivityProcesses[name].iterations[i].ruleResults[name]
```

RuleResult(...)

This method creates a RuleResult with data for a *RemeshingRule* for a given adaptivity iteration.

Path

```
mdb.adaptivityProcesses[name].iterations[i].RuleResult
```

Required arguments

name
A String specifying the name of the Remeshing Rule to which these results correspond.

indicatorResults
A repository of *ErrorIndicatorResult* objects specifying the calculated results from the sizing function corresponding to the error indicator variables for the Remeshing Rule.

numElems
An Int specifying the number of elements before remeshing in the region of the Remeshing Rule.

minSizeElemCount
An Int specifying the number of elements that were constrained to the minimum element size by the Remeshing Rule.

Optional arguments

satisfiedVars
A sequence of Strings specifying the error indicator variables that have satisfied the Remeshing Rule.

Return value

A RuleResult object.

Exceptions

AbaqusException.

Members

The RuleResult object has members with the same names and descriptions as the arguments to the *RuleResult* method.

VelocityAdaptiveMeshConstraint object

The VelocityAdaptiveMeshConstraint object stores the data for an Arbitrary Lagrangian Eularian (ALE) style velocity adaptive mesh constraint.

The VelocityAdaptiveMeshConstraint object is derived from the *AdaptiveMeshConstraint* object.

Access

```
import step
mdb.models[name].adaptiveMeshConstraints[name]
```

VelocityAdaptiveMeshConstraint(...)

This method creates a VelocityAdaptiveMeshConstraint object.

Path

```
mdb.models[name].VelocityAdaptiveMeshConstraint
```

Required arguments

name
A String specifying the adaptive mesh constraint repository key.

createStepName
A String specifying the name of the step in which the adaptive mesh constraint is created.

region
A *Region* object specifying the region to which the adaptive mesh constraint is applied.

Optional arguments

v1
A Float or a SymbolicConstant specifying the velocity component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *v1*, *v2*, *v3*, *vr1*, *vr2*, and *vr3* are optional arguments, at least one of them must be specified.

v2
A Float or a SymbolicConstant specifying the velocity component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

v3
A Float or a SymbolicConstant specifying the velocity component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr1
A Float or a SymbolicConstant specifying the rotational velocity component about the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr2
A Float or a SymbolicConstant specifying the rotational velocity component about the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr3
A Float or a SymbolicConstant specifying the rotational velocity component about the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the adaptive mesh constraint has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the adaptive mesh constraint's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

motionType
A SymbolicConstant specifying the mesh motion in relation to the underlying material. Possible values are INDEPENDENT, FOLLOW and USER_DEFINED. The default value is INDEPENDENT.

Return value

A VelocityAdaptiveMeshConstraint object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing VelocityAdaptiveMeshConstraint object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VelocityAdaptiveMeshConstraint* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing VelocityAdaptiveMeshConstraint object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the adaptive mesh constraint is modified.

Optional arguments

v1
A Float or a SymbolicConstant specifying the velocity component in the 1-direction. Possible values for the SymbolicConstant are SET and FREED.

v2
A Float or a SymbolicConstant specifying the velocity component in the 2-direction. Possible values for the SymbolicConstant are SET and FREED.

v3
A Float or a SymbolicConstant specifying the velocity component in the 3-direction. Possible values for the SymbolicConstant are SET and FREED.

vr1
A Float or a SymbolicConstant specifying the rotational velocity component about the 1-direction. Possible values for the SymbolicConstant are SET and FREED.

vr2
A Float or a SymbolicConstant specifying the rotational velocity component about the 2-direction. Possible values for the SymbolicConstant are SET and FREED.

vr3
A Float or a SymbolicConstant specifying the rotational velocity component about the 3-direction. Possible values for the SymbolicConstant are SET and FREED.

amplitude
A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the adaptive mesh constraint is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value
`None`.

Exceptions
None.

Members
The VelocityAdaptiveMeshConstraint object can have the following members:

name
A String specifying the adaptive mesh constraint repository key.

category
A SymbolicConstant specifying the category of the adaptive mesh constraint. Possible values are MECHANICAL and THERMAL.

region
A *Region* object specifying the region to which the adaptive mesh constraint is applied.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the adaptive mesh constraint's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

VelocityAdaptiveMeshConstraintState object

The VelocityAdaptiveMeshConstraintState object stores the propagating data for an Arbitrary Lagrangian Eularian (ALE) style velocity adaptive mesh constraint in a step. One instance of this object is created internally by the *VelocityAdaptiveMeshConstraint* object for each step. The instance is also deleted internally by the *VelocityAdaptiveMeshConstraint* object.

The VelocityAdaptiveMeshConstraintState object has no constructor or methods.

The VelocityAdaptiveMeshConstraintState object is derived from the *AdaptiveMeshConstraintState* object.

Access

```
import step
mdb.models[name].steps[name].adaptiveMeshConstraintStates[name]
```

Members

The VelocityAdaptiveMeshConstraintState object has the following members:

v1
A Float specifying the velocity component in the 1-direction.

v2
A Float specifying the velocity component in the 2-direction.

v3
A Float specifying the velocity component in the 3-direction.

vr1
A Float specifying the rotational velocity component about the 1-direction.

vr2
A Float specifying the rotational velocity component about the 2-direction.

vr3
A Float specifying the rotational velocity component about the 3-direction.

v1State
A SymbolicConstant specifying the propagation state of the velocity component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v2State
A SymbolicConstant specifying the propagation state of the velocity component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v3State
A SymbolicConstant specifying the propagation state of the velocity component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr1State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr2State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr3State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *AdaptiveMeshConstraintState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the adaptive mesh constraint has no amplitude reference.

Corresponding analysis keywords

- **ADAPTIVE MESH CONSTRAINT*, TYPE=VELOCITY (degree of freedom: 1, 2, 3, 4, 5, or 6)

Amplitude commands

Amplitude commands are used to create arbitrary time or frequency variations of load, displacement, and some interaction attributes throughout a step using step time or throughout an analysis using total time.

In this section:

- *Amplitude object*
- *ActuatorAmplitude object*
- *BaselineCorrection object*
- *Correlation object*
- *DecayAmplitude object*
- *EquallySpacedAmplitude object*
- *ModulatedAmplitude object*
- *PeriodicAmplitude object*
- *PsdDefinition object*
- *SmoothStepAmplitude object*
- *SolutionDependentAmplitude object*
- *SpectrumAmplitude object*
- *TabularAmplitude object*
- *UserAmplitude object*

Amplitude object

The Amplitude object is the abstract base type for other Amplitude objects. The Amplitude object has no explicit constructor. The methods and members of the Amplitude object are common to all objects derived from the Amplitude.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

Members

The Amplitude object has the following members:

name

A String specifying the repository key.

timeSpan

A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

ActuatorAmplitude object

The ActuatorAmplitude object defines an actuator amplitude curve.

The ActuatorAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

ActuatorAmplitude(...)

This method creates a ActuatorAmplitude object.

Path

```
mdb.models[name].ActuatorAmplitude
session.odbs[name].ActuatorAmplitude
```

Required arguments

name A String specifying the repository key.

Optional arguments

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

An ActuatorAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the ActuatorAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ActuatorAmplitude* method, except for the *name* argument.

Return value
`None.`

Exceptions
RangeError.

Members
The ActuatorAmplitude object has members with the same names and descriptions as the arguments to the *ActuatorAmplitude* method.

Corresponding analysis keywords
- **AMPLITUDE*

BaselineCorrection object

The BaselineCorrection object modifies an acceleration history to minimize the overall drift of the displacement obtained from the time integration of the given acceleration.

Access

```
import amplitude
mdb.models[name].amplitudes[name].baselineCorrection
import odbAmplitude
session.odbs[name].amplitudes[name].baselineCorrection
```

BaselineCorrection(...)

This method creates a BaselineCorrection object.

Path

```
mdb.models[name].amplitudes[name].BaselineCorrection
session.odbs[name].amplitudes[name].BaselineCorrection
```

Required arguments

None.

Optional arguments

intervals A sequence of Floats specifying the correction time interval end points. Possible values are positive and monotonically increasing Floats. The default value is an empty sequence.

Return value

A BaselineCorrection object.

Exceptions

RangeError.

setValues(...)

This method modifies the BaselineCorrection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BaselineCorrection* method.

Return value

`None.`

Exceptions

RangeError.

Members

The BaselineCorrection object has members with the same names and descriptions as the arguments to the *BaselineCorrection* method.

Corresponding analysis keywords

- **BASELINE CORRECTION*

Correlation object

A Correlation is an object used to define the cross-correlation as part of the definition of random loading.

The Correlation object is derived from the *Amplitude* object.

Access

```
import load
mdb.models[name].boundaryConditions[name].correlation[i]
```

Members

The Correlation object has the following members:

name
A String specifying the repository key.

approach
A SymbolicConstant specifying the approach used in the correlation data representation. Possible values are CORRELATED, MOVING_NOISE, UNCORRELATED, and USER. The default value is CORRELATED.

data
A tuple of tuples of Floats specifying the real and imaginary part of the scaling factor. If *approach*=MOVING_NOISE, then *data* represents the noise velocity components 1, 2, and 3.

timeSpan
A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

DecayAmplitude object

The DecayAmplitude object defines an amplitude curve using an exponential decay.

The DecayAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

DecayAmplitude(...)

This method creates a DecayAmplitude object.

Path

```
mdb.models[name].DecayAmplitude
session.odbs[name].DecayAmplitude
```

Required arguments

name
A String specifying the repository key.

initial
A Float specifying the constant A_0.

maximum
A Float specifying the coefficient A.

start
A Float specifying the starting time t_0. Possible values are non-negative numbers.

decayTime
A Float specifying the decay time t_d. Possible values are non-negative numbers.

Optional arguments

timeSpan
A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A DecayAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the DecayAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DecayAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The DecayAmplitude object has members with the same names and descriptions as the arguments to the *DecayAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

EquallySpacedAmplitude object

The EquallySpacedAmplitude object defines a list of amplitude values at fixed time intervals beginning at a specified value of time.

The EquallySpacedAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

EquallySpacedAmplitude(...)

This method creates an EquallySpacedAmplitude object.

Path

```
mdb.models[name].EquallySpacedAmplitude
session.odbs[name].EquallySpacedAmplitude
```

Required arguments

name	A String specifying the repository key.
fixedInterval	A Float specifying the fixed time interval at which the amplitude data are given. Possible values are positive numbers.
data	A sequence of Floats specifying the amplitude values.

Optional arguments

begin	A Float specifying the time at which the first amplitude data are given. Possible values are non-negative numbers. The default value is 0.0.
smooth	The SymbolicConstant SOLVER_DEFAULT or a Float specifying the degree of smoothing. Possible float values are $0 \le smoothing \le 0.5$. If *smooth*=SOLVER_DEFAULT, the default degree of smoothing will be determined by the solver. The default value is SOLVER_DEFAULT.
timeSpan	A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

An EquallySpacedAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the EquallySpacedAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EquallySpacedAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The EquallySpacedAmplitude object has members with the same names and descriptions as the arguments to the *EquallySpacedAmplitude* method.

In addition, the EquallySpacedAmplitude object can have the following member:

baselineCorrection A *BaselineCorrection* object.

Corresponding analysis keywords

- **AMPLITUDE*

ModulatedAmplitude object

The ModulatedAmplitude object defines a modulated amplitude curve.

The ModulatedAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

ModulatedAmplitude(...)

This method creates a ModulatedAmplitude object.

Path

```
mdb.models[name].ModulatedAmplitude
session.odbs[name].ModulatedAmplitude
```

Required arguments

name A String specifying the repository key.

initial A Float specifying the constant A_0.

magnitude A Float specifying the coefficient A.

start A Float specifying the starting time t_0. Possible values are non-negative numbers.

frequency1 A Float specifying the circular frequency 1 (ω_1). Possible values are positive numbers.

frequency2 A Float specifying the circular frequency 2 (ω_2). Possible values are positive numbers.

Optional arguments

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A ModulatedAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the ModulatedAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ModulatedAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ModulatedAmplitude object has members with the same names and descriptions as the arguments to the *ModulatedAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

PeriodicAmplitude object

The PeriodicAmplitude object defines an amplitude curve using a Fourier series.

The PeriodicAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

PeriodicAmplitude(...)

This method creates a PeriodicAmplitude object.

Path

```
mdb.models[name].PeriodicAmplitude
session.odbs[name].PeriodicAmplitude
```

Required arguments

name A String specifying the repository key.

frequency A Float specifying the circular frequency ω. Possible values are positive numbers.

start A Float specifying the starting time t_0. Possible values are positive numbers.

a_0 A Float specifying the constant A_0.

data A sequence of pairs of Floats specifying A_i and B_i pairs.

Optional arguments

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A PeriodicAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the PeriodicAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PeriodicAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PeriodicAmplitude object has members with the same names and descriptions as the arguments to the *PeriodicAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

PsdDefinition object

The PsdDefinition object defines the cross-spectral density frequency function for random response loading.

The PsdDefinition object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

PsdDefinition(...)

This method creates a PsdDefinition object.

Path

```
mdb.models[name].PsdDefinition
session.odbs[name].PsdDefinition
```

Required arguments

name
A String specifying the repository key.

data
A sequence of sequences of Floats specifying the real part of the frequency function, the imaginary part of the frequency function, and the frequency or frequency band number values, depending on the value of *unitType*.

Optional arguments

unitType
A SymbolicConstant specifying the type of units for specifying the frequency function. FORCE implies power units. BASE implies gravity used to define base motion. DB implies decibel units. Possible values are FORCE, BASE, and DB. The default value is FORCE.

referenceGravityAcceleration
A Float specifying the reference gravity acceleration. This argument applies when *unitType* = BASE. The default value is 1.0.

referenecePower
A Float specifying the reference power value, in load units squared. This argument applies when *unitType* = DB. The default value is 0.0.

user
A Boolean specifying whether the frequency function is defined in user subroutine UPSD. If specified, then *data* is not applicable, and the *unitType* value must not be DB. The default value is OFF.

timeSpan
A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

amplitude A String specifying the name of the amplitude that describes the dynamic event used to define the cross-spectral density frequency function. The default value is an empty string.

Return value

A PsdDefinition object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the PsdDefinition object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PsdDefinition* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PsdDefinition object has members with the same names and descriptions as the arguments to the *PsdDefinition* method.

Corresponding analysis keywords

- **PSD-DEFINITION*

SmoothStepAmplitude object

The SmoothStepAmplitude object defines an amplitude that ramps up or down smoothly from one data point to another.

The SmoothStepAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

SmoothStepAmplitude(...)

This method creates a SmoothStepAmplitude object.

Path

```
mdb.models[name].SmoothStepAmplitude
session.odbs[name].SmoothStepAmplitude
```

Required arguments

name A String specifying the repository key.

data A sequence of pairs of Floats specifying time/frequency and amplitude pairs. Possible values for time/frequency are positive numbers.

Optional arguments

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A SmoothStepAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the SmoothStepAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SmoothStepAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SmoothStepAmplitude object has members with the same names and descriptions as the arguments to the *SmoothStepAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

SolutionDependentAmplitude object

The SolutionDependentAmplitude object defines a solution-dependent amplitude for superplastic forming analysis.

The SolutionDependentAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

SolutionDependentAmplitude(...)

This method creates a SolutionDependentAmplitude object.

Path

```
mdb.models[name].SolutionDependentAmplitude
session.odbs[name].SolutionDependentAmplitude
```

Required arguments

name A String specifying the repository key.

Optional arguments

initial A Float specifying the initial amplitude value. Possible values are those between *minimum* and *maximum*. The default value is 1.0.

minimum A Float specifying the minimum amplitude value. Possible values are those smaller than *maximum* and *initial*. The default value is 0.1.

maximum A Float specifying the maximum amplitude value. Possible values are those larger than *minimum* and *initial*. The default value is 1000.0.

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A SolutionDependentAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the SolutionDependentAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SolutionDependentAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SolutionDependentAmplitude object has members with the same names and descriptions as the arguments to the *SolutionDependentAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

SpectrumAmplitude object

The SpectrumAmplitude object defines the spectrum of responses for displacement, velocity, or acceleration to be used in a response spectrum analysis.

The SpectrumAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

SpectrumAmplitude(...)

This method creates a SpectrumAmplitude object.

Path

```
mdb.models[name].SpectrumAmplitude
session.odbs[name].SpectrumAmplitude
```

Required arguments

name A String specifying the repository key.

method A SymbolicConstant specifying the method for specifying the spectrum. Possible values are DEFINE and CALCULATE.

data A sequence of sequences of Floats specifying the magnitude, frequency, and damping values.

Optional arguments

specificationUnits A SymbolicConstant specifying the units used for specifying the spectrum. Possible values are DISPLACEMENT, VELOCITY, ACCELERATION, and GRAVITY. The default value is ACCELERATION.

eventUnits A SymbolicConstant specifying the units used to describe the dynamic event in the amplitude used for the calculation. Possible values are EVENT_DISPLACEMENT, EVENT_VELOCITY, EVENT_ACCELERATION, and EVENT_GRAVITY. The default value is EVENT_ACCELERATION.

solution A SymbolicConstant specifying the solution method for the dynamic equations. Possible values are ABSOLUTE_VALUE and RELATIVE_VALUE. The default value is ABSOLUTE_VALUE.

timeIncrement A Float specifying the implicit time increment used to calculate the spectrum. This argument is required when the *method* = CALCULATE. The default value is 0.0.

gravity
A Float specifying the acceleration due to gravity. This argument applies only when *specificationUnits* = GRAVITY or*eventUnits* = GRAVITY. The default value is 1.0.

criticalDamping
A Boolean specifying whether to calculate the spectrum for only the specified range of critical damping values or a list of values. If *criticalDamping* = ON, the spectrum is calculated only for the specified range of critical damping values. If *criticalDamping* = OFF, the spectrum is calculated for a list of damping values. The default value is OFF.

timeSpan
A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

amplitude
A String specifying the name of the amplitude that describes the dynamic event used to calculate the spectrum. The default value is an empty string.

Return value

A SpectrumAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the SpectrumAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SpectrumAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SpectrumAmplitude object has members with the same names and descriptions as the arguments to the *SpectrumAmplitude* method.

Corresponding analysis keywords

- **SPECTRUM*

TabularAmplitude object

The TabularAmplitude object defines an amplitude curve as a table of values at convenient points on the time scale.

The TabularAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

TabularAmplitude(...)

This method creates a TabularAmplitude object.

Path

```
mdb.models[name].TabularAmplitude
session.odbs[name].TabularAmplitude
```

Required arguments

name A String specifying the repository key.

data A sequence of pairs of Floats specifying time/frequency and amplitude pairs. Possible values for time/frequency are positive numbers.

Optional arguments

smooth The SymbolicConstant SOLVER_DEFAULT or a Float specifying the degree of smoothing. Possible float values are between 0 and 0.5. If *smooth*=SOLVER_DEFAULT, the default degree of smoothing will be determined by the solver. The default value is SOLVER_DEFAULT.

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A TabularAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the TabularAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TabularAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TabularAmplitude object has members with the same names and descriptions as the arguments to the *TabularAmplitude* method.

In addition, the TabularAmplitude object can have the following member:

baselineCorrection A *BaselineCorrection* object.

Corresponding analysis keywords

- **AMPLITUDE*

UserAmplitude object

The UserAmplitude object defines an amplitude curve using the *UAMP* or *VUAMP* user subroutine.

The UserAmplitude object is derived from the *Amplitude* object.

Access

```
import amplitude
mdb.models[name].amplitudes[name]
import odbAmplitude
session.odbs[name].amplitudes[name]
```

UserAmplitude(...)

This method creates a UserAmplitude object.

Path

```
mdb.models[name].UserAmplitude
session.odbs[name].UserAmplitude
```

Required arguments

name A String specifying the repository key.

numVariables An Int specifying the number of variables for the *UAMP* or *VUAMP* user subroutine.

Optional arguments

timeSpan A SymbolicConstant specifying the time span of the amplitude. Possible values are STEP and TOTAL. The default value is STEP.

Return value

A UserAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the UserAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *UserAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The UserAmplitude object has members with the same names and descriptions as the arguments to the *UserAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

Animation commands

Animation commands are used to create and control object-based and image-based animations in the viewport.

In this section:

- *AnimationController object*
- *AnimationOptions object*
- *AVIOptions object*
- *GIFOptions object*
- *ImageAnimationOptions object*
- *ImageAnimation object*
- *QuickTimeOptions object*
- *MP4Options object*
- *Movie object*
- *Session object*

AnimationController object

The AnimationController object controls all object-based animation to be displayed in the viewports. The AnimationController object has no constructor. Abaqus creates the *animationController* member when it creates the *Session* object.

Access

```
import animation
session.viewports[name].animationController
```

play(...)

This method begins the animation.

Required arguments

None.

Optional arguments

duration The SymbolicConstant UNLIMITED or an Int specifying how many seconds to play the animation. The default value is UNLIMITED.

Return value

None.

Exceptions

If *animationType*=NONE: AnimationError: animationType not set

stop()

This method stops the animation.

Arguments

None.

Return value

None.

Exceptions

None.

incrementFrame()

This method increments the animation frame.

Arguments

None.

Return value

`None.`

Exceptions

None.

decrementFrame()

This method decrements the animation frame.

Arguments

None.

Return value

`None.`

Exceptions

None.

showFrame(...)

This method renders the specified frame of the animation.

Required arguments

None.

Optional arguments

frame An Int specifying the frame number.

value A Float specifying the frame: for *animationType*=TIME_HISTORY the frame with the time nearest to this value, for *animationType*=HARMONIC the frame with the angle nearest to this value, for *animationType*=SCALE_FACTOR the frame with the scale value nearest to this value.

Return value

`None.`

Exceptions

None.

showFirstFrame()

This method renders the first frame of the animation.

Arguments

None.

Return value

`None.`

Exceptions

None.

showLastFrame()

This method renders the last frame of the animation.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the AnimationController object.

Required arguments

None.

Optional arguments

animationType A SymbolicConstant specifying the type of movie to play. Possible values are SCALE_FACTOR, HARMONIC, TIME_HISTORY, and NONE. The default value is NONE.

Return value

`None.`

Exceptions

RangeError.

Members

The AnimationController object can have the following members:

animationType A SymbolicConstant specifying the type of movie to play. Possible values are SCALE_FACTOR, HARMONIC, TIME_HISTORY, and NONE. The default value is NONE.

state A SymbolicConstant specifying the state of the animation controller. Possible values are STOP and PLAY. The default value is STOP.

AnimationOptions object

The AnimationOptions object is used to store values and attributes associated with an *AnimationController* object.

The AnimationOptions object has no constructor command. Abaqus creates the *animationOptions* member when it creates the *AnimationController* object.

Access

```
import animation
session.animationOptions
```

`setValues(...)`

This method modifies the AnimationOptions object.

Required arguments

None.

Optional arguments

mode

A SymbolicConstant specifying the animation mode. Possible values are PLAY_ONCE, LOOP, LOOP_BACKWARD, and SWING. The default value is LOOP.

frameRate

An Int specifying the animation rate in frames/second. Possible values are 1 ≤*frameRate*≤ 100. The default value is 50.

frameCounter

A Boolean specifying whether to show the frame counter. The default value is ON.

relativeScaling

A SymbolicConstant specifying the relative scaling when the *AnimationController* object's *animationType*=SCALE_FACTOR or HARMONIC. Possible values are FULL_CYCLE and HALF_CYCLE. The default value is HALF_CYCLE.

numScaleFactorFrames

An Int specifying the number of frames to be used when the *AnimationController* object's *animationType*=SCALE_FACTOR or HARMONIC. The default value is 7.

timeHistoryMode

A SymbolicConstant specifying whether the time history animation is time based or frame based. Possible values are FRAME_BASED and TIME_BASED. The default value is FRAME_BASED.

maxTime

A Float specifying the maximum time for time based time history animation when *maxTimeAutoCompute* = False.

maxTimeAutoCompute

A Boolean specifying whether the animation maximum time value should be computed from the active frames when *timeHistoryMode* is set to TIME_BASED. The default value is ON.

minTime

A Float specifying the minimum time for time based time history animation when *minTimeAutoCompute* = False.

minTimeAutoCompute

A Boolean specifying whether the animation minimum time value should be computed from the active frames when *timeHistoryMode* is set to TIME_BASED. The default value is ON.

timeIncrement

A Float specifying the time increment for frame selection when *timeHistoryMode* is set to TIME_BASED.

xyUseHighlightMethod

A Boolean specifying whether to use the highlight method to draw the time tracker line and symbols. The default value is ON.

xyShowLine

A Boolean specifying whether to show the time tracker line. The default value is ON.

xyLineStyle

A SymbolicConstant specifying the *X–Y* time tracker line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

xyLineThickness

A SymbolicConstant specifying the *X–Y* time tracker line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is MEDIUM.

xyLineColor

A String specifying the color used to plot the *X–Y* time tracker line when *xyUseHighlightMethod* = False. The default value is "Yellow".

xyShowSymbol

A Boolean specifying whether to show the time tracker symbols. The default value is ON.

xySymbolMarker

A SymbolicConstant specifying the marker type to be used for all animation capable *X–Y* curve or the SymbolicConstant DEFAULT specifying that the system will take the marker associated to each curve Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER
- DEFAULT

The default value is DEFAULT.

xySymbolSize

A SymbolicConstant specifying the size of the markers. Possible values are SMALL, MEDIUM, and LARGE. The default value is MEDIUM.

xySymbolColor A String specifying the color used to plot *X–Y* time tracker symbols when *xyUseHighlightMethod* = False. When setting the color to 'Default' the system will take the color associated to each curve. The default value is "Default".

Return value

`None.`

Exceptions

None.

Members

The AnimationOptions object can have the following members:

mode A SymbolicConstant specifying the animation mode. Possible values are PLAY_ONCE, LOOP, LOOP_BACKWARD, and SWING. The default value is LOOP.

frameRate An Int specifying the animation rate in frames/second. Possible values are 1 ≤*frameRate*≤ 100. The default value is 50.

frameCounter A Boolean specifying whether to show the frame counter. The default value is ON.

relativeScaling A SymbolicConstant specifying the relative scaling when the *AnimationController* object's *animationType*=SCALE_FACTOR or HARMONIC. Possible values are FULL_CYCLE and HALF_CYCLE. The default value is HALF_CYCLE.

numScaleFactorFrames An Int specifying the number of frames to be used when the *AnimationController* object's *animationType*=SCALE_FACTOR or HARMONIC. The default value is 7.

timeHistoryMode A SymbolicConstant specifying whether the time history animation is time based or frame based. Possible values are FRAME_BASED and TIME_BASED. The default value is FRAME_BASED.

maxTime A Float specifying the maximum time for time based time history animation when *maxTimeAutoCompute* = False.

maxTimeAutoCompute A Boolean specifying whether the animation maximum time value should be computed from the active frames when *timeHistoryMode* is set to TIME_BASED. The default value is ON.

maxTimeAutoValue A Float specifying the maximum time when *timeHistoryMode* is set to TIME_BASED and the *maxTimeAutoCompute* value is True. This value is computed as the maximum time of all active frames displayed in viewports where the animation is active.

minTime A Float specifying the minimum time for time based time history animation when *minTimeAutoCompute* = False.

minTimeAutoCompute

A Boolean specifying whether the animation minimum time value should be computed from the active frames when *timeHistoryMode* is set to TIME_BASED. The default value is ON.

minTimeAutoValue

A Float specifying the minimum time when *timeHistoryMode* is set to TIME_BASED and the *minTimeAutoCompute* value is True. This value is computed as the minimum time of all active frames displayed in viewports where the animation is active.

timeIncrement

A Float specifying the time increment for frame selection when *timeHistoryMode* is set to TIME_BASED.

xyUseHighlightMethod

A Boolean specifying whether to use the highlight method to draw the time tracker line and symbols. The default value is ON.

xyShowLine

A Boolean specifying whether to show the time tracker line. The default value is ON.

xyLineStyle

A SymbolicConstant specifying the *X–Y* time tracker line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

xyLineThickness

A SymbolicConstant specifying the *X–Y* time tracker line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is MEDIUM.

xyShowSymbol

A Boolean specifying whether to show the time tracker symbols. The default value is ON.

xySymbolMarker

A SymbolicConstant specifying the marker type to be used for all animation capable *X–Y* curve or the SymbolicConstant DEFAULT specifying that the system will take the marker associated to each curve Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER
- DEFAULT

The default value is DEFAULT.

xySymbolSize

A SymbolicConstant specifying the size of the markers. Possible values are SMALL, MEDIUM, and LARGE. The default value is MEDIUM.

xyLineColor

A String specifying the color used to plot the *X–Y* time tracker line when *xyUseHighlightMethod* = False. The default value is "Yellow".

xySymbolColor

A String specifying the color used to plot *X–Y* time tracker symbols when *xyUseHighlightMethod* = False. When setting the color to 'Default' the system will take the color associated to each curve. The default value is "Default".

AVIOptions object

The AVIOptions object is used to store values and attributes to be used in generating AVI animation. The AVIOptions object has no constructor. Abaqus creates the *aviOptions* member when the animation module is imported.

Access

```
import animation
session.aviOptions
```

`setValues(...)`

This method modifies the AVIOptions object.

Required arguments

None.

Optional arguments

compressionMethod A SymbolicConstant specifying the compression method for the AVI format. Possible values are:

- RAW8, specifying an uncompressed format of 8 bits per pixel.
- RAW32, specifying an uncompressed format of 32 bits per pixel.
- RLE8, specifying a run length encoded format of 8 bits per pixel.
- CODEC, specifying a format defined by CODEC (encoder/decoder). The availability of the CODEC is system dependent

The default value is RLE8.

compressionQuality An Int specifying the quality of the compression as a percentage when the *compressionMethod* is set to CODEC.

codecOptions A String specifying the system specific options defining the CODEC when the *compressionMethod* is set to CODEC.

sizeDefinition A SymbolicConstant specifying how the width and height of the image are specified. Possible values are SIZE_ON_SCREEN and USER_DEFINED. The default value is SIZE_ON_SCREEN.

imageSize A pair of Ints specifying the width and height of the image in pixels when *sizeDefinition*=USER_DEFINED. Possible values are Ints in the range (*minWidth*, *minHeight*) $\leq$ (width, height) $\leq$ (*maxWidth*, *maxHeight*). The default value is the screen size.



Note:

The values of the minimum width (*minWidth*) and height (*minHeight*) depend on the following:

- viewport font size,
- whether decorations are printed,

- decoration size, and
- screen resolution.

The minimum width and height are normally < 50 pixels. The values of the maximum width (*maxWidth*) and height (*maxHeight*) depend on the graphics capabilities of the system. The maximum width and height will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

None.

Members

The AVIOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

GIFOptions object

The GIFOptions object is used to store values and attributes to be used in generating animated GIF animations. The GIFOptions object has no constructor. Abaqus creates the *gifOptions* member when the animation module is imported.

Access

```
import animation
session.gifOptions
```

`setValues(...)`

This method modifies the GIFOptions object.

Required arguments

None.

Optional arguments

sizeDefinition A SymbolicConstant specifying how the width and height of the image are specified. Possible values are SIZE_ON_SCREEN and USER_DEFINED. The default value is SIZE_ON_SCREEN.

imageSize A pair of Ints specifying the width and height of the image in pixels when *sizeDefinition*=USER_DEFINED. Possible values are Ints in the range (*minWidth*, *minHeight*) (width, height) (*maxWidth*, *maxHeight*). The default value is the screen size.



Note: The values of the minimum width (*minWidth*) and height (*minHeight*) depend on the following:

- viewport font size,
- whether decorations are printed,
- decoration size, and
- screen resolution.

The minimum width and height are normally < 50 pixels. The values of the maximum width (*maxWidth*) and height (*maxHeight*) depend on the graphics capabilities of the system. The maximum width and height will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

None.

Members

The GIFOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

ImageAnimationOptions object

The ImageAnimationOptions object is used to store values and attributes associated with saving viewport animations. The ImageAnimationOptions object has no constructor. Abaqus creates the *imageAnimationOptions* member when the animation module is imported.

Access

```
import animation
session.imageAnimationOptions
```

setValues(...)

This method modifies the ImageAnimationOptions object.

Required arguments

None.

Optional arguments

frameRate An Int specifying the frame rate to record on the saved animation file. The effective frame rate in frames per second will be obtained by dividing the given frame rate by the time scale.

timeScale An Int specifying the time scale to apply to the frame rate.

vpDecorations A Boolean specifying whether to capture the viewport border and title. The default value is ON.

vpBackground A Boolean specifying whether to capture viewport backgrounds. The default value is OFF.

compass A Boolean specifying whether to capture the view compass. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The ImageAnimationOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

ImageAnimation object

The ImageAnimation object is used to build frame by frame animation.

Access

```
import animation
session.imageAnimation
```

ImageAnimation(...)

This method creates an ImageAnimation object from the specified filename and format.

Path

```
session.ImageAnimation
```

Required arguments

fileName	A String specifying the name of the animation file to generate.
format	A SymbolicConstant specifying the format of the generated file. Possible values are MP4, AVI, QUICKTIME, and GIF.

Optional arguments

None.

Return value

An ImageAnimation object.

Exceptions

None.

writeFrame(...)

This method adds a frame to the ImageAnimation object.

Required arguments

None.

Optional arguments

canvasObjects	A sequence specifying the canvas objects to capture. The default is to capture all canvas objects.

Return value

```
None.
```

Exceptions

None.

close()

This method closes the ImageAnimation object.

Arguments

None.

Return value

`None.`

Exceptions

None.

closed()

This method indicates if the ImageAnimation is open or closed for writing animation frames.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The ImageAnimation object has the following member:

fileName A String specifying the file to which the animation frames is to be written.

QuickTimeOptions object

The QuickTimeOptions object is used to store values and attributes to be used in generating QuickTime animation. The QuickTimeOptions object has no constructor. Abaqus creates the *quickTimeOptions* member when the animation module is imported.

Access

```
import animation
session.quickTimeOptions
```

`setValues(...)`

This method modifies the QuickTimeOptions object.

Required arguments

None.

Optional arguments

compressionMethod A SymbolicConstant specifying the compression method for the QuickTime format. Possible values are:

- RAW24, specifying an uncompressed format of 24 bits per pixel.
- RLE24, specifying a run length encoded format of 24 bits per pixel.

The default value is RLE24.

sizeDefinition A SymbolicConstant specifying how the width and height of the image are specified. Possible values are SIZE_ON_SCREEN and USER_DEFINED. The default value is SIZE_ON_SCREEN.

imageSize A pair of Ints specifying the width and height of the image in pixels when *sizeDefinition*=USER_DEFINED. Possible values are Ints in the range (*minWidth*, *minHeight*) $\leq$ (width, height) $\leq$ (*maxWidth*, *maxHeight*). The default value is the screen size.



Note:

The values of the minimum width (*minWidth*) and height (*minHeight*) depend on the following:

- viewport font size,
- whether decorations are printed,
- decoration size, and
- screen resolution.

The minimum width and height are normally < 50 pixels. The values of the maximum width (*maxWidth*) and height (*maxHeight*) depend on the graphics capabilities of the system. The maximum width and height will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

None.

Members

The QuickTimeOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

MP4Options object

The MP4Options object is used to store values and attributes to be used in generating MP4 animations.
The MP4Options object has no constructor. Abaqus creates the *mp4Options* member when the animation module is imported.

Access

```
import animation
session.mp4Options
```

`setValues(...)`

This method modifies the MP4Options object.

Required arguments

None.

Optional arguments

compressionQuality An Int specifying the quality of the compression as a percentage. The default is 75.

codecOptions A String specifying the codec implementation. Possible values are "mpeg4" and, if an appropriate FFmpeg installation can be found that supports it, "libx264." The default is "libx264" when available.

sizeDefinition A SymbolicConstant specifying how the width and height of the image are specified. Possible values are SIZE_ON_SCREEN and USER_DEFINED. The default value is SIZE_ON_SCREEN.

imageSize A pair of Ints specifying the width and height of the image in pixels when *sizeDefinition*=USER_DEFINED. Possible values are Ints in the range (*minWidth*, *minHeight*) (width, height) (*maxWidth*, *maxHeight*). The default value is the screen size.



Note: The values of the minimum width (*minWidth*) and height (*minHeight*) depend on the following:

- viewport font size,
- whether decorations are printed,
- decoration size, and
- screen resolution.

The minimum width and height are normally < 50 pixels. The values of the maximum width (*maxWidth*) and height (*maxHeight*) depend on the graphics capabilities of the system. The maximum width and height will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

None.

Members

The MP4Options object has members with the same names and descriptions as the arguments to the *setValues* method.

Movie object

The Movie object is used to store values and attributes associated with a movie file. Upon creation, the Movie object is added to the session.movies repository.

Access

```
import animation
session.movies[name]
```

Movie(...)

This method creates a Movie object from the contents of the specified file.

Path

```
session.Movie
```

Required arguments

name
A String specifying the repository name for the movie.

fileName
A String specifying the file from which the movie is to be read. The file extension must be specified and indicates the movie format (for example, `.mp4`, `.avi`, etc.).

Optional arguments

startFrame
An Int specifying the first frame to be displayed from this movie. The default value is 0.

endFrame
An Int specifying the last frame to be displayed from this movie. A negative number will indicate reverse numbering: -1 is the last frame of the movie. The default value is −1.

timelineStartFrame
An Int specifying the global timeline frame number that corresponds to *startFrame*. A value of 0 will indicate the first frame to be displayed in the viewport. The default value is 0.

timelineEndFrame
An Int specifying the global timeline frame number that corresponds to *endFrame*. A negative number will indicate reverse numbering: -1 indicates the last frame to be displayed in the viewport. The default value is −1.

timelineStartTime
A Float specifying the global timeline time that corresponds to the time of *startFrame*. The default value is 0.0.

timelineEndTime
The SymbolicConstant END_FRAME_TIME or a Float specifying the global timeline time that corresponds to the time of *endFrame*. The SymbolicConstant END_FRAME_TIME indicates the time in this movie corresponding to *endFrame*. The default value is END_FRAME_TIME.

Return value

A Movie object.

Exceptions

ValueError.

If *fileName* does not exist or can not be read:	ValueError: Unable to open movie file
If *fileName* references an unsupported movie file format:	ValueError: Unsupported movie format
If the contents of *fileName* are corrupt or can not be decoded:	ValueError: Unable to decode movie file

`setValues(...)`

This method modifies the Movie object.

Required arguments

None.

Optional arguments

startFrame
An Int specifying the first frame to be displayed from this movie. The default value is 0.

endFrame
An Int specifying the last frame to be displayed from this movie. A negative number will indicate reverse numbering: -1 is the last frame of the movie. The default value is −1.

timelineStartFrame
An Int specifying the global timeline frame number that corresponds to *startFrame*. A value of 0 will indicate the first frame to be displayed in the viewport. The default value is 0.

timelineEndFrame
An Int specifying the global timeline frame number that corresponds to *endFrame*. A negative number will indicate reverse numbering: -1 indicates the last frame to be displayed in the viewport. The default value is −1.

timelineStartTime
A Float specifying the global timeline time that corresponds to the time of *startFrame*. The default value is 0.0.

timelineEndTime
The SymbolicConstant END_FRAME_TIME or a Float specifying the global timeline time that corresponds to the time of *endFrame*. The SymbolicConstant END_FRAME_TIME indicates the time in this movie corresponding to *endFrame*. The default value is END_FRAME_TIME.

Return value

`None.`

Exceptions

RangeError.

If *startFrame*, *endFrame*, *timelineStartFrame*, *timelineEndFrame*, *timelineStartTime*, or *timelineEndTime* are outside their respective valid range.	RangeError: startFrame must be an Integer in the range: 0 to numFrames-1

Members

The Movie object has members with the same names and descriptions as the arguments to the *Movie* method.

In addition, the Movie object has the following members:

width	An Int specifying the width of the movie in pixels.
height	An Int specifying the height of the movie in pixels.
numFrames	An Int specifying the total number of frames on the movie file.
duration	A Float specifying the duration of the movie in seconds.
memory	An Int specifying the memory taken by the movie frames as selected.

Session object

The following commands operate on Session objects. For more information about the Session object, see *Session object*.

Access

```
import animation
```

`writeImageAnimation(...)`

This method writes the animations present in the list of canvas objects to a file. It generates an animation file using the given file name and file format and uses the values in the appropriate options object.

Required arguments

fileName
A String specifying the name of the animation file to generate.

format
A SymbolicConstant specifying the format of the generated file. Possible values are MP4, AVI, QUICKTIME, GIF, VRML, and COMPRESSED_VRML.

Optional arguments

canvasObjects
A sequence specifying the canvas objects to capture. The default behavior is to capture all canvas objects.

Return value

`None.`

Exceptions

None.

Annotation commands

Annotation commands are used to create, position, and modify viewport annotations.

In this section:

- *Annotation object*
- *AnnotationsToPlotArray object*
- *Arrow object*
- *Text object*
- *Viewport object*

Annotation object

The Annotation object is the abstract base type for other Annotation objects. The Annotation object has no explicit constructor. The methods and members of the Annotation object are common to all objects derived from Annotation.

Access

```
import annotationToolset
mdb.annotations[name]
session.odbs[name].userData.annotations[name]
session.viewports[name].annotationsToPlot[i]
```

bringToFront()

This method brings the Annotation object to the top of the annotation stack.

Arguments

None.

Return value

None.

Exceptions

None.

sendToBack()

This method sends the Annotation object to the bottom of the annotation stack.

Arguments

None.

Return value

None.

Exceptions

None.

bringForward()

This method brings the Annotation object one position up in the annotation stack.

Arguments

None.

Return value

None.

Exceptions

None.

sendBackward()

This method sends the Annotation object one position down in the annotation stack.

Arguments

None.

Return value

None.

Exceptions

None.

moveBefore(...)

This method moves the Annotation object before another object in the same repository.

Required arguments

name A String specifying the name of the other Annotation object.

Optional arguments

None.

Return value

None.

Exceptions

None.

moveAfter(...)

This method moves the Annotation object after another object in the same repository.

Required arguments

name A String specifying the name of the other Annotation object.

Optional arguments

None.

Return value

None.

Exceptions

None.

translate(...)

This method translates the Annotation object on the viewport plane.

Required arguments

None.

Optional arguments

x A Float specifying the *X* translation amount in millimeters.

y A Float specifying the *Y* translation amount in millimeters.

Return value

`None.`

Exceptions

None.

Members

The Annotation object has the following member:

name A String specifying the annotation repository key.

AnnotationsToPlotArray object

The AnnotationsToPlotArray object is a sequence that stores references to plotted annotations. By adding annotations to and removing annotations from this sequence, you can control which annotations are displayed in a particular viewport.

Access

```
import annotationToolset
session.viewports[name].annotationsToPlot
```

bringForward(...)

This method brings the *Annotation* object one position forward in the AnnotationsToPlotArray sequence.

Required arguments

index An Int specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

Optional arguments

None.

Return value

None.

Exceptions

None.

bringToFront(...)

This method brings the *Annotation* object to the beginning of the AnnotationsToPlotArray sequence.

Required arguments

index An Int specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

Optional arguments

None.

Return value

None.

Exceptions

None.

moveAfter(...)

This method moves the *Annotation* object after another object in the same AnnotationsToPlotArray sequence.

Required arguments

index An Integer specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

other An Integer specifying the index of the other *Annotation* object in the AnnotationsToPlotArray sequence after which this object will be moved.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

moveBefore(...)

This method moves the *Annotation* object before another object in the same AnnotationsToPlotArray sequence.

Required arguments

index An Int specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

other An Int specifying the index of the other *Annotation* object in the AnnotationsToPlotArray sequence before which this object will be moved.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

sendBackward(...)

This method sends the *Annotation* object one position backward in the AnnotationsToPlotArray sequence.

Required arguments

index An Int specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`sendToBack(...)`

This method sends the *Annotation* object to the end of the AnnotationsToPlotArray sequence.

Required arguments

index An Int specifying the index of the *Annotation* object in the AnnotationsToPlotArray sequence.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The AnnotationsToPlotArray object has no members.

Arrow object

The Arrow object stores the visual settings and location of an arrow annotation.

The Arrow object is derived from the *Annotation* object.

Access

```
import annotationToolset
mdb.annotations[name]
session.odbs[name].userData.annotations[name]
session.viewports[name].annotationsToPlot[i]
```

Arrow(...)

This method creates an Arrow object.

Path

mdb.Arrow

session.odbs[*name*].userData.Arrow

Required arguments

name
A String specifying the annotation repository key.

Optional arguments

startPoint
A pair of Floats specifying the start point *X*- and *Y*-offsets in millimeters from *startAnchor*. The default value is (0, 0).

endPoint
A pair of Floats specifying the end point *X*- and *Y*-offsets in millimeters from *endAnchor*. The default value is (0, 0).

startAnchor
A SymbolicConstant or a sequence of Floats specifying a point. A sequence of two Floats specifies the *X*- and *Y*-coordinates as percentages of the viewport width and height. A sequence of three Floats specifies the *X*-, *Y*-, and *Z*-coordinates of a point in the model coordinate system. A SymbolicConstant indicates a relative position. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

endAnchor

A SymbolicConstant or a sequence of Floats specifying a point. A sequence of two Floats specifies the *X*- and *Y*-coordinates as percentages of the viewport width and height. A Sequence of three Floats specifies the *X*-, *Y*-, and *Z*-coordinates of a point in the model coordinate system. A SymbolicConstant indicates a relative position. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

startHeadStyle

A SymbolicConstant specifying the style of the start head. Possible values are:

- ARROW
- FILLED_ARROW
- HOLLOW_CIRCLE
- FILLED_CIRCLE
- HOLLOW_DIAMOND
- FILLED_DIAMOND
- HOLLOW_SQUARE
- FILLED_SQUARE
- NONE

The default value is NONE.

endHeadStyle

A SymbolicConstant specifying the style of the end head. Possible values are:

- ARROW
- FILLED_ARROW
- HOLLOW_ CIRCLE
- FILLED_CIRCLE
- HOLLOW_DIAMOND
- FILLED_DIAMOND
- HOLLOW_SQUARE
- FILLED_SQUARE
- NONE

The default value is FILLED_ARROW.

startGap A Float specifying the distance in millimeters between the arrow start point and the arrow start head. The default value is 0.0.

endGap A Float specifying the distance in millimeters between the arrow end point and the arrow end head. The default value is 0.0.

color A String specifying the color of the arrow. Possible string values are any valid color. The default value is "White".

lineStyle A SymbolicConstant specifying the line style of the arrow. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

lineThickness A SymbolicConstant specifying the line thickness of the arrow. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

Return value

An Arrow object.

Exceptions

InvalidNameError.

translateStartPoint(...)

This method translates the start point of the Arrow object on the viewport plane.

Required arguments

None.

Optional arguments

x A Float specifying the *X* translation amount in millimeters.

y A Float specifying the*Y* translation amount in millimeters.

Return value

`None.`

Exceptions

None.

translateEndPoint(...)

This method translates the end point of the Arrow object on the viewport plane.

Required arguments

None.

Optional arguments

x A Float specifying the *X* translation amount in millimeters.

y A Float specifying the*Y* translation amount in millimeters.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the Arrow object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Arrow* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Arrow object has members with the same names and descriptions as the arguments to the *Arrow* method.

Text object

The Text object stores the text settings and location of a text annotation.

The Text object is derived from the *Annotation* object.

Access

```
import annotationToolset
mdb.annotations[name]
session.odbs[name].userData.annotations[name]
session.viewports[name].annotationsToPlot[i]
```

Text(...)

This method creates a Text object.

Path

mdb.Text

session.odbs[*name*].userData.Text

Required arguments

name A String specifying the annotation repository key.

Optional arguments

text A String specifying the text of the Text object. The default value is an empty string.

offset A pair of Floats specifying the *X*- and *Y*-offsets in millimeters of the Text object from *anchor*. The default value is (0, 0).

anchor A SymbolicConstant or a sequence of Floats specifying a point. A sequence of two Floats specifies the *X*- and *Y* coordinates as percentages of the viewport width and height. A Sequence of three Floats specifies the *X*-, *Y*-, and *Z*-coordinates of a point in the model coordinate system. A SymbolicConstant specifies a relative position. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

referencePoint

A SymbolicConstant or a sequence of Floats specifying a point. The sequence of two Floats specifies the *X*- and *Y*-coordinates of the reference point of the Text annotation given as percentages of its width and height. The SymbolicConstant indicates a relative position. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

rotationAngle

A Float specifying the amount of rotation in degrees about *referencePoint*. The default value is 0.0.

color

A String specifying the color of the Text object. Possible string values are any valid color. The default value is "White".

font

A String specifying the font of the Text object. Possible string values are any valid font specification. The default value is "-*-verdana-medium-r-normal--120-*".

backgroundStyle

A SymbolicConstant specifying the Text object background style. Possible values are MATCH, TRANSPARENT, and OTHER. The default value is TRANSPARENT.

backgroundColor

A String specifying the color of the Text object background. Possible string values are any valid color. The default value matches the viewport background.

box

A Boolean specifying whether the box around the text is shown. The default value is OFF.

justification

A SymbolicConstant specifying the Text object justification for multiline text. Possible values are JUSTIFY_LEFT, JUSTIFY_CENTER, and JUSTIFY_RIGHT. The default value is JUSTIFY_LEFT.

Return value

A Text object.

Exceptions

None.

`setValues(...)`

This method modifies the Text object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Text* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Text object has members with the same names and descriptions as the arguments to the *Text* method.

In addition, the Text object has the following members:

width A Float specifying the width in millimeters of the Text object.

height A Float specifying the height in millimeters of the Text object.

Viewport object

The following commands operate on Viewport objects. For more information about the Viewport object, see *Viewport object*.

Access

```
import annotationToolset
```

`plotAnnotation(...)`

This method plots an *Annotation* object in aViewport.

Required arguments

annotation

An *Annotation* object to plot.

Optional arguments

index

An Int specifying the index of the *Annotation* object in the sequence of annotations to plot. The default value is zero.

Return value

`None.`

Exceptions

None.

Assembly commands

Features in Abaqus/CAE include Parts, Datums, Partitions, and Assembly operations.
Assembly commands create Feature objects on only the *rootAssembly* object. The commands that create Feature objects on only the Part object are described in *Part commands*. The commands that create Feature objects on both the Part and the *rootAssembly* objects are described in *Feature commands*.

In this section:

- *Assembly object*
- *ConnectorOrientation object*
- *Feature object*
- *PartInstance object*
- *ModelInstance object*
- *Model object*

Assembly object

An Assembly object is a container for instances of parts.
The Assembly object has no constructor command. Abaqus creates the *rootAssembly* member when a *Model* object is created.

Access

```
import assembly
mdb.models[name].rootAssembly
```

backup()

This method makes a backup copy of the features in the assembly. The `backup()` method is used in conjunction with the `restore()` method.

Arguments

None.

Return value

`None.`

Exceptions

None.

clearGeometryCache()

This method deletes the geometry cache. Deleting the geometry cache reduces the amount of memory being used.

Arguments

None.

Return value

`None.`

Exceptions

None.

deleteAllFeatures()

This method deletes all the features in the assembly.

Arguments

None.

Return value

`None.`

Exceptions

None.

deleteFeatures(...)

This method deletes specified features from the assembly.

Required arguments

featureNames A sequence of Strings specifying the feature names that will be deleted from the assembly.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

excludeFromSimulation(...)

This method excludes the specified part instances from the analysis.

Required arguments

instances A sequence of *PartInstance* objects to be excluded from the analysis.

exclude A Bool specifying whether to exclude the selected instances from the analysis or include them.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

featurelistInfo()

This method prints the name and status of all the features in the feature lists.

Arguments

None.

Return value

`None.`

Exceptions

None.

`getMassProperties(...)`

This method returns the mass properties of the assembly, or instances or regions. Only beams, trusses, shells, solids, point, nonstructural mass, and rotary inertia elements are supported.

Required arguments

None.

Optional arguments

regions A MeshElementArray, CellArray, FaceArray, EdgeArray, or list of PartInstance objects specifying the regions whose mass properties are to be queried. The whole assembly is queried by default.

relativeAccuracy A SymbolicConstant specifying the relative accuracy for geometry computation. Possible values are LOW, MEDIUM, and HIGH. The default value is LOW.

useMesh A Boolean specifying whether the mesh should be used in the computation if the geometry is meshed. The default value is False.

specifyDensity A Boolean specifying whether a user-specified density should be used in regions with density errors such as undefined material density. The default value is False.

density A double value specifying the user-specified density value to be used in regions with density errors. The user-specified density should be greater than 0.

specifyThickness A Boolean specifying whether a user-specified thickness should be used in regions with thickness errors such as undefined thickness. The default value is False.

thickness A double value specifying the user-specified thickness value to be used in regions with thickness errors. The user-specified thickness should be greater than 0.

miAboutCenterOfMass A Boolean specifying if the moments of inertia should be evaluated about the center of mass. The default value is True.

miAboutPoint A tuple of three floats specifying the coordinates of the point about which to evaluate the moment of inertia. By default if the moments of inertia are not being evaluated about the center of mass, they will be evaluated about the origin.

Return value

A Dictionary object with the following items:

area: None or a Float specifying the sum of the area of the specified faces. The area is computed only for one side for shells.

areaCentroid: None or a tuple of three Floats representing the coordinates of the area centroid.

volume: None or a Float specifying the volume of the specified regions.

volumeCentroid: None or a tuple of three Floats representing the coordinates of the volume centroid.

massFromMassPerUnitSurfaceArea: None or a Float specifying the mass due to mass per unit surface area.

mass: None or a Float specifying the mass of the specified regions. It is the total mass and includes mass from quantities such as mass per unit surface area.

centerOfMass: None or a tuple of three Floats representing the coordinates of the center of mass.

momentOfInertia: None or a tuple of six Floats representing the moments of inertia about the center of mass or about the point specified.

warnings: A tuple of SymbolicConstants representing the problems encountered while computing the mass properties. Possible SymbolicConstants are:

UNSUPPORTED_ENTITIES: Some unsupported entities exist in the specified regions. The mass properties are computed only for beams, trusses, shells, solids, point and nonstructural mass elements, and rotary inertia elements. The mass properties are not computed for axisymmetric elements, springs, connectors, gaskets, or any other elements.

MISSING_THICKNESS: For some regions, the section definitions are missing thickness values.

ZERO_THICKNESS: For some regions, the section definitions have a zero thickness value.

VARIABLE_THICKNESS: The nodal thickness or field thickness specified for some regions has been ignored.

NON_APPLICABLE_THICKNESS: For some regions, the thickness value is not applicable to the corresponding sections specified on the regions.

MISSING_DENSITY: For some regions, the section definitions are missing material density values.

MISSING_MATERIAL_DEFINITION: For some regions, the material definition is missing.

ZERO_DENSITY: For some regions, the section definitions have a zero material density value.

UNSUPPORTED_DENSITY: For some regions, either a negative material density or a temperature-dependent density has been specified, or the material value is missing for one or more plies in the composite section.

SHELL_OFFSETS: For shells, this method does not account for any offsets specified.

MISSING_SECTION_DEFINITION: For some regions, the section definition is missing.

UNSUPPORTED_SECTION_DEFINITION: The section definition provided for some regions is not supported.

REINFORCEMENTS: This method does not account for any reinforcements specified on the model.

SMEARED_PROPERTIES: For regions with composite section assignments, the density is smeared across the thickness. The volume centroid and center of mass computations for a composite shell use a lumped mass approach where the volume and mass is assumed to be lumped in the plane of the shell. As a result of these approximations the volume centroid, center of mass and moments of inertia might be slightly inaccurate for regions with composite section assignments.

UNSUPPORTED_NON_STRUCTURAL_MASS_ENTITIES: This method does not account for any nonstructural mass on wires.

INCORRECT_MOMENT_OF_INERTIA: For geometry regions with nonstructural mass per volume, the nonstructural mass is assumed to be a point mass at the centroid of the regions. Thus, the moments of inertia might be inaccurate as

the distribution of the nonstructural mass is not accounted for. Use the mesh for accurately computing the moments of inertia.

MISSING_BEAM_ORIENTATIONS: For some regions with beam section assignments, the beam section orientations are missing.

UNSUPPORTED_BEAM_PROFILES: This method supports the Box, Pipe, Circular, Rectangular, Hexagonal, Trapezoidal, I, L, T, Arbitrary, and Tapered beam profiles. Any other beam profile is not supported.

TAPERED_BEAM_MI: Moment of inertia calculations for tapered beams are not accurate.

SUBSTRUCTURE_INCORRECT_PROPERTIES: The user assigned density and thickness is not considered for substructures.

Exceptions

None.

`getAngle(...)`

This method returns the angle between the specified entities.

Required arguments

plane1
A *Face*, *MeshFace*, or a *Datum* object specifying the first plane. The *Datum* object must represent a datum plane. The *plane1* and *line1* arguments are mutually exclusive. One of them must be specified.

plane2
A *Face*, *MeshFace*, or a *Datum* object specifying the second plane. The *Datum* object must represent a datum plane. The *plane2* and *line2* arguments are mutually exclusive. One of them must be specified.

line1
An *Edge*, *MeshEdge*, or a *Datum* object specifying the first curve. The *Datum* object must represent a datum axis. The *plane1* and *line1* arguments are mutually exclusive. One of them must be specified.

line2
An *Edge*, *MeshEdge*, or a *Datum* object specifying the second curve. The *Datum* object must represent a datum axis. The *plane2* and *line2* arguments are mutually exclusive. One of them must be specified.

Optional arguments

commonVertex
If the two selected *Edge* objects have more than one vertex in common, this *Vertex* object specifies the vertex at which to evaluate the angle.

Return value

A Float specifying the angle between the specified entities. If you provide a plane as an argument, Abaqus/CAE computes the angle using the normal to the plane.

Exceptions

None.

`getCoordinates(...)`

This method returns the coordinates of a specified point.

Required arguments

entity A *Vertex*, *Datum* point, *MeshNode*, or *ReferencePoint* specifying the entity to query.

Optional arguments

csys A *DatumCsys* object specifying the desired coordinate system of the returned coordinates. By default, coordinates are given in the global coordinate system.

Return value

A tuple of three Floats representing the coordinates of the specified point.

Exceptions

None.

`getDistance(...)`

Depending on the arguments provided, this method returns one of the following:

- The distance between two points.
- The minimum distance between a point and an edge.
- The minimum distance between two edges.

Required arguments

entity1 A *Vertex*, *Datum* point, *MeshNode*, or *Edge* specifying the first entity from which to measure.

entity2 A *Vertex*, *Datum* point, *MeshNode*, or *Edge* specifying the second entity to which to measure.

Optional arguments

printResults A Boolean that determines whether a verbose output is to be printed. The default is True

Return value

A Float specifying the calculated distance.

Exceptions

None.

getFacesAndVerticesOfAttachmentLines(...)

Given an array of edge objects, this method returns a tuple of dictionary objects. Each object consists of five members including the attachment line and associated face and vertex objects.

Required arguments

edges An *EdgeArray* object that is a sequence of *Edge* objects.

Optional arguments

None.

Return value

A tuple of dictionary objects. Each dictionary contains five items with the following keys:

edge: An *Edge* object specifying the attachment line.

startFace: A *Face* object specifying the face associated with one end of the attachment line.

endFace: A *Face* object specifying the face associated with the other end of the attachment line.

startVertex: A *Vertex* object specifying the vertex associated with one end of the attachment line. This end is also associated with the startFace.

endVertex: A *Vertex* object specifying the vertex associated with the other end of the attachment line. This end is also associated with the endFace.

Exceptions

None.

getSurfaceSections(...)

This method returns a list of the sections assigned to the regions encompassed by the specified surface.

Required arguments

surface A string specifying the Surface name.

Optional arguments

None.

Return value

A tuple of strings representing the section names. If no section names are found, the tuple will contain one empty string.

Exceptions

None.

importEafFile(...)

This method imports an assembly from an EAF file into the root assembly.

Required arguments

filename A String specifying the path to the EAF file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

importParasolidFile(...)

This method imports an assembly from the Parasolid file into the root assembly.

Required arguments

filename A String specifying the path to a Parasolid file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

importCatiaV5File(...)

This method imports an assembly from a CATIA V5 Elysium Neutral file into the root assembly.

Required arguments

filename A String specifying the path to the CATIA V5 Elysium Neutral file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`importEnfFile(...)`

This method imports an assembly from an Elysium Neutral file created by Pro/ENGINEER, I-DEAS, or CATIA V5 into the root assembly.

Required arguments

filename A String specifying the path to the Elysium Neutral file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`importIdeasFile(...)`

This method imports an assembly from an I-DEAS Elysium Neutral file into the root assembly.

Required arguments

filename A String specifying the path to the I-DEAS Elysium Neutral file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`importProEFile(...)`

This method imports an assembly from a Pro/ENGINEER Elysium Neutral file into the root assembly.

Required arguments

filename A String specifying the path to the Pro/ENGINEER Elysium Neutral file from which to import the assembly.

Optional arguments

You use the following optional argument only to import specific instances and their associated parts.

ids A sequence of Ints. Each Int in the sequence is a unique identifier of the occurrence in the assembly tree or component identifier associated with the part in the EAF file. If *ids* is an empty sequence, all occurrences or parts will be imported. The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`makeDependent(...)`

This method converts the specified part instances from independent to dependent part instances.

Required arguments

instances A sequence of *PartInstance* objects to convert to dependent part instances.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`makeIndependent(...)`

This method converts the specified part instances from dependent to independent part instances.

Required arguments

instances A sequence of *PartInstance* objects to convert to independent part instances.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`printAssignedSections()`

This method prints a summary of assigned connector sections.

Arguments

None.

Return value

`None.`

Exceptions

None.

`printConnectorOrientations()`

This method prints a summary of connector orientations.

Arguments

None.

Return value

```
None.
```

Exceptions

None.

projectReferencesOntoSketch(...)

This method projects the specified edges, vertices, and datum points from the assembly onto the specified *ConstrainedSketch* object. The edges, vertices, and datum points appear on the sketch as reference geometry.

Required arguments

sketch The *ConstrainedSketch* object on which the edges, vertices, and datum points are projected.

Optional arguments

filter A SymbolicConstant specifying how to limit the amount of projection. Possible values are ALL_EDGES and COPLANAR_EDGES. If *filter*=COPLANAR_EDGES, edges that are coplanar to the sketching plane are the only candidates for projection. The default value is ALL_EDGES.

upToFeature A Feature object specifying a marker in the feature-based history of the part. Abaqus/CAE projects onto the sketch only the part entities that were created before the feature specified by this marker. By default, all part entities are candidates for projection.

edges A sequence of candidate edges to be projected onto the sketch. By default, all edges are candidates for projection.

vertices A sequence of candidate vertices to be projected onto the sketch. By default, all vertices are candidates for projection.

Return value

```
None.
```

Exceptions

None.

queryCachedStates()

This method displays the position of geometric states relative to the sequence of features in the assembly cache. The output is displayed in the message area.

Arguments

None.

Return value
`None.`

Exceptions

None.

`regenerate()`

This method regenerates the assembly and brings it up to date with the latest values of the assembly parameters. When you modify features of an assembly, it might be convenient to postpone regeneration until you make all your changes, since regeneration can be time consuming. In contrast, when you modify features of a part that is included in the assembly, you should use this command to regenerate the assembly. When you regenerate the assembly, it will reflect the changes that you made to the part.

Arguments

None.

Return value
`None.`

Exceptions

None.

`regenerationWarnings()`

This method prints any regeneration warnings associated with the features.

Arguments

None.

Return value
`None.`

Exceptions

None.

`restore()`

This method restores the parameters of all features in the assembly to the value they had before a failed regeneration. Use the `restore` method after a failed regeneration, followed by a `regenerate` command.

Arguments

None.

Return value
`None.`

Exceptions

None.

resumeAllFeatures()

This method resumes all the suppressed features in the part or assembly.

Arguments

None.

Return value

`None.`

Exceptions

None.

resumeFeatures(...)

This method resumes the specified suppressed features in the assembly.

Required arguments

featureNames A sequence of Strings specifying the names of features to resume.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

resumeLastSetFeatures()

This method resumes the last set of features to be suppressed in the assembly.

Arguments

None.

Return value

`None.`

Exceptions

None.

rotate(...)

This method rotates given instances by the specified amount.

Required arguments

instanceList	A sequence of Strings specifying the names of instances to rotate.
axisPoint	A sequence of three Floats specifying the coordinates of a point on the axis.
axisDirection	A sequence of three Floats specifying the direction of the axis.
angle	A Float specifying the rotation angle in degrees. Use the right-hand rule to determine the direction.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

translate(...)

This method translates given instances by the specified amount.

Required arguments

instanceList	A sequence of Strings specifying the names of instances to translate.
vector	A sequence of three Floats specifying a translation vector.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

saveGeometryCache()

This method caches the current geometry, which improves regeneration performance.

Arguments

None.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the behavior associated with the specified assembly.

Required arguments

regenerateConstraintsTogether A Boolean specifying whether the positioning constraints in the assembly should be regenerated together before regenerating other assembly features. The default value is ON.

If the assembly has position constraint features and you modify the value of *regenerateConstraintsTogether*, Abaqus/CAE will regenerate the assembly features.

Optional arguments

None.

Return value

`None.`

Exceptions

If one or more features in the assembly fails to regenerate: FeatureError: Regeneration failed

`suppressFeatures(...)`

This method suppresses specified features.

Required arguments

featureNames A sequence of Strings specifying the names of features to suppress in the assembly.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

unlinkInstances(...)

This method converts the specified *PartInstance* objects from linked child instances to regular instances. The parts associated with the selected instances will be converted to regular parts as well.

Required arguments

instances A sequence of *PartInstance* objects to be converted to regular part instances.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

writeAcisFile(...)

This method exports the assembly to a named file in ACIS part (SAT) or assembly (ASAT) format.

Required arguments

fileName A String specifying the name of the file to which to write. The file name's extension is used to determine whether a part or assembly is written. Use the file extension `.asat` for the assembly format.

Optional arguments

version A Float specifying the ACIS version. For example, the Float "12.0" corresponds to ACIS Version 12.0. The default value is the current version of ACIS.

Return value

`None.`

Exceptions

None.

writeCADParameters(...)

This method writes the parameters that were imported from the CAD system to a parameter file.

Required arguments

paramFile
A String specifying the parameter file name.

Optional arguments

modifiedParams
A tuple of tuples each containing the part name, the parameter name, and the modified parameter value. Default is an empty tuple.

updatePaths
A Bool specifying whether to update the path of the CAD model file specified in the *parameterFile* to the current directory, if the CAD model is present in the current directory.

Return value
`None.`

Exceptions
None.

`lock()`

This method locks the assembly. Locking the assembly prevents any further changes to the assembly that can trigger regeneration of the assembly.

Arguments
None.

Return value
`None.`

Exceptions
None.

`unlock()`

This method unlocks the assembly. Unlocking the assembly allows it to be regenerated after any modifications to the assembly.

Arguments
None.

Return value
`None.`

Exceptions
None.

setMeshNumberingControl(...)

This method changes the start node and/or element labels on the specified independent part instances before or after Abaqus/CAE generates the meshes. For the meshed instances, Abaqus/CAE changes the node and/or element labels while preserving the original order and incrementation.

Required arguments

instances A sequence of *PartInstance* objects to change the start node and/or element labels.

Optional arguments

startNodeLabel A positive Integer specifying the new start node label.

startElemLabel A positive Integer specifying the new start element label.

Return value

`None.`

Exceptions

None.

copyMeshPattern(...)

This method copies a mesh pattern from a source region consisting of a set of shell elements or element faces onto a target face, mapping nodes and elements in a one-one correspondence between source and target.

Required arguments

None.

Optional arguments

elements A sequence of *MeshElement* objects or a Set object containing elements and specifying the source region.

faces A sequence of *Face* objects that have associated with shell elements or element faces and specifying the source region.

elemFaces A sequence of *MeshFace* objects specifying the source region.

targetFace A *MeshFace* object specifying the target region. The target face can be of a different part instance.

nodes A sequence of MeshNode objects or a Set object containing nodes on the boundary of source region that are to be positioned to the boundary of target face.

coordinates
A sequence of three-dimensional coordinate tuples specifying the coordinates for each of the given nodes. When specified, the number of coordinate tuples must match the number of given nodes, and be ordered to correspond to the given nodes in *ascending order* according to index. These coordinates are positions of the nodes of a mesh that will be the target face corresponding to nodes provided.

Return value
`None.`

Exceptions
None.

`smoothNodes(...)`

This method smooths the given nodes of a native mesh, moving them locally to a more optimal location that improves the quality of the mesh

Required arguments

nodes
A sequence of MeshNode objects or a Set object containing nodes.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

Members

The Assembly object can have the following members:

isOutOfDate
An Int specifying that feature parameters have been modified but that the assembly has not been regenerated. Possible values are 0 and 1.

timeStamp
A Float specifying which gives an indication when the assembly was last modified.

isLocked
An Int specifying whether the assembly is locked or not. Possible values are 0 and 1.

regenerateConstraintsTogether
A Boolean specifying whether the positioning constraints in the assembly should be regenerated together before regenerating other assembly features. The default value is ON.

If the assembly has position constraint features and you modify the value of *regenerateConstraintsTogether*, Abaqus/CAE will regenerate the assembly features.

vertices
A *VertexArray* object specifying all the vertices existing at the assembly level. This member does not provide access to the vertices at the instance level.

edges
An *EdgeArray* object specifying all the edges existing at the assembly level. This member does not provide access to the edges at the instance level.

elements
A *MeshElementArray* object specifying all the elements existing at the assembly level. This member does not provide access to the elements at the instance level.

nodes
A *MeshNodeArray* object specifying all the nodes existing at the assembly level. This member does not provide access to the nodes at the instance level.

instances
A repository of *PartInstance* objects.

datums
A repository of *Datum* objects specifying all *Datum* objects in the assembly.

features
A repository of *Feature* objects specifying all *Feature* objects in the assembly.

featuresById
A repository of *Feature* objects specifying all *Feature* objects in the assembly.

The *Feature* objects in the featuresById repository are the same as the *Feature* objects in the features repository. However, the key to the objects in the featuresById repository is an integer specifying the *ID*, whereas the key to the objects in the features repository is a string specifying the *name*.

surfaces
A repository of *Surface* objects specifying for more information, see *Region commands*.

allSurfaces
A repository of *Surface* objects specifying for more information, see *Region commands*.

allInternalSurfaces
A repository of *Surface* objects specifying picked regions.

sets
A repository of *Set* objects.

allSets
A repository of *Set* objects specifying for more information, see *Region commands*.

allInternalSets
A repository of *Set* objects specifying picked regions.

skins
A repository of *Skin* objects specifying the skins created on the assembly.

stringers
A repository of *Stringer* objects specifying the stringers created on the assembly.

referencePoints
A repository of *ReferencePoint* objects.

modelInstances
A repository of *ModelInstance* objects.

allinstances A *PartInstance* object specifying the PartInstances and A *ModelInstance* object specifying the ModelInstances.

engineeringFeatures An *EngineeringFeature* object.

modelName A String specifying the name of the model to which the assembly belongs.

connectorOrientations A *ConnectorOrientationArray* object.

sectionAssignments A *SectionAssignmentArray* object.

ConnectorOrientation object

The ConnectorOrientation object is used to assign a connector orientation to a connector.

Access

```
import assembly
mdb.models[name].rootAssembly.connectorOrientations[i]
import odbAccess
session.odbs[name].rootAssembly.connectorOrientations[i]
```

ConnectorOrientation(...)

This method creates a ConnectorOrientation object.

Path

```
mdb.models[name].rootAssembly.ConnectorOrientation
session.odbs[name].rootAssembly.ConnectorOrientation
```

Required arguments

region A *Set* object specifying the region to which the orientation is assigned.

Optional arguments

localCsys1 A *DatumCsys* object specifying the local coordinate system of the first connector point. This value may be None, indicating the global coordinate system.

axis1 A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle1 A Float specifying the angle of the additional rotation. The default value is 0.0.

orient2sameAs1 A Boolean specifying whether or not the second connector point is to use the same local coordinate system, axis, and angle as the first point. The default value is ON.

localCsys2 A *DatumCsys* object specifying the local coordinate system of the second connector point. This value may be None, indicating the global coordinate system.

axis2 A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle2 A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

A ConnectorOrientation object.

Exceptions

None.

setValues(...)

This method modifies the ConnectorOrientation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorOrientation* method.

Return value

`None.`

Exceptions

None.

Members

The ConnectorOrientation object has members with the same names and descriptions as the arguments to the *ConnectorOrientation* method.

Feature object

The following commands operate on Feature objects. For more information about the Feature object, see *Feature object*.

Access

```
import assembly
```

AttachmentLines(...)

This method creates a Feature object by creating attachment lines between the given set of source and target faces. The given points are first projected onto the source faces using the specified projection method. The points are then projected normal to the source faces onto the target faces. The user can specify the number of projections or the length of projection vector for projection onto the target faces. The lines are then created between the source face and the closest target face. Subsequent lines are created between the target faces.

Path

mdb.models[*name*].rootAssembly.AttachmentLines

Required arguments

name
A String specifying a unique Feature name.

points
A tuple of points. Each point can be a Vertex, Datum point, Reference point, an Attachment point, orphan mesh Node, or an Interesting point object.

sourceFaces
A sequence of *Face* objects specifying the geometry faces onto which the points are to be projected.

sourceElementFaces
A sequence of *MeshFace* objects specifying the orphan mesh element faces onto which the points are to be projected.

targetFaces
A sequence of *Face* objects specifying the geometry faces on which the attachment lines will terminate.

targetElementFaces
A sequence of *MeshFace* objects specifying the orphan mesh element faces on which the attachment lines will terminate.

Optional arguments

projectionMethod
A SymbolicConstant specifying the method to be used to project onto source faces. Possible values are PROJECT_BY_PROXIMITY and PROJECT_BY_DIRECTION. The default value is PROJECT_BY_PROXIMITY.

projectionDirStartPt
A point specifying the start point of the projection direction to project onto source faces. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting Point object, or a tuple of Floats representing the coordinates of a point.

projectionDirEndPt A point specifying the end point of the projection direction to project onto source faces. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

sourceToTargetProjMethod A SymbolicConstant specifying the method to be used to project onto target faces. Possible values are PROJECT_BY_NUMBER and PROJECT_BY_DISTANCE. The default value is PROJECT_BY_NUMBER.

numProjections An integer specifying the maximum number of layers each point should be projected onto when the source to target projection method is PROJECT_BY_NUMBER.

projectionDistance A float specifying the maximum distance of the projection vector when the source to target projection method is PROJECT_BY_DISTANCE.

flipSourceToTargetDirection A Boolean specifying whether the computed projection direction from the source to the target faces should be flipped.

setName A String specifying a unique set name.

Return value

A Feature object.

Exceptions

None.

Coaxial(...)

This method moves an instance so that its selected face is coaxial with the selected face of a fixed instance.

Path

mdb.models[*name*].rootAssembly.Coaxial

Required arguments

movableAxis A cylindrical or conical Face on the part instance to be moved.

fixedAxis A cylindrical or conical Face on the part instance that remains fixed.

flip A Boolean specifying whether the axes are forward aligned (OFF) or reverse aligned (ON).

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

CoincidentPoint(...)

This method moves an instance so that a specified point is coincident with a specified point of a fixed instance.

Path

mdb.models[*name*].rootAssembly.CoincidentPoint

Required arguments

movablePoint A Vertex, a Datum point, or a ReferencePoint or a mesh node from an orphan mesh on the part instance to be moved.

fixedPoint A Vertex, a Datum point, or a ReferencePoint or a mesh node from an orphan mesh on the part instance to remain fixed.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

EdgeToEdge(...)

This method moves an instance so that its edge is parallel to an edge of a fixed instance.

Path

mdb.models[*name*].rootAssembly.EdgeToEdge

Required arguments

movableAxis A straight Edge, a Datum axis, or an element edge from an orphan mesh on the part instance to be moved.

fixedAxis A straight Edge, a Datum axis, or an element edge from an orphan mesh on the part instance to remain fixed.

flip A Boolean specifying whether the edges are forward aligned (OFF) or reverse aligned (ON).

clearance A Float specifying the distance between the two edges (for two-dimensional and axisymmetric instances only).

Optional arguments

None.

Return value

A Feature Object.

Exceptions

AbaqusException.

FaceToFace(...)

This method moves an instance so that its face is coincident with a face of a fixed instance.

Path

mdb.models[*name*].rootAssembly.FaceToFace

Required arguments

movablePlane A planar face, a Datum plane, or a face from an orphan mesh on the part instance to be moved.

fixedPlane A planar face, a Datum plane, or a face from an orphan mesh on the part instance to remain fixed.

flip A Boolean specifying whether the normals to the faces are forward aligned (OFF) or reverse aligned (ON).

clearance A Float specifying the distance between the two faces.

Optional arguments

None.

Return value

A Feature Object.

Exceptions

AbaqusException.

ParallelCsys(...)

This method moves an instance so that its Datum coordinate system is parallel to a Datum coordinate system of a fixed instance.

Path

mdb.models[*name*].rootAssembly.ParallelCsys

Required arguments

movableCsys	A Datum coordinate system on the part instance to be moved.
fixedCsys	A Datum coordinate system on the part instance to remain fixed.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

ParallelEdge(...)

This method moves an instance so that its edge is parallel to an edge of a fixed instance.

Path

mdb.models[*name*].rootAssembly.ParallelEdge

Required arguments

movableAxis	A straight Edge, a Datum axis, or an element edge from an orphan mesh on the part instance to be moved.
fixedAxis	A straight Edge, a Datum axis, or an element edge from an orphan mesh on the part instance to remain fixed.
flip	A Boolean specifying whether the edges are forward aligned (OFF) or reverse aligned (ON).

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

ParallelFace(...)

This method moves an instance so that its face is parallel to a face of a fixed instance.

Path

mdb.models[*name*].rootAssembly.ParallelFace

Required arguments

movablePlane	A planar face, a Datum plane, or a face from an orphan mesh on the part instance to be moved.
fixedPlane	A planar face, a Datum plane, or a face from an orphan mesh on the part instance to remain fixed.
flip	A Boolean specifying whether the normals to the faces are forward aligned (OFF) or reverse aligned (ON).

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartInstance object

A PartInstance object is an instance of a Part object.

Access

```
import assembly
mdb.models[name].rootAssembly.allInstances[name]
mdb.models[name].rootAssembly.instances[name]
```

Instance(...)

This method creates a PartInstance object and puts it into the instances repository.

Path

```
mdb.models[name].rootAssembly.Instance
```

Required arguments

name A String specifying the repository key. The name must be a valid Abaqus object name.

part A *Part* object to be instanced. If the part does not exist, no PartInstance object is created.

Optional arguments

autoOffset A Boolean specifying whether to apply an auto offset to the new part instance that will offset it from existing part instances. The default value is OFF.

dependent A Boolean specifying whether the part instance is dependent or independent. If *dependent*=OFF, the part instance is independent. The default value is OFF.

Return value

A PartInstance object.

Exceptions

None.

InstanceFromBooleanCut(...)

This method creates a PartInstance in the instances repository after subtracting or cutting the geometries of a group of part instances from that of a base part instance.

Path

```
mdb.models[name].rootAssembly.InstanceFromBooleanCut
```

Required arguments

name
A String specifying the repository key. The name must be a valid Abaqus object name.

instanceToBeCut
A PartInstance specifying the base instance from which to cut other instances.

cuttingInstances
A sequence of PartInstance objects specifying the instances with which to cut the base instance.

Optional arguments

originalInstances
A SymbolicConstant specifying whether the original instances should be suppressed or deleted after the merge operation. Possible values are SUPPRESS or DELETE. The default value is SUPPRESS.

Return value

A PartInstance object.

Exceptions

None.

InstanceFromBooleanMerge(...)

This method creates a PartInstance in the instances repository after merging two or more part instances.

Path

```
mdb.models[name].rootAssembly.InstanceFromBooleanMerge
```

Required arguments

name
A String specifying the repository key. The name must be a valid Abaqus object name.

instances
A sequence of PartInstance objects specifying the part instances to merge.

Optional arguments

keepIntersections
A Boolean specifying whether the boundary intersections of Abaqus native part instances should be retained after the merge operation. The default value is False.

originalInstances
A SymbolicConstant specifying whether the original instances should be suppressed or deleted after the merge operation. Possible values are SUPPRESS or DELETE. The default value is SUPPRESS.

domain
A SymbolicConstant specifying whether geometry or mesh of the specified part instances is to be merged. Possible values are GEOMETRY, MESH or BOTH. The default value is GEOMETRY.

mergeNodes
A SymbolicConstant specifying which nodes of the specified part instances should be considered for merging. This argument is only applicable if *domain* is MESH. Possible values are BOUNDARY_ONLY, ALL, or NONE. The default value is BOUNDARY_ONLY.

nodeMergingTolerance
A Float specifying the maximum distance between nodes of the specified part instances that will be merged and replaced with a single node in the new part. The location of the new node is the average position of the deleted nodes. This argument is only applicable if *domain* is MESH. The default value is 10^{-6}.

removeDuplicateElements
A Boolean specifying whether elements with the same connectivity in the new part will be merged into a single element. This argument is only applicable if *domain* is MESH. The default value is True.

Return value

A PartInstance object.

Exceptions

None.

LinearInstancePattern(...)

This method creates multiple PartInstance objects in a linear pattern and puts them into the instances repository.

Path

```
mdb.models[name].rootAssembly.LinearInstancePattern
```

Required arguments

instanceList
A sequence of Strings specifying the names of instances to pattern.

number1
An Int specifying the total number of instances, including the original instances, that appear along the first direction in the pattern.

spacing1
A Float specifying the spacing between instances along the first direction in the pattern.

number2
An Int specifying the total number of instances, including the original instances, that appear along the second direction in the pattern.

spacing2
A Float specifying the spacing between instances along the second direction in the pattern.

Optional arguments

direction1
A sequence of three Floats specifying a vector along the first direction. The default value is (1.0, 0.0, 0.0).

direction2
A sequence of three Floats specifying a vector along the second direction. The default value is (0.0, 1.0, 0.0).

Return value

A sequence of PartInstance objects.

Exceptions

None.

RadialInstancePattern(...)

This method creates multiple PartInstance objects in a radial pattern and puts them into the instances repository.

Path

```
mdb.models[name].rootAssembly.RadialInstancePattern
```

Required arguments

instanceList
A sequence of Strings specifying the names of instances to pattern.

number
An Int specifying the total number of instances, including the original instances, that appear in the radial pattern.

totalAngle
A Float specifying the total angle in degrees between the first and last instance in the pattern. A positive angle corresponds to a counter-clockwise direction. The values 360° and −360° represent a special case where the pattern makes a full circle. In this case, because the copy would overlay the original, the copy is not placed at the last position. Possible values are $-360.0 \leq totalAngle \leq 360.0$.

Optional arguments

point
A sequence of three Floats specifying the center of the radial pattern. The default value is (0.0, 0.0, 0.0).

axis
A sequence of three Floats specifying the central axis of the radial pattern. The default value is (0.0, 0.0, 1.0).

Return value

A sequence of PartInstance objects.

Exceptions

None.

checkGeometry(...)

This method checks the validity of the geometry of the part instance and prints a count of all topological entities on the part instance (faces, edges, vertices, etc.).

Required arguments

None.

Optional arguments

detailed A Boolean specifying whether detailed output will be printed to the replay file. The default value is OFF.

level An Int specifying which level of checking is performed. Values can range from 20 to 70, with higher values reporting less and less important errors. The default value is 20, which reports all critical errors. When the default value is used, the stored validity status is updated to agree with the result of this check.

Return value

`None.`

Exceptions

An exception is thrown if this is a dependent part instance and *level* was either not specified or was set to 20, because the validity status cannot be updated for a dependent part instance. In this case, this command should be called on the *Part* instead.	The geometry of dependent part instances cannot be changed.

Contact(...)

This method translates an instance along the specified direction until it is in contact with a fixed instance.

Required arguments

movableList A sequence of *Face* or *Edge* objects on the part instance to be moved.

fixedList A sequence of *Face* or *Edge* objects on the part instance to remain fixed.

direction A sequence of three Floats specifying the direction of contact.

clearance A Float specifying the distance between the two faces along the direction of contact.

Optional arguments

isFaceEdges A Boolean specifying how Abaqus calculates the contact. If *isFaceEdges* is OFF, contact is computed from the movable face to the fixed face. If *isFaceEdges* is ON, contact is computed using only the edges of the movable face and not its interior. The default value is OFF.

Return value

A *Feature* object.

Exceptions

None.

ConvertConstraints()

This method converts the position constraints of an instance to absolute positions. The method deletes the constraint features on the instance but preserves the position in space.

Arguments

None.

Return value

`None.`

Exceptions

None.

getPosition()

This method prints the sum of the translations and rotations applied to the PartInstance object.

Arguments

None.

Return value

`None.`

Exceptions

None.

getRotation()

This method returns a tuple including the point of rotation, axis of rotation, and rotation angle (in degrees).

Arguments

None.

Return value

A tuple including the point of rotation, axis of rotation, and rotation angle (in degrees).

Exceptions

None.

`getTranslation()`

This method returns a tuple of three Floats representing translation in the *X*-, *Y*-, and *Z*-directions.

Arguments

None.

Return value

A tuple of three Floats representing the translation.

Exceptions

None.

`replace(...)`

This method replaces one instance with an instance of another part.

Required arguments

instanceOf A *Part* object specifying which Part will be instanced in place of the original *Part*.

Optional arguments

applyConstraints A Boolean specifying whether to apply existing constraints on the new instance or to position the new instance in the same place as the original instance. The default value is True. A value of False indicates that constraints applies to the instance are deleted will be deleted from the feature list.

Return value

`None.`

Exceptions

None.

`rotateAboutAxis(...)`

This method translates an instance by the specified amount.

Required arguments

axisPoint A sequence of three Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point on the axis.

axisDirection A sequence of three Floats specifying the direction vector of the axis.

angle A Float specifying the rotation angle in degrees. Use the right-hand rule to determine the direction.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`translate(...)`

This method translates an instance by the specified amount.

Required arguments

vector A sequence of three Floats specifying a translation vector.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`translateTo(...)`

This method translates an instance along the specified direction until it is in contact with a fixed instance.

Required arguments

movableList A sequence of *Face* or *Edge* objects on the part instance to be moved.

fixedList A sequence of *Face* or *Edge* objects on the part instances to remain fixed.

direction A sequence of three Floats specifying the direction of contact.

clearance A Float specifying the distance between the two faces along the direction of contact.

Optional arguments

vector A sequence of three Floats specifying a translation vector. If this argument is specified, the movable instance will be translated by the specified amount without solving for the actual contact.

Return value

A *Feature* object.

Exceptions

None.

Members

The PartInstance object can have the following members:

analysisType A SymbolicConstant specifying the part type. Possible values are DEFORMABLE_BODY, EULERIAN, DISCRETE_RIGID_SURFACE, and ANALYTIC_RIGID_SURFACE.

cells A *CellArray* object.

datums A repository of *Datum* objects.

dependent A Boolean specifying whether the part instance is dependent or independent. If *dependent*=OFF, the part instance is independent. The default value is OFF.

edges An *EdgeArray* object.

elemEdges A repository of *MeshEdge* objects specifying all the element edges in the part instance. For a given element and a given edge index on a given face within that element, the corresponding *MeshEdge* object can be retrieved from the repository by using the key calculated as (i*32 + j*4 + k), where i, j, and k are zero-based element, face, and edge indices, respectively.

elementEdges A *MeshEdgeArray* object.

elementFaces A *MeshFaceArray* object.

elements A *MeshElementArray* object.

elemFaces A repository of *MeshFace* objects specifying all the element faces in the part instance. For a given element and a given face index within that element, the corresponding *MeshFace* object can be retrieved from the repository by using the key calculated as (i*8 + j), where i and j are zero-based element and face indices, respectively.

excludedFromSimulation A Boolean specifying whether the part instance is excluded from the simulation. If *excludedFromSimulation*=ON, the part instance is excluded from the simulation. The default value is OFF.

faces

A *FaceArray* object.

geometryValidity

A Boolean specifying the validity of the geometry of the instance. The value is computed, but it can be set to ON to perform feature and mesh operations on an invalid instance. There is no guarantee that such operations will work if the instance was originally invalid.

ignoredEdges

An *IgnoredEdgeArray* object.

ignoredVertices

An *IgnoredVertexArray* object.

name

A String specifying the repository key. The name must be a valid Abaqus object name.

nodes

A *MeshNodeArray* object.

part

A *Part* object specifying the instanced part.

partName

A String specifying the name of the part from which the instance was created.

referenceNode

An Int specifying the reference node number. This member is valid only if *analysisType*=DISCRETE_RIGID_SURFACE or ANALYTIC_RIGID_SURFACE.

referencePoints

A repository of *ReferencePoint* objects.

sets

A repository of *Set* objects specifying the sets created on the part. For more information, see *Region commands*.

skins

A repository of *Skin* objects specifying the skins created on the part. For more information, see *Region commands*.

stringers

A repository of *Stringer* objects specifying the stringers created on the part. For more information, see *Region commands*.

surfaces

A repository of *Surface* objects specifying the surfaces created on the part. For more information, see *Region commands*.

vertices

A *VertexArray* object.

ModelInstance object

A ModelInstance object is an instance of a Model.

Access

```
import assembly
mdb.models[name].rootAssembly.modelInstances[i]
```

Instance(...)

This method creates a ModelInstance object and puts it into the instances repository.

Path

```
mdb.models[name].rootAssembly.Instance
```

Required arguments

name The repository key. The name must be a valid Abaqus object name.

model A *Model* object to be instanced. If the model does not exist, no ModelInstance object is created.

Optional arguments

autoOffset A Boolean specifying whether to apply an auto offset to the new instance that will offset it from existing instances. The default value is OFF.

Return value

A ModelInstance object.

Exceptions

None.

ConvertConstraints()

This method converts the position constraints of an instance to absolute positions. The method deletes the constraint features on the instance but preserves the position in space.

Arguments

None.

Return value

```
None.
```

Exceptions

None.

getPosition()

This method prints the sum of the translations and rotations applied to the ModelInstance object.

Arguments

None.

Return value

`None.`

Exceptions

None.

replace(...)

This method replaces one instance with an instance of another model.

Required arguments

instanceOf

A *Model* object to be instanced. If the model does not exist, no ModelInstance object is created.

Optional arguments

applyConstraints

A Boolean specifying whether to apply existing constraints on the new instance or to position the new instance in the same place as the original instance. The default value is True. A value of False indicates that constraints applies to the instance are deleted will be deleted from the feature list.

Return value

`None.`

Exceptions

None.

translate(...)

This method translates an instance by the specified amount.

Required arguments

vector

A sequence of three Floats specifying a translation vector.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The ModelInstance object can have the following members:

sets
A repository of *Set* objects specifying the sets created on the assembly. For more information, see *Region commands*.

surfaces
A repository of *Surface* objects specifying the surfaces created on the assembly. For more information, see *Region commands*.

vertices
A *VertexArray* object.

edges
An *EdgeArray* object.

elements
A *MeshElementArray* object.

nodes
A *MeshNodeArray* object.

datums
A repository of *Datum* objects.

referencePoints
A repository of *ReferencePoint* objects.

Model object

The following commands operate on Model objects. For more information about the Model object, see *Model object*.

Access

```
import assembly
```

Instance(...)

This method copies a *PartInstance* object from the specified model and creates a new *PartInstance* object.

Path

mdb.models[*name*].Instance

Required arguments

name	A String specifying the repository key.
objectToCopy	A *PartInstance* object to be copied.

Optional arguments

None.

Return value

A Model object.

Exceptions

None.

convertAllSketches(...)

This method converts all sketches from Abaqus 6.5 or earlier to the equivalent *ConstrainedSketch* objects.

Required arguments

None.

Optional arguments

regenerate	A Boolean specifying if all the features in assembly as well as in all the parts in the model should be regenerated after the conversion. The default value is True.
convertReversedSketches	A Boolean specifying whether sketches in analytic rigid parts should be converted even if they cause the orientation of surfaces defined on them to be flipped. The default value is True.

Return value

A list of strings describing any warnings or errors encountered during the conversion process.

Exceptions

None.

`linkInstances(...)`

This method links the selected *PartInstance* objects to the corresponding *PartInstance* objects from the specified models. If all instances of a Part are selected for linking, the Part will be linked as well. If not, a new linked child Part object will be created and added to the repository.

Required arguments

instancesMap
A tuple of tuples containing the instance name to be linked and the corresponding *PartInstance* object to which it will be linked.

Optional arguments

None.

Return value

A list of strings describing any warnings or errors encountered during the conversion process.

Exceptions

None.

Basic geometry commands

The basic geometry commands create geometric entities that are referenced by Part objects, Instance objects, and GeometrySet objects.

In this section:

- *Cell object*
- *CellArray object*
- *Edge object*
- *EdgeArray object*
- *Face object*
- *FaceArray object*
- *IgnoredEdge object*
- *IgnoredEdgeArray object*
- *IgnoredVertex object*
- *IgnoredVertexArray object*
- *InterestingPoint object*
- *ModelDot object*
- *ReferencePoint object*
- *Transform object*
- *Vertex object*
- *VertexArray object*

Cell object

Cells are volumetric regions of geometry.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].cells[i]
mdb.models[name].parts[name].allSets[name].cells[i]
mdb.models[name].parts[name].cells[i]
mdb.models[name].parts[name].sets[name].cells[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].cells[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name].cells[i]
mdb.models[name].rootAssembly.allInternalSets[name].cells[i]
mdb.models[name].rootAssembly.allSets[name].cells[i]
mdb.models[name].rootAssembly.instances[name].cells[i]
mdb.models[name].rootAssembly.instances[name].sets[name].cells[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].cells[i]
mdb.models[name].rootAssembly.sets[name].cells[i]
```

`getSize(...)`

This method returns a Float indicating the volume of the cell.

Required arguments

None.

Optional arguments

printResults	A Boolean that determines whether a verbose output is to be printed. The default is True.

Return value

A Float.

Exceptions

None.

`getFaces()`

This method returns a sequence consisting of the face IDs of the faces which bound the cell.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getEdges()

This method returns a sequence consisting of the edge IDs of the edges on the cell.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getVertices()

This method returns a sequence consisting of the vertex IDs of the vertices on the cell.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getAdjacentCells()

This method returns an array of cell objects that share at least one face of the cell.

Arguments

None.

Return value

A *CellArray* object which is a sequence of Cell objects.

Exceptions

None.

getNodes()

This method returns an array of node objects that are associated with the cell.

Arguments

None.

Return value

A *MeshNodeArray* object which is a sequence of *MeshNode* objects.

Exceptions

None.

`getElements()`

This method returns an array of element objects that are associated with the cell.

Arguments

None.

Return value

A *MeshElementArray* object which is a sequence of *MeshElement* objects.

Exceptions

None.

`getCADAttributes(...)`

This method returns an array of CAD attribute strings associated with the cell when the part was created from CAD data.

Arguments

None.

Return value

An array of String.

Exceptions

None.

Members

The Cell object has the following members:

index

An Int specifying the index of the cell in the CellArray.

isReferenceRep

A Boolean specifying whether the cell belongs to the reference representation of the Part or Instance.

pointOn

A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the cell.

featureName

A tuple of Floats specifying the name of the feature that created this cell.

instanceName

A tuple of Floats specifying the name of the part instance for this cell (if applicable).

CellArray object

The CellArray is a sequence of *Cell* objects.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].cells
mdb.models[name].parts[name].allSets[name].cells
mdb.models[name].parts[name].cells
mdb.models[name].parts[name].sets[name].cells
import assembly
mdb.models[name].rootAssembly.allInstances[name].cells
mdb.models[name].rootAssembly.allInstances[name].sets[name].cells
mdb.models[name].rootAssembly.allInternalSets[name].cells
mdb.models[name].rootAssembly.allSets[name].cells
mdb.models[name].rootAssembly.instances[name].cells
mdb.models[name].rootAssembly.instances[name].sets[name].cells
mdb.models[name].rootAssembly.modelInstances[i].sets[name].cells
mdb.models[name].rootAssembly.sets[name].cells
```

CellArray(...)

This method creates a CellArray object.

Path

part.CellArray

Required arguments

cells A list of *Cell* objects.

Optional arguments

None.

Return value

A CellArray object.

Exceptions

None.

findAt(...)

This method returns the object or objects in the CellArray located at the given coordinates. `findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any entity that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities). The arbitrary point must not be shared by a second cell. If two cells intersect or coincide at the arbitrary point, `findAt` chooses the first cell that it encounters, and you should not rely on the return value being consistent.

`findAt` will always try to find objects among all the cells in the part or assembly instance and will not restrict itself to a subset even if the CellArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either a Cell object or a sequence of Cell objects based on the type of input. If *coordinates* is a sequence of Floats, `findAt` returns the Cell object at that point. If *coordinates* is a sequence of sequence of Floats, `findAt` returns a sequence of Cell objects at the given locations. The sequence of sequence of Floats must be a sequence of sequence of point and normal data, not a sequence of point data. For example,

```
cells1 = myCrackedBlockInstance.cells.findAt(
                                ((5.5, -8.3, 294.2),),
                                ((12.1, -8.3, 287.8),),
                                ((14.4, -10.4, 285.5),),)
```

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

A Cell object.

Exceptions

None.

`getExteriorFaces()`

This method returns the cell faces on the exterior of the CellArray. That is, it returns the faces that are referenced by exactly one of the cells in the sequence.

Arguments

None.

Return value

A FaceArray object representing the faces on the exterior of the cells.

Exceptions

None.

`getSequenceFromMask(...)`

This method returns the object or objects in the CellArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A Cell object or a sequence of Cell objects.

Exceptions

An exception occurs if the resulting sequence is empty. Error: The mask results in an empty sequence

`getMask()`

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

`getByBoundingBox(...)`

This method returns an array of cell objects that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin A float specifying the minimum *X*-boundary of the bounding box.

yMin A float specifying the minimum *Y*-boundary of the bounding box.

zMin A float specifying the minimum *Z*-boundary of the bounding box.

xMax A float specifying the maximum *X*-boundary of the bounding box.

yMax A float specifying the maximum *Y*-boundary of the bounding box.

zMax A float specifying the maximum *Z*-boundary of the bounding box.

Return value

A CellArray object, which is a sequence of *Cell* objects.

Exceptions

None.

`getByBoundingCylinder(...)`

This method returns an array of cell objects that lie within the specified bounding cylinder.

Required arguments

center1 A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the first end of the cylinder.

center2 A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the second end of the cylinder.

radius A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

A CellArray object, which is a sequence of *Cell* objects.

Exceptions

None.

`getByBoundingSphere(...)`

This method returns an array of cell objects that lie within the specified bounding sphere.

Required arguments

center A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the sphere.

radius A float specifying the radius of the sphere.

Optional arguments

None.

Return value

A CellArray object, which is a sequence of *Cell* objects.

Exceptions

None.

`getBoundingBox()`

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the cell sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

high: a tuple of three floats representing the maximum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

Exceptions

None.

Members

The CellArray object has no members.

Edge object

Edges are one-dimensional regions of geometry.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].edges[i]
mdb.models[name].parts[name].allInternalSurfaces[name].edges[i]
mdb.models[name].parts[name].allSets[name].edges[i]
mdb.models[name].parts[name].allSurfaces[name].edges[i]
mdb.models[name].parts[name].edges[i]
mdb.models[name].parts[name].sets[name].edges[i]
mdb.models[name].parts[name].surfaces[name].edges[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].edges[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name].edges[i]
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].edges[i]
mdb.models[name].rootAssembly.allInternalSets[name].edges[i]
mdb.models[name].rootAssembly.allInternalSurfaces[name].edges[i]
mdb.models[name].rootAssembly.allSets[name].edges[i]
mdb.models[name].rootAssembly.allSurfaces[name].edges[i]
mdb.models[name].rootAssembly.edges[i]
mdb.models[name].rootAssembly.instances[name].edges[i]
mdb.models[name].rootAssembly.instances[name].sets[name].edges[i]
mdb.models[name].rootAssembly.instances[name].surfaces[name].edges[i]
mdb.models[name].rootAssembly.modelInstances[i].edges[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].edges[i]
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]\
.edges[i]
mdb.models[name].rootAssembly.sets[name].edges[i]
mdb.models[name].rootAssembly.surfaces[name].edges[i]
```

`isTangentFlipped()`

This method determines whether the tangent to the edge is flipped from its default direction by the use of the `flipTangent` method on a Part object.

Arguments

None.

Return value

A Boolean value of True if the tangent is flipped and False if not.

Exceptions

None.

`getCurvature(...)`

This method returns curvature information at a location on the edge.

Required arguments

parameter A Float specifying the normalized parameter location on the edge where the curvature is to be computed. This argument is mutually exclusive with the argument *point*.

point A tuple of *X*-, *Y*-, and *Z*-coordinates of a point at which the curvature is to be computed. If *point* does not lie on the edge an attempt is made to project it onto the edge and use the projected point.

Optional arguments

None.

Return value

A dictionary with keys 'evaluationPoint', 'curvature', 'radius', and 'tangent', where 'evaluationPoint' specifies the location at which the curvature was computed; 'curvature' specifies the curvature vector at that location; 'radius' is the radius of curvature; and 'tangent' specifies the tangent to the edge at that location.

Exceptions

The given edge is straight.

`getFaces()`

This method returns a sequence consisting of the face ids of the faces which share this edge.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

`getAdjacentEdges()`

This method returns an array of Edge objects that share at least one vertex of the edge.

Arguments

None.

Return value

An *EdgeArray* object, which is a sequence of Edge objects.

Exceptions

None.

`getEdgesByEdgeAngle(...)`

This method returns an array of Edge objects that are obtained by recursively finding adjacent edges that are at an angle of less than or equal to the specified face angle.

Required arguments

angle A float specifying the value of the face angle in degrees.

Optional arguments

None.

Return value

An *EdgeArray* object, which is a sequence of Edgeobjects.

Exceptions

None.

`getNodes()`

This method returns an array of node objects that are associated with the edge.

Arguments

None.

Return value

A *MeshNodeArray* object, which is a sequence of *MeshNode* objects.

Exceptions

None.

`getElements()`

This method returns an array of element objects that are associated with the edge.

Arguments

None.

Return value

A *MeshElementArray* object which is a sequence of *MeshElement* objects.

Exceptions

None.

getRadius()

This method returns the radius of circular edges.

Arguments

None.

Return value

A Float specifying the radius.

Exceptions

The given edges is not circular.

getSize(...)

This method returns a Float indicating the length of the edge.

Required arguments

None.

Optional arguments

printResults	A Bool specifying whether verbose output is printed. The default is True.

Return value

A Float.

Exceptions

None.

getVertices()

This method returns a sequence of indices of the vertices that bound this edge. The first index refers to the vertex where the normalized curve parameter = 0.0, and the second index refers to the vertex where the normalized curve parameter = 1.0. If the edge is a closed curve, only one vertex index is returned.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

Members

The Edge object has the following members:

index
An Int specifying the index of the edge in the EdgeArray.

isReferenceRep
A Boolean specifying whether the edge belongs to the reference representation of the Part or Instance.

pointOn
A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the edge.

featureName
A tuple of Floats specifying the name of the feature that created this edge.

instanceName
A tuple of Floats specifying the name of the part instance for this edge (if applicable).

EdgeArray object

The EdgeArray is a sequence of *Edge* objects. If the part is modified, then EdgeArray must be updated for that part.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].edges
mdb.models[name].parts[name].allInternalSurfaces[name].edges
mdb.models[name].parts[name].allSets[name].edges
mdb.models[name].parts[name].allSurfaces[name].edges
mdb.models[name].parts[name].edges
mdb.models[name].parts[name].sets[name].edges
mdb.models[name].parts[name].surfaces[name].edges
import assembly
mdb.models[name].rootAssembly.allInstances[name].edges
mdb.models[name].rootAssembly.allInstances[name].sets[name].edges
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].edges
mdb.models[name].rootAssembly.allInternalSets[name].edges
mdb.models[name].rootAssembly.allInternalSurfaces[name].edges
mdb.models[name].rootAssembly.allSets[name].edges
mdb.models[name].rootAssembly.allSurfaces[name].edges
mdb.models[name].rootAssembly.edges
mdb.models[name].rootAssembly.instances[name].edges
mdb.models[name].rootAssembly.instances[name].sets[name].edges
mdb.models[name].rootAssembly.instances[name].surfaces[name].edges
mdb.models[name].rootAssembly.modelInstances[i].edges
mdb.models[name].rootAssembly.modelInstances[i].sets[name].edges
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name].edges
mdb.models[name].rootAssembly.sets[name].edges
mdb.models[name].rootAssembly.surfaces[name].edges
```

EdgeArray(...)

This method creates an EdgeArray object.

Path

part.EdgeArray

Required arguments

edges A list of *Edge* objects.

Optional arguments

None.

Return value

A EdgeArray object.

Exceptions

None.

findAt(...)

This method returns the object or objects in the EdgeArray located at the given coordinates.

`findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any edge that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities). The arbitrary point must not be shared by a second edge. If two edges intersect or coincide at the arbitrary point, `findAt` chooses the first edge that it encounters, and you should not rely on the return value being consistent.

`findAt` will always try to find objects among all the edges in the part or assembly instance and will not restrict itself to a subset even if the EdgeArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either an Edge object or a sequence of Edge objects based on the type of input.

- If *coordinates* is a sequence of Floats, `findAt` returns the Edge object at that point.
- If you omit the *coordinates* keyword argument, `findAt` accepts as arguments a sequence of sequence of floats in the following format:

```
edges = e.findAt(((20.19686, -169.513997, 27.798593), ),
                 ((19.657627, -167.295749, 27.056402), ),
                 ((18.274129, -157.144741, 25.15218), ))
```

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

An Edge object or a sequence of Edge objects.

Exceptions

None.

getClosest(...)

This method returns an object or objects in the EdgeArray closest to the given set of points, where the given points need not lie on the edges in the EdgeArray.

Required arguments

coordinates

A sequence of a sequence of floats, where each sequence of floats describes the *X*-, *Y*-, and *Z*-coordinates of a point.

```
r=e.getClosest(coordinates=((20.0,20.0,10.0),(-1.0, -15.0, 15),))

r.keys()

[0, 1]

r[0]

(mdb.models['Model-1'].parts['Part-1'].edges[3],
                                    (15.7090625762939, 20.0, 10.0))
```

Optional arguments

searchTolerance

A double specifying the distance within which the closest object must lie. The default value is half of the parent part/instance size.

Return value

This method returns a dictionary object. The key to the dictionary object is the position of the input point in the tuple specified in the *coordinates* starting at index 0. If a closest edge could be found then the value is a sequence consisting of two objects. The first object in the sequence is an *Edge* that is close to the input point referred to by the key. The second object in the sequence is a sequence of floats that specifies the *X*-, *Y*-, and *Z*-location of the closest point on the *Edge* to the given point. See program listing above.

Exceptions

None.

`getSequenceFromMask(...)`

This method returns the object or objects in the EdgeArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When a large number of objects are involved, this method is highly efficient.

Required arguments

mask

A String specifying the object or objects.

Optional arguments

None.

Return value

An Edge object or a sequence of Edge objects.

Exceptions

An exception occurs if the resulting sequence is empty.	Error: The mask results in an empty sequence

`getMask()`

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

`getByBoundingBox(...)`

This method returns an array of edge objects that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin	A float specifying the minimum *X*-boundary of the bounding box.
yMin	A float specifying the minimum *Y*-boundary of the bounding box.
zMin	A float specifying the minimum *Z*-boundary of the bounding box.
xMax	A float specifying the maximum *X*-boundary of the bounding box.
yMax	A float specifying the maximum *Y*-boundary of the bounding box.
zMax	A float specifying the maximum *Z*-boundary of the bounding box.

Return value

An EdgeArray object, which is a sequence of *Edge* objects.

Exceptions

None.

`getByBoundingCylinder(...)`

This method returns an array of edge objects that lie within the specified bounding cylinder.

Required arguments

center1	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the first end of the cylinder.
center2	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the second end of the cylinder.
radius	A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

An EdgeArray object, which is a sequence of *Edge* objects.

Exceptions

None.

`getByBoundingSphere(...)`

This method returns an array of edge objects that lie within the specified bounding sphere.

Required arguments

center	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the sphere.
radius	A float specifying the radius of the sphere.

Optional arguments

None.

Return value

An EdgeArray object, which is a sequence of *Edge* objects.

Exceptions

None.

`getBoundingBox()`

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the edge sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

high: a tuple of three floats representing the maximum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

Exceptions

None.

Members

The EdgeArray object has no members.

Face object

Faces are two-dimensional regions of geometry.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].faces[i]
mdb.models[name].parts[name].allInternalSurfaces[name].faces[i]
mdb.models[name].parts[name].allSets[name].faces[i]
mdb.models[name].parts[name].allSurfaces[name].faces[i]
mdb.models[name].parts[name].faces[i]
mdb.models[name].parts[name].sets[name].faces[i]
mdb.models[name].parts[name].surfaces[name].faces[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].faces[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name].faces[i]
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].faces[i]
mdb.models[name].rootAssembly.allInternalSets[name].faces[i]
mdb.models[name].rootAssembly.allInternalSurfaces[name].faces[i]
mdb.models[name].rootAssembly.allSets[name].faces[i]
mdb.models[name].rootAssembly.allSurfaces[name].faces[i]
mdb.models[name].rootAssembly.instances[name].faces[i]
mdb.models[name].rootAssembly.instances[name].sets[name].faces[i]
mdb.models[name].rootAssembly.instances[name].surfaces[name].faces[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].faces[i]
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]\
.faces[i]
mdb.models[name].rootAssembly.sets[name].faces[i]
mdb.models[name].rootAssembly.surfaces[name].faces[i]
```

getCentroid()

This method returns the centroid of a face.

Arguments

None.

Return value

A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the centroid of the face.

Exceptions

None.

getCurvature(...)

This method returns information about the curvature at a location on the face.

Required arguments

point
A tuple specifying the *X*-, *Y*-, and *Z* coordinates of the point where the curvature is desired. If the *point* does not lie on the face it will be projected onto the face. This argument and *uParam* and *vParam* are mutually exclusive.

uParam
A Float specifying the normalized *U* parameter value at which the curvature is to be computed. This value must lie between (0,1). *vParam* must also be specified. This argument is mutually exclusive with *point*.

vParam
A Float specifying the normalized *V* parameter value at which the curvature is to be computed. This value must lie between (0,1).

Optional arguments

None.

Return value

A dictionary with keys 'evaluationPoint', 'principalAxis1', 'principalAxis2', 'curvature1', 'curvature2' and 'gaussianCurvature'. Where the evaluationPoint specifies the location at which the curvature was evaluated. 'principalAxis1' and 'principalAxis2' refer to the vectors specifying the two principal axes of the face. 'curvature1' and 'curvature2' specify the curvatures along the two principal axes.

Exceptions

None.

`getElements()`

This method returns an array of element objects that are associated with the face.

Arguments

None.

Return value

A *MeshElementArray* object which is a sequence of *MeshElement* objects.

Exceptions

None.

`getElementFaces(...)`

This method returns an array of mesh face objects. Each mesh face object contains the element label and the side of the element that lies on the geometric face.

Required arguments

None.

Optional arguments

faceSide

A symbolic constant specifying the side of the geometric face of a solid from which the element faces should be retrieved. Possible values are *SIDE1*, *SIDE2*, and *BOTH_SIDES*. The default value is *BOTH_SIDES*. For shell faces this option is ignored.

Return value

A *MeshFaceArray* object which is a sequence of *MeshFace* objects.

Exceptions

None.

`getNodes(...)`

This method returns an array of mesh node objects. Each mesh node object contains the label of the node that lies on the geometric face.

Required arguments

None.

Optional arguments

faceSide

A symbolic constant specifying the side of the geometric face of a solid from which the nodes should be retrieved. Possible values are xo*SIDE1*, *SIDE2*, and *BOTH_SIDES*. The default value is *BOTH_SIDES*. For shell faces and for faces with compatible meshes on either sides this option is ignored. Otherwise, the nodes on the specified side(s) of the face are output.

Return value

A *MeshNodeArray* object which is a sequence of *MeshNode* objects.

Exceptions

None.

`getNormal(...)`

This method returns the normal to a face at the location specified by the *pointOn* member. The normal at a different location on the face can be obtained by specifying the optional *point* argument.

Required arguments

None.

Optional arguments

point

A tuple specifying the *X*-, *Y*-, and *Z* coordinates of the point where the normal to the face is desired. If the *point* does not lie on the face it will be projected onto the face.

Return value

A sequence of Floats specifying the *X*-, *Y*-, and *Z*-components of the normal to the face.

Exceptions

An exception is raised if the optional argument *point* is provided but the point cannot be projected onto the face.

getSize(...)

This method returns a Float indicating the area of the face.

Required arguments

None.

Optional arguments

printResults A Bool specifying whether verbose output is printed. The default value is True.

Return value

A Float.

Exceptions

None.

getEdges()

This method returns a sequence consisting of the edge ids of the edges on the face.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getVertices()

This method returns a sequence consisting of the vertex ids of the vertices of the face.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getCells()

This method returns a sequence consisting of the cell ids of the cells to which this face belongs.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

getAdjacentFaces()

This method returns an array of face objects that share at least one edge of the face.

Arguments

None.

Return value

A *FaceArray* object which is a sequence of Face objects.

Exceptions

None.

getFacesByFaceAngle(...)

This method returns an array of Face objects that are obtained by recursively finding adjacent faces that are at an angle of less than or equal to the specified angle.

Required arguments

angle	A float specifying the value of the face angle.

Optional arguments

None.

Return value

A *FaceArray* object, which is a sequence of Face objects.

Exceptions

None.

getFacesByCurvature()

This method returns an array of Face objects that are obtained by recursively finding adjacent faces that share the same curvature.

Arguments

None.

Return value

A *FaceArray* object, which is a sequence of Face objects.

Exceptions

None.

isNormalFlipped()

This method determines whether the normal to the face is flipped from its default direction by the use of the `flipNormal` method on a Part object.

Arguments

None.

Return value

A Boolean value of True if the normal is flipped and False if not.

Exceptions

None.

getCADAttributes(...)

This method returns an array of CAD attribute strings associated with the Face when the part was created from CAD data.

Arguments

None.

Return value

An array of String

Exceptions

None.

Members

The Face object has the following members:

index

An Int specifying the index of the face in the FaceArray.

isReferenceRep

A Boolean specifying whether the face belongs to the reference representation of the Part or Instance.

pointOn

A tuple of tuples of Floats specifying the coordinates. For a face of a shell *pointOn* specifies the *X*-, *Y*-, and *Z*-coordinates of a point located on the face and the *X*-, *Y*-, and *Z*-components of the normal to the face.

For a face of a solid *pointOn* specifies the *X*-, *Y*-, and *Z*-coordinates of a point located on the face.

featureName

A tuple of Floats specifying the name of the feature that created this face.

instanceName

A tuple of Floats specifying the name of the part instance for this face (if applicable).

FaceArray object

The FaceArray is a sequence of *Face* objects. If the part is modified, then FaceArray must be updated for that part.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].faces
mdb.models[name].parts[name].allInternalSurfaces[name].faces
mdb.models[name].parts[name].allSets[name].faces
mdb.models[name].parts[name].allSurfaces[name].faces
mdb.models[name].parts[name].faces
mdb.models[name].parts[name].sets[name].faces
mdb.models[name].parts[name].surfaces[name].faces
import assembly
mdb.models[name].rootAssembly.allInstances[name].faces
mdb.models[name].rootAssembly.allInstances[name].sets[name].faces
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].faces
mdb.models[name].rootAssembly.allInternalSets[name].faces
mdb.models[name].rootAssembly.allInternalSurfaces[name].faces
mdb.models[name].rootAssembly.allSets[name].faces
mdb.models[name].rootAssembly.allSurfaces[name].faces
mdb.models[name].rootAssembly.instances[name].faces
mdb.models[name].rootAssembly.instances[name].sets[name].faces
mdb.models[name].rootAssembly.instances[name].surfaces[name].faces
mdb.models[name].rootAssembly.modelInstances[i].sets[name].faces
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name].faces
mdb.models[name].rootAssembly.sets[name].faces
mdb.models[name].rootAssembly.surfaces[name].faces
```

FaceArray(...)

This method creates a FaceArray object.

Path

part.FaceArray

Required arguments

faces A list of *Face* objects.

Optional arguments

None.

Return value

A FaceArray object.

Exceptions

None.

findAt(...)

This method returns the object or objects in the FaceArray located at the given coordinates.

`findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any face that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities). The arbitrary point must not be shared by a second face. If two faces intersect or coincide at the arbitrary point, `findAt` chooses the first face that it encounters, and you should not rely on the return value being consistent.

`findAt` will always try to find objects among all the faces in the part or assembly instance and will not restrict itself to a subset even if the FaceArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either a Face object or a sequence of Face objects based on the type of input.

- If *coordinates* is a sequence of Floats, `findAt` returns the Face object at that point.
- If you omit the *coordinates* keyword argument, `findAt` accepts as arguments a sequence of pairs of sequences describing each face's coordinate and normal, and `findAt` returns a sequence of Face objects at the given locations. If you omit the *coordinates* keyword argument, you must also omit the *normal* argument.

```
faces = f.findAt(((-16.438578, -41.835673, -24.19804), ),
                 ((25.210364, -35.689868, 1.860314), ),
                 ((26.727683, -38.207055, 4.164759), ))
```

Optional arguments

normal A sequence of Floats specifying the *X*-, *Y*-, and *Z*-components of a vector indicating the face normal.

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

A Face object.

Exceptions

None.

getExteriorEdges()

This method returns the edges on the exterior of the faces in the FaceArray. That is, it returns the edges that are referenced by exactly one of the faces in the sequence.

Arguments

None.

Return value

An EdgeArray object specifying the exterior edges.

Exceptions

None.

getSequenceFromMask(...)

This method returns the object or objects in the FaceArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When a large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A Face object or a sequence of Face objects.

Exceptions

None.

getMask()

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

getByBoundingBox(...)

This method returns an array of face objects that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin	A float specifying the minimum *X*-boundary of the bounding box.
yMin	A float specifying the minimum *Y*-boundary of the bounding box.
zMin	A float specifying the minimum *Z*-boundary of the bounding box.
xMax	A float specifying the maximum *X*-boundary of the bounding box.
yMax	A float specifying the maximum *Y*-boundary of the bounding box.
zMax	A float specifying the maximum *Z*-boundary of the bounding box.

Return value

A FaceArray object, which is a sequence of *Face* objects.

Exceptions

None.

`getByBoundingCylinder(...)`

This method returns an array of face objects that lie within the specified bounding cylinder.

Required arguments

center1	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the first end of the cylinder.
center2	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the second end of the cylinder.
radius	A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

A FaceArray object, which is a sequence of *Face* objects.

Exceptions

None.

getByBoundingSphere(...)

This method returns an array of face objects that lie within the specified bounding sphere.

Required arguments

center A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the sphere.

radius A float specifying the radius of the sphere.

Optional arguments

None.

Return value

A FaceArray object, which is a sequence of *Face* objects.

Exceptions

None.

getBoundingBox()

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the face sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

high: a tuple of three floats representing the maximum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

Exceptions

None.

getClosest(...)

This method returns an object or objects in the FaceArray closest to the given set of points, where the given points need not lie on the faces in the FaceArray.

Required arguments

coordinates A sequence of a sequence of floats, where each sequence of floats describes the *X*-, *Y*-, and *Z*-coordinates of a point.

```
r=f.getClosest(coordinates=((20.0,20.0,10.0),(-1.0, -15.0, 15),))

r.keys()

[0, 1]

r[0]

(mdb.models['Model-1'].parts['Part-1'].faces[0],
                                    (15.7090625762939, 20.0, 10.0))
```

Optional arguments

searchTolerance A double specifying the distance within which the closest object must lie. The default value is half of the parent part/instance size.

Return value

This method returns a dictionary object. The key to the dictionary object is the position of the input point in the tuple specified in the *coordinates* starting at index 0. If a closest face could be found then the value is a sequence consisting of two objects. The first object in the sequence is a *Face* that is close to the input point referred to by the key. The second object in the sequence is a sequence of floats that specifies the *X*-, *Y*-, and *Z*-location of the closest point on the *Face* to the given point. See program listing above.

Exceptions

An exception occurs if the resulting sequence is empty. Error: The mask results in an empty sequence

Members

The FaceArray object has no members.

IgnoredEdge object

An IgnoredEdge object is a one-dimensional region of geometry that has been abstracted away by a virtual topology feature.

Access

```
import part
mdb.models[name].parts[name].ignoredEdges[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].ignoredEdges[i]
mdb.models[name].rootAssembly.instances[name].ignoredEdges[i]
```

`getSize(...)`

This method returns a Float indicating the length of the edge.

Required arguments

None.

Optional arguments

printResults A Bool specifying whether verbose output is printed. The default is True.

Return value

A Float.

Exceptions

None.

`getRadius()`

This method returns the radius of a circular IgnoredEdge object.

Arguments

None.

Return value

A Float specifying the radius.

Exceptions

The given IgnoredEdge object is not circular.

`getCurvature(...)`

This method returns curvature information at a location on the IgnoredEdge object.

Required arguments

parameter A Float specifying the normalized parameter location on the IgnoredEdge where the curvature is to be computed. This argument is mutually exclusive with the argument *point*.

point A tuple of *X*-, *Y*-, and *Z*-coordinates of a point at which the curvature is to be computed. If *point* does not lie on the IgnoredEdge an attempt is made to project it onto the IgnoredEdge and use the projected point.

Optional arguments

None.

Return value

A dictionary with keys 'evaluationPoint', 'curvature', 'radius', 'tangent'. Where 'evaluationPoint' specifies the location at which the curvature was computed. 'curvature' specifies the curvature vector at that location. 'radius' is the Radius of curvature and 'tangent' specifies the tangent to the IgnoredEdge at that location.

Exceptions

The given IgnoredEdge is straight.

Members

The IgnoredEdge object has the following members:

index An Int specifying the index of the IgnoredEdge in the *IgnoredEdgeArray*.

pointOn A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the edge.

IgnoredEdgeArray object

The IgnoredEdgeArray is a sequence of *IgnoredEdge* objects. If the part is modified, then IgnoredEdgeArray must be updated for that part.

Access

```
import part
mdb.models[name].parts[name].ignoredEdges
import assembly
mdb.models[name].rootAssembly.allInstances[name].ignoredEdges
mdb.models[name].rootAssembly.instances[name].ignoredEdges
```

`findAt(...)`

This method returns the object or objects in the IgnoredEdgeArray located at the given coordinates.

`findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any *IgnoredEdge* that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities). The arbitrary point must not be shared by a second *IgnoredEdge*. If two *IgnoredEdge* objects intersect or coincide at the arbitrary point, `findAt` chooses the first *IgnoredEdge* that it encounters, and you should not rely on the return value being consistent.

`findAt` will always try to find objects among all the ignored edges in the part or assembly instance and will not restrict itself to a subset even if the IgnoredEdgeArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either an *IgnoredEdge* object or a sequence of *IgnoredEdge* objects based on the type of input.

- If *coordinates* is a sequence of Floats, `findAt` returns the *IgnoredEdge* object at that point.
- If you omit the *coordinates* keyword argument, `findAt` accepts as arguments a sequence of sequence of floats in the following format:

```
ignoredEdges = e.findAt(((20.19686, -169.513997, 27.798593), ),
                        ((19.657627, -167.295749, 27.056402), ),
                        ((18.274129, -157.144741, 25.15218), ))
```

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

An *IgnoredEdge* object or a sequence of *IgnoredEdge* objects.

Exceptions

None.

`getSequenceFromMask(...)`

This method returns the object or objects in the IgnoredEdgeArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

An *IgnoredEdge* object or a sequence of *IgnoredEdge* objects.

Exceptions

None.

`getMask()`

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

`getClosest(...)`

This method returns a object or objects in the IgnoredEdgeArray closest to the given set of points, where the given points need not lie on the edges in the IgnoredEdgeArray.

Required arguments

coordinates A sequence of a sequence of floats, where each sequence of floats describes the *X*-, *Y*-, and *Z*-coordinates of a point.

```
r=e.getClosest(coordinates=((20.0,20.0,10.0),(-1.0, -15.0, 15),))

r.keys()

[0, 1]

r[0]

(mdb.models['Model-1'].parts['Part-1'].ignoredEdges[3],
                                    (15.7090625762939, 20.0, 10.0))
```

Optional arguments

searchTolerance A double specifying the distance within which the closest object must lie. The default value is half of the parent part/instance size.

Return value

This method returns a dictionary object. The key to the dictionary object is the position of the input point in the tuple specified in the *coordinates* starting at index 0. If a closest *IgnoredEdge* could be found then the value is a sequence consisting of two objects. The first object in the sequence is an *IgnoredEdge* that is close to the input point referred to by the key. The second object in the sequence, is a sequence of floats which specify the *X*-, *Y*-, and *Z*-location of the closest point on the *IgnoredEdge* to the given point. See program listing above.

Exceptions

An exception occurs if the resulting sequence is empty. Error: The mask results in an empty sequence

Members

The IgnoredEdgeArray object has no members.

IgnoredVertex object

An IgnoredVertex object is a point region of the geometry that was abstracted away by a virtual topology feature.

Access

```
import part
mdb.models[name].parts[name].ignoredVertices[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].ignoredVertices[i]
mdb.models[name].rootAssembly.instances[name].ignoredVertices[i]
```

Members

The IgnoredVertex object has the following members:

index An Int specifying the index of the IgnoredVertex in the IgnoredVertexArray.

pointOn A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the vertex.

IgnoredVertexArray object

The IgnoredVertexArray is a sequence of *IgnoredVertex* objects. If the part is modified, then IgnoredVertexArray must be updated for that part.

Access

```
import part
mdb.models[name].parts[name].ignoredVertices
import assembly
mdb.models[name].rootAssembly.allInstances[name].ignoredVertices
mdb.models[name].rootAssembly.instances[name].ignoredVertices
```

`findAt(...)`

This method returns the object or objects in the IgnoredVertexArray located at the given coordinates.

`findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any *IgnoredVertex* object that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities).

`findAt` will always try to find objects among all the ignored vertices in the part or assembly instance and will not restrict itself to a subset even if the IgnoredVertexArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either a *IgnoredVertex* object or a sequence of *IgnoredVertex* objects based on the type of input.

- If *coordinates* is a sequence of Floats, `findAt` returns the *IgnoredVertex* object at that point.
- If you omit the *coordinates* keyword argument, `findAt` accepts as arguments a sequence of sequence of floats in the following format:

```
verts = v.findAt(((20.19686, -169.513997, 27.798593), ),
                 ((19.657627, -167.295749, 27.056402), ),
                 ((18.274129, -157.144741, 25.15218), ))
```

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

An *IgnoredVertex* object or a sequence of *IgnoredVertex* objects.

Exceptions

None.

getSequenceFromMask(...)

This method returns the object or objects in the IgnoredVertexArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

An *IgnoredVertex* object or a sequence of *IgnoredVertex* objects.

Exceptions

None.

getMask()

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

getClosest(...)

This method returns a object or objects in the IgnoredVertexArray closest to the given set of points, where the given points need not lie on the vertices in the IgnoredVertexArray.

Required arguments

coordinates A sequence of a sequence of floats, where each sequence of floats describes the *X*-, *Y*-, and *Z*-coordinates of a point.

```
r=e.getClosest(coordinates=((20.0,20.0,10.0),(-1.0, -15.0, 15),))

r.keys()

[0, 1]

r[0]

(mdb.models['Model-1'].parts['Part-1'].ignoredVertices[3],
                                    (15.7090625762939, 20.0, 10.0))
```

Optional arguments

searchTolerance A double specifying the distance within which the closest object must lie. The default value is half of the parent part/instance size.

Return value

This method returns a dictionary object. The key to the dictionary object is the position of the input point in the tuple specified in the *coordinates* starting at index 0. If a closest *IgnoredVertex* could be found then the value is a sequence consisting of two objects. The first object in the sequence is a *IgnoredVertex* that is close to the input point referred to by the key. The second object in the sequence, is a sequence of floats which specify the *X*-, *Y*-, and *Z*-location of the *IgnoredVertex*. See program listing above.

Exceptions

An exception occurs if the resulting sequence is empty. Error: The mask results in an empty sequence

Members

The IgnoredVertexArray object has no members.

InterestingPoint object

Interesting points can be located at the following:

- The middle of an edge.
- The middle of an arc.
- The center of an arc.

An InterestingPoint object is a temporary object and cannot be accessed from the *Mdb* object.

Access

```
import part
import assembly
```

InterestingPoint(...)

This method creates an interesting point along an edge. An InterestingPoint is a temporary object.

Path

mdb.models[*name*].parts[*name*].InterestingPoint

mdb.models[*name*].rootAssembly.instances[*name*].InterestingPoint

Required arguments

edge	An Edge object specifying the edge on which the interesting point is positioned.
rule	A SymbolicConstant specifying the position of the interesting point. Possible values are MIDDLE or CENTER.

Optional arguments

None.

Return value

An InterestingPoint object.

Exceptions

None.

Members

The InterestingPoint object has no members.

ModelDot object

The ModelDot object can be used to access an actual *MeshNode*, *ReferencePoint*, or *Vertex* object.

Members

The ModelDot object has no members.

ReferencePoint object

The ReferencePoint object has no direct constructor; it is created when a *Feature* object is created. The `ReferencePoint` method creates a *Feature* object that creates a ReferencePoint object.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].referencePoints[i]
mdb.models[name].parts[name].allSets[name].referencePoints[i]
mdb.models[name].parts[name].referencePoints[i]
mdb.models[name].parts[name].sets[name].referencePoints[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].referencePoints[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name]\
.referencePoints[i]
mdb.models[name].rootAssembly.allInternalSets[name].referencePoints[i]
mdb.models[name].rootAssembly.allSets[name].referencePoints[i]
mdb.models[name].rootAssembly.instances[name].referencePoints[i]
mdb.models[name].rootAssembly.instances[name].sets[name]\
.referencePoints[i]
mdb.models[name].rootAssembly.modelInstances[i].referencePoints[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name]\
.referencePoints[i]
mdb.models[name].rootAssembly.referencePoints[i]
mdb.models[name].rootAssembly.sets[name].referencePoints[i]
```

Members

The ReferencePoint object has no members.

Transform object

The `MakeSketchTransform` method creates a Transform object. The Transform object has no direct constructor. A Transform object is a 4×3 matrix of Floats that represents the transformation from sketch coordinates to assembly coordinates or to part coordinates.

Access

```
import part
import assembly
```

`matrix()`

This method returns the transformation matrix as a tuple of 12 Floats.

Arguments

None.

Return value

A tuple of 12 Floats.

Exceptions

None.

Members

The Transform object has no members.

Vertex object

Vertices are point regions of geometry.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].vertices[i]
mdb.models[name].parts[name].allSets[name].vertices[i]
mdb.models[name].parts[name].sets[name].vertices[i]
mdb.models[name].parts[name].vertices[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].sets[name].vertices[i]
mdb.models[name].rootAssembly.allInstances[name].vertices[i]
mdb.models[name].rootAssembly.allInternalSets[name].vertices[i]
mdb.models[name].rootAssembly.allSets[name].vertices[i]
mdb.models[name].rootAssembly.instances[name].sets[name].vertices[i]
mdb.models[name].rootAssembly.instances[name].vertices[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].vertices[i]
mdb.models[name].rootAssembly.modelInstances[i].vertices[i]
mdb.models[name].rootAssembly.sets[name].vertices[i]
mdb.models[name].rootAssembly.vertices[i]
```

`getEdges()`

This method returns a sequence consisting of the edge ids of the edges which share this vertex.

Arguments

None.

Return value

A tuple of integers.

Exceptions

None.

`getNodes()`

This method returns an array of node objects that are associated with the vertex.

Arguments

None.

Return value

A *MeshNodeArray* object which is a sequence of *MeshNode* objects.

Exceptions

None.

`getElements()`

This method returns an array of element objects that are associated with the vertex.

Arguments

None.

Return value

A *MeshElementArray* object which is a sequence of *MeshElement* objects.

Exceptions

None.

Members

The Vertex object has the following members:

index	An Int specifying the index of the Vertex in the VertexArray.
isReferenceRep	A Boolean specifying whether the vertex belongs to the reference representation of the Part or Instance.
pointOn	A tuple of Floats specifying the *X* -, *Y* -, and *Z* -coordinates of the vertex.
featureName	A tuple of Floats specifying the name of the feature that created this vertex.
instanceName	A tuple of Floats specifying the name of the part instance for this vertex (if applicable).

VertexArray object

The VertexArray is a sequence of *Vertex* objects. If the part is modified, then VertexArray must be updated for that part.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].vertices
mdb.models[name].parts[name].allSets[name].vertices
mdb.models[name].parts[name].sets[name].vertices
mdb.models[name].parts[name].vertices
import assembly
mdb.models[name].rootAssembly.allInstances[name].sets[name].vertices
mdb.models[name].rootAssembly.allInstances[name].vertices
mdb.models[name].rootAssembly.allInternalSets[name].vertices
mdb.models[name].rootAssembly.allSets[name].vertices
mdb.models[name].rootAssembly.instances[name].sets[name].vertices
mdb.models[name].rootAssembly.instances[name].vertices
mdb.models[name].rootAssembly.modelInstances[i].sets[name].vertices
mdb.models[name].rootAssembly.modelInstances[i].vertices
mdb.models[name].rootAssembly.sets[name].vertices
mdb.models[name].rootAssembly.vertices
```

VertexArray(...)

This method creates a VertexArray object.

Path

part.VertexArray

Required arguments

vertices A list of *Vertex* objects.

Optional arguments

None.

Return value

A VertexArray object.

Exceptions

None.

findAt(...)

This method returns the object or objects in the VertexArray located at the given coordinates.

`findAt` initially uses the ACIS tolerance of 1E-6. As a result, `findAt` returns any *Vertex* object that is at the arbitrary point specified or at a distance of less than 1E-6 from the arbitrary point. If nothing is found, `findAt` uses the tolerance for imprecise geometry (applicable only for imprecise geometric entities).

`findAt` will always try to find objects among all the vertices in the part or assembly instance and will not restrict itself to a subset even if the VertexArray represents such subset.

Required arguments

coordinates A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the object to find.

`findAt` returns either a Vertex object or a sequence of Vertex objects based on the type of input.

- If *coordinates* is a sequence of Floats, `findAt` returns the Vertex object at that point.
- If you omit the *coordinates* keyword argument, `findAt` accepts as arguments a sequence of sequence of floats in the following format:

```
verts = v.findAt(((20.19686, -169.513997, 27.798593), ),
                 ((19.657627, -167.295749, 27.056402), ),
                 ((18.274129, -157.144741, 25.15218), ))
```

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

A Vertex object or a sequence of Vertex objects.

Exceptions

None.

`getSequenceFromMask(...)`

This method returns the object or objects in the VertexArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When a large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A *Vertex* object or a sequence of *Vertex* objects.

Exceptions

None.

getMask()

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

getByBoundingBox(...)

This method returns an array of vertex objects that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin A float specifying the minimum *X*-boundary of the bounding box.

yMin A float specifying the minimum *Y*-boundary of the bounding box.

zMin A float specifying the minimum *Z*-boundary of the bounding box.

xMax A float specifying the maximum *X*-boundary of the bounding box.

yMax A float specifying the maximum *Y*-boundary of the bounding box.

zMax A float specifying the maximum *Z*-boundary of the bounding box.

Return value

A VertexArray object, which is a sequence of *Vertex* objects.

Exceptions

None.

getByBoundingCylinder(...)

This method returns an array of vertex objects that lie within the specified bounding cylinder.

Required arguments

center1	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the first end of the cylinder.
center2	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the second end of the cylinder.
radius	A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

A VertexArray object, which is a sequence of *Vertex* objects.

Exceptions

None.

`getByBoundingSphere(...)`

This method returns an array of vertex objects that lie within the specified bounding sphere.

Required arguments

center	A tuple of the *X*-, *Y*-, and *Z*-coordinates of the center of the sphere.
radius	A float specifying the radius of the sphere.

Optional arguments

None.

Return value

A VertexArray object, which is a sequence of *Vertex* objects.

Exceptions

None.

`getBoundingBox()`

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the vertex sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

high: a tuple of three floats representing the maximum *X*-, *Y*-, and *Z*-boundary values of the bounding box.

Exceptions

None.

`getClosest(...)`

This method returns a object or objects in the VertexArray closest to the given set of points, where the given points need not lie on *Vertex* objects in the VertexArray.

Required arguments

coordinates A sequence of a sequence of floats, where each sequence of floats describes the *X*-, *Y*-, and *Z*-coordinates of a point.

```
r=v.getClosest(coordinates=((20.0,20.0,10.0),(-1.0, -15.0, 15),))
```

```
r.keys()
```

```
[0, 1]
```

```
r[0]
```

```
(mdb.models['Model-1'].parts['Part-1'].vertices[0],
        (15.7090625762939, 29.1666641235352, 20.0))
```

Optional arguments

searchTolerance A double specifying the distance within which the closest object must lie. The default value is half of the parent part/instance size.

Return value

This method returns a dictionary object. The key to the dictionary object is the position of the input point in the tuple specified in the *coordinates* starting at index 0. If a closest vertex could be found then the value is a sequence consisting of two objects. The first object in the sequence is a *Vertex* that is close to the input point referred to by the key. The second object in the sequence is a sequence of floats that specifies the *X*-, *Y*-, and *Z*-location of the *Vertex*. See program listing above.

Exceptions

An exception occurs if the resulting sequence is empty. Error: The mask results in an empty sequence

Members

The VertexArray object has no members.

Beam Section profile commands

The Beam Section profile commands are used to create profiles.
A profile definition specifies the properties of a beam that are related to its cross-section geometry. When you define a beam section, you must include a reference to a profile in the section definition.

In this section:

- *Profile object*
- *ArbitraryProfile object*
- *BoxProfile object*
- *ChannelProfile object*
- *CircularProfile object*
- *GeneralizedProfile object*
- *HatProfile object*
- *HexagonalProfile object*
- *IProfile object*
- *LProfile object*
- *PipeProfile object*
- *RectangularProfile object*
- *TProfile object*
- *TrapezoidalProfile object*

Profile object

The Profile object defines the geometrical properties of a beam cross-section. Profile is an abstract base type.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

beamProfilesFromOdb(...)

This method creates Profile objects by reading an output database. The new profiles are placed in the profiles repository.

Path

mdb.models[*name*].beamProfilesFromOdb

Required arguments

fileName A String specifying the name of the output database file (including the `.odb` extension) to be read. The String can also be the full path to the output database file if it is located in another directory.

Optional arguments

None.

Return value

A list of Profile objects.

Exceptions

None.

Members

The Profile object has the following member:

name A String specifying the repository key.

ArbitraryProfile object

The ArbitraryProfile object defines the properties of an arbitrary profile.

The ArbitraryProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

ArbitraryProfile(...)

This method creates a ArbitraryProfile object.

Path

```
mdb.models[name].ArbitraryProfile
session.odbs[name].ArbitraryProfile
```

Required arguments

name	A String specifying the repository key.
table	A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

The first sequence in the table specifies the following:

- 1-coordinate of the first point defining the profile.
- 2-coordinate of the first point defining the profile.

All other sequences in the table specify the following:

- 1–coordinate of the next point defining the profile.
- 2–coordinate of the next point defining the profile.
- The thickness of the segment ending at that point.

Return value

An ArbitraryProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ArbitraryProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ArbitraryProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ArbitraryProfile object has members with the same names and descriptions as the arguments to the *ArbitraryProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=ARBITRARY

BoxProfile object

The BoxProfile object defines the properties of a box profile.

The BoxProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

BoxProfile(...)

This method creates a BoxProfile object.

Path

```
mdb.models[name].BoxProfile
session.odbs[name].BoxProfile
```

Required arguments

name
A String specifying the repository key.

a
A Float specifying the *a* dimension of the box profile. For more information, see *Beam Cross-Section Library*.

b
A Float specifying the *b* dimension of the box profile.

uniformThickness
A Boolean specifying whether the thickness is uniform.

t1
A Float specifying the uniform wall thickness if *uniformThickness*=ON, and the wall thickness of the first segment if *uniformThickness*=OFF.

Optional arguments

t2
A Float specifying the wall thickness of the second segment. *t2* is required only when *uniformThickness*=OFF. The default value is 0.0.

t3
A Float specifying the wall thickness of the third segment. *t3* is required only when *uniformThickness*=OFF. The default value is 0.0.

t4
A Float specifying the wall thickness of the fourth segment. *t4* is required only when *uniformThickness*=OFF. The default value is 0.0.

Return value

A BoxProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the BoxProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BoxProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The BoxProfile object has members with the same names and descriptions as the arguments to the *BoxProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=BOX

ChannelProfile object

The ChannelProfile object defines the properties of a Channel profile.

The ChannelProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

ChannelProfile(...)

This method creates a ChannelProfile object.

Path

```
mdb.models[name].ChannelProfile
session.odbs[name].ChannelProfile
```

Required arguments

name A String specifying the repository key.

l A positive Float specifying the *l* dimension (offset of 1–axis from the bottom flange surface) of the Channel profile. For more information, see *Beam Cross-Section Library*.

h A positive Float specifying the *h* dimension (height) of the Channel profile.

b1 A positive Float specifying the *b1* dimension (lower flange width) of the Channel profile.

b2 A positive Float specifying the *b2* dimension (upper flange width) of the Channel profile.

t1 A positive Float specifying the *t1* dimension (lower flange thickness) of the Channel profile.

t2 A positive Float specifying the *t2* dimension (upper flange thickness) of the Channel profile.

t3 A positive Float specifying the *t3* dimension (web thickness) of the Channel profile.

o A positive Float specifying the *o* dimension (offset of 2–axis from the left edge of web) of the Channel profile.

Optional arguments

None.

Return value

A ChannelProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ChannelProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ChannelProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ChannelProfile object has members with the same names and descriptions as the arguments to the *ChannelProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=CHANNEL

CircularProfile object

The CircularProfile object defines the properties of a solid circular profile.

The CircularProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

CircularProfile(...)

This method creates a CircularProfile object.

Path

```
mdb.models[name].CircularProfile
session.odbs[name].CircularProfile
```

Required arguments

name A String specifying the repository key.

r A positive Float specifying the *r* dimension (outer radius) of the circular profile. For more information, see *Beam Cross-Section Library*.

Optional arguments

None.

Return value

A CircularProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CircularProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CircularProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The CircularProfile object has members with the same names and descriptions as the arguments to the *CircularProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=CIRC

GeneralizedProfile object

The GeneralizedProfile object defines the properties of a profile via its area, moment of inertia, etc.

The GeneralizedProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

GeneralizedProfile(...)

This method creates a GeneralizedProfile object.

Path

```
mdb.models[name].GeneralizedProfile
session.odbs[name].GeneralizedProfile
```

Required arguments

name
A String specifying the repository key.

area
A Float specifying the cross-sectional area for the profile.

i11
A Float specifying the moment of inertia for bending about the 1-axis, I_{11}.

i12
A Float specifying the moment of inertia for cross bending, I_{12}.

i22
A Float specifying the moment of inertia for bending about the 2-axis, I_{22}.

j
A Float specifying the torsional constant, J.

gammaO
A Float specifying the sectorial moment, Γ_0.

gammaW
A Float specifying the warping constant, Γ_W.

Optional arguments

None.

Return value

A GeneralizedProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the GeneralizedProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GeneralizedProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The GeneralizedProfile object has members with the same names and descriptions as the arguments to the *GeneralizedProfile* method.

Corresponding analysis keywords

- **BEAM GENERAL SECTION*, SECTION=GENERAL or NONLINEAR GENERAL

HatProfile object

The HatProfile object defines the properties of a Hat profile.

The HatProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

HatProfile(...)

This method creates a HatProfile object.

Path

```
mdb.models[name].HatProfile
session.odbs[name].HatProfile
```

Required arguments

name A String specifying the repository key.

l A positive Float specifying the *l* dimension (offset of 1–axis from the bottom flange surface) of the Hat profile. For more information, see *Beam Cross-Section Library*.

h A positive Float specifying the *h* dimension (height) of the Hat profile.

b A positive Float specifying the *b* dimension (bottom width) of the Hat profile.

b1 A positive Float specifying the *b1* dimension (upper flange width) of the Hat profile.

b2 A positive Float specifying the *b2* dimension (lower flange width) of the Hat profile.

t1 A positive Float specifying the *t1* dimension (upper flange thickness) of the Hat profile.

t2 A positive Float specifying the *t2* dimension (lower flange thickness) of the Hat profile.

t3 A positive Float specifying the *t3* dimension (web thickness) of the Hat profile.

Optional arguments

None.

Return value

A HatProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the HatProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HatProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The HatProfile object has members with the same names and descriptions as the arguments to the *HatProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=HAT

HexagonalProfile object

The HexagonalProfile object defines the properties of a hexagonal profile.

The HexagonalProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

HexagonalProfile(...)

This method creates a HexagonalProfile object.

Path

```
mdb.models[name].HexagonalProfile
session.odbs[name].HexagonalProfile
```

Required arguments

name A String specifying the repository key.

r A positive Float specifying the *r* dimension (outer radius) of the hexagonal profile. For more information, see *Beam Cross-Section Library*.

t A positive Float specifying the *t* dimension (wall thickness) of the hexagonal profile, $t < (sqrt(3)/2)r$.

Optional arguments

None.

Return value

A HexagonalProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the HexagonalProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HexagonalProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The HexagonalProfile object has members with the same names and descriptions as the arguments to the *HexagonalProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=HEX

IProfile object

The IProfile object defines the properties of an I profile.

The IProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

IProfile(...)

This method creates an IProfile object.

Path

```
mdb.models[name].IProfile
session.odbs[name].IProfile
```

Required arguments

name A String specifying the repository key.

l A Float specifying the *l* dimension (offset of 1–axis from the bottom flange surface) of the I profile. For more information, see *Beam Cross-Section Library*.

h A Float specifying the *h* dimension (height) of the I profile.

b1 A Float specifying the *b1* dimension (bottom flange width) of the I profile.

b2 A Float specifying the *b2* dimension (top flange width) of the I profile.

t1 A Float specifying the *t1* dimension (bottom flange thickness) of the I profile.

t2 A Float specifying the *t2* dimension (top flange thickness) of the I profile.

t3 A Float specifying the *t3* dimension (web thickness) of the I profile.

Optional arguments

None.

Return value

An IProfile object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the IProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The IProfile object has members with the same names and descriptions as the arguments to the *IProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=I

LProfile object

The LProfile object defines the properties of a L profile.

The LProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

LProfile(...)

This method creates a LProfile object.

Path

```
mdb.models[name].LProfile
session.odbs[name].LProfile
```

Required arguments

name A String specifying the repository key.

a A positive Float specifying the *a* dimension (flange length) of the L profile. For more information, see *Beam Cross-Section Library*.

b A positive Float specifying the *b* dimension (flange length) of the L profile.

t1 A positive Float specifying the *t1* dimension (flange thickness) of the L profile ($t1 < b$).

t2 A positive Float specifying the *t2* dimension (flange thickness) of the L profile ($t2 < a$).

Optional arguments

None.

Return value

A LProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the LProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The LProfile object has members with the same names and descriptions as the arguments to the *LProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=L

PipeProfile object

The PipeProfile object defines the properties of a circular pipe profile.

The PipeProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

PipeProfile(...)

This method creates a PipeProfile object.

Path

```
mdb.models[name].PipeProfile
session.odbs[name].PipeProfile
```

Required arguments

name A String specifying the repository key.

r A Float specifying the outer radius of the pipe. For more information, see *Beam Cross-Section Library*.

t A Float specifying the wall thickness of the pipe.

Optional arguments

formulation A SymbolicConstant specifying the wall thickness of the pipe profile. Possible values are THICK_WALL and THIN_WALL. The default value is THIN_WALL.

Return value

A PipeProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the PipeProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PipeProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PipeProfile object has members with the same names and descriptions as the arguments to the *PipeProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=PIPE
- **BEAM SECTION*, SECTION=THICK PIPE

RectangularProfile object

The RectangularProfile object defines the properties of a solid rectangular profile.

The RectangularProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

RectangularProfile(...)

This method creates a RectangularProfile object.

Path

```
mdb.models[name].RectangularProfile
session.odbs[name].RectangularProfile
```

Required arguments

name A String specifying the repository key.

a A positive Float specifying the *a* dimension of the rectangular profile. For more information, see *Beam Cross-Section Library*.

b A positive Float specifying the *b* dimension of the rectangular profile.

Optional arguments

None.

Return value

A RectangularProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the RectangularProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RectangularProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The RectangularProfile object has members with the same names and descriptions as the arguments to the *RectangularProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=RECT

TProfile object

The TProfile object defines the properties of a T profile.

The TProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

TProfile(...)

This method creates a TProfile object.

Path

```
mdb.models[name].TProfile
session.odbs[name].TProfile
```

Required arguments

name
A String specifying the repository key.

b
A positive Float specifying the *b* dimension (flange width) of the T profile. For more information, see *Beam Cross-Section Library*.

h
A positive Float specifying the *h* dimension (height) of the T profile.

l
A positive Float specifying the *l* dimension (offset of 1–axis from the edge of web) of the T profile.

tf
A positive Float specifying the *tf* dimension (flange thickness) of the T profile ($tf < h$).

tw
A positive Float specifying the *tw* dimension (web thickness) of the T profile ($tw < b$).

Optional arguments

None.

Return value

A TProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the TProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TProfile object has members with the same names and descriptions as the arguments to the *TProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=I

TrapezoidalProfile object

The TrapezoidalProfile object defines the properties of a trapezoidal profile.

The TrapezoidalProfile object is derived from the *Profile* object.

Access

```
import section
mdb.models[name].profiles[name]
import odbSection
session.odbs[name].profiles[name]
```

TrapezoidalProfile(...)

This method creates a TrapezoidalProfile object.

Path

```
mdb.models[name].TrapezoidalProfile
session.odbs[name].TrapezoidalProfile
```

Required arguments

name A String specifying the repository key.

a A positive Float specifying the *a* dimension of the Trapezoidal profile. For more information, see *Beam Cross-Section Library*.

b A positive Float specifying the *b* dimension of the Trapezoidal profile.

c A positive Float specifying the *c* dimension of the Trapezoidal profile.

d A Float specifying the *d* dimension of the Trapezoidal profile.

Optional arguments

None.

Return value

A TrapezoidalProfile object.

Exceptions

RangeError.

setValues(...)

This method modifies the TrapezoidalProfile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TrapezoidalProfile* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TrapezoidalProfile object has members with the same names and descriptions as the arguments to the *TrapezoidalProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=TRAPEZOID

Boundary Condition commands

A specific type of boundary condition object and a specific type of boundary condition state object are designed for each type of boundary condition. A BoundaryCondition object stores the non-propagating data of a boundary condition as well as a number of instances of the corresponding BoundaryConditionState object, each of which stores the propagating data of the boundary condition in a single step.
Instances of the BoundaryConditionState object are created and deleted internally by its corresponding BoundaryCondition object.

In this section:

- *BoundaryCondition object*
- *AccelerationBaseMotionBC object*
- *AccelerationBaseMotionBCState object*
- *AccelerationBC object*
- *AccelerationBCState object*
- *AcousticPressureBC object*
- *AcousticPressureBCState object*
- *BoundaryConditionState object*
- *ConcentrationBC object*
- *ConcentrationBCState object*
- *ConnAccelerationBC object*
- *ConnAccelerationBCState object*
- *ConnDisplacementBC object*
- *ConnDisplacementBCState object*
- *ConnVelocityBC object*
- *ConnVelocityBCState object*
- *DisplacementBaseMotionBC object*
- *DisplacementBaseMotionBCState object*
- *DisplacementBC object*
- *DisplacementBCState object*
- *ElectricPotentialBC object*
- *ElectricPotentialBCState object*
- *ElectricPotentialElectrolyteBC object*
- *ElectricPotentialElectrolyteBCState object*
- *EulerianBC object*
- *EulerianBCState object*
- *EulerianMotionBC object*
- *EulerianMotionBCState object*
- *FluidCavityPressureBC object*
- *FluidCavityPressureBCState object*
- *IonConcentrationBC object*
- *IonConcentrationBCState object*
- *MagneticVectorPotentialBC object*
- *MaterialFlowBC object*

- *MaterialFlowBCState object*
- *PorePressureBC object*
- *PorePressureBCState object*
- *RetainedNodalDofsBC object*
- *SecondaryBaseBC object*
- *SecondaryBaseBCState object*
- *SubmodelBC object*
- *SubmodelBCState object*
- *TemperatureBC object*
- *TemperatureBCState object*
- *TypeBC object*
- *TypeBCState object*
- *VelocityBaseMotionBC object*
- *VelocityBaseMotionBCState object*
- *VelocityBC object*
- *VelocityBCState object*

BoundaryCondition object

The BoundaryCondition object is the abstract base type for other BoundaryCondition objects. The BoundaryCondition object has no explicit constructor. The methods and members of the BoundaryCondition object are common to all objects derived from the BoundaryCondition.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

deactivate(...)

This method deactivates the boundary condition in the specified step and all subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is deactivated.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

move(...)

This method moves the boundary condition state from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the boundary condition state is moved.

toStepName A String specifying the name of the step to which the boundary condition state is moved.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

reset(...)

This method resets the boundary condition state of the specified step to the state of the previous analysis step.

Required arguments

stepName	A String specifying the name of the step in which the boundary condition state is reset.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

resume()

This method resumes the boundary condition that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

suppress()

This method suppresses the boundary condition.

Arguments

None.

Return value

`None.`

Exceptions

None.

delete(...)

This method allows you to delete existing boundary conditions.

Required arguments

indices A sequence of Ints specifying the index of each boundary condition to delete.

Optional arguments

None.

Return value

`None`.

Exceptions

None.

Members

The BoundaryCondition object can have the following members:

name A String specifying the boundary condition repository key.

category A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region A *Region* object specifying the region to which the boundary condition is applied.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

AccelerationBaseMotionBC object

The AccelerationBaseMotionBC object stores the data for an acceleration base motion boundary condition.

The AccelerationBaseMotionBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

AccelerationBaseMotionBC(...)

This method creates a AccelerationBaseMotionBC object.

Path

```
mdb.models[name].AccelerationBaseMotionBC
```

Required arguments

name
A String specifying the boundary condition repository key.

createStepName
A String specifying the name of the step in which the boundary condition is created.

dof
A SymbolicConstant specifying the constrained degree-of-freedom. Possible values for the SymbolicConstant are U1, U2, U3, UR1, UR2, UR3. The default value is U1.

Optional arguments

amplitudeScaleFactor
A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

centerOfRotation
A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation
A *CorrelationArray* object.

secondaryBase
A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

useComplex
A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

An AccelerationBaseMotionBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing AccelerationBaseMotionBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AccelerationBaseMotionBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing AccelerationBaseMotionBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The AccelerationBaseMotionBC object can have the following members:

name

A String specifying the boundary condition repository key.

amplitudeScaleFactor

A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

useComplex

A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

centerOfRotation

A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation

A *CorrelationArray* object.

secondaryBase

A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

AccelerationBaseMotionBCState object

The AccelerationBaseMotionBCState object stores the propagating data for a velocity base motion boundary condition in a step. One instance of this object is created internally by the *AccelerationBaseMotionBC* object for each step. The instance is also deleted internally by the *AccelerationBaseMotionBC* object.

The AccelerationBaseMotionBCState object has no constructor or methods.

The AccelerationBaseMotionBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The AccelerationBaseMotionBCState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BASE MOTION*

AccelerationBC object

The AccelerationBC object stores the data for an acceleration boundary condition.

The AccelerationBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

AccelerationBC(...)

This method creates an AccelerationBC object.

Path

```
mdb.models[name].AccelerationBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

a1 A Float or a SymbolicConstant specifying the acceleration component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *a1*, *a2*, *a3*, *ar1*, *ar2*, and *ar3* are optional arguments, at least one of them must be specified.

a2 A Float or a SymbolicConstant specifying the acceleration component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

a3 A Float or a SymbolicConstant specifying the acceleration component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar1 A Float or a SymbolicConstant specifying the rotational acceleration component about the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar2
A Float or a SymbolicConstant specifying the rotational acceleration component about the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar3
A Float or a SymbolicConstant specifying the rotational acceleration component about the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

Return value

An AccelerationBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing AccelerationBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AccelerationBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing AccelerationBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

a1 A Float or a SymbolicConstant specifying the acceleration component in the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

a2 A Float or a SymbolicConstant specifying the acceleration component in the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

a3 A Float or a SymbolicConstant specifying the acceleration component in the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar1 A Float or a SymbolicConstant specifying the rotational acceleration component about the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar2 A Float or a SymbolicConstant specifying the rotational acceleration component about the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar3 A Float or a SymbolicConstant specifying the rotational acceleration component about the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The AccelerationBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

AccelerationBCState object

The AccelerationBCState object stores the propagating data of an acceleration boundary condition in a step. One instance of this object is created internally by the *AccelerationBC* object for each step. The instance is also deleted internally by the *AccelerationBC* object.

The AccelerationBCState object has no constructor or methods.

The AccelerationBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The AccelerationBCState object has the following members:

a1

A Float specifying the acceleration component in the 1-direction.

a2

A Float specifying the acceleration component in the 2-direction.

a3

A Float specifying the acceleration component in the 3-direction.

ar1

A Float specifying the rotational acceleration component about the 1-direction.

ar2

A Float specifying the rotational acceleration component about the 2-direction.

ar3

A Float specifying the rotational acceleration component about the 3-direction.

a1State

A SymbolicConstant specifying the propagation state of the acceleration component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

a2State

A SymbolicConstant specifying the propagation state of the acceleration component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

a3State

A SymbolicConstant specifying the propagation state of the acceleration component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar1State

A SymbolicConstant specifying the propagation state of the rotational acceleration component about the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar2State

A SymbolicConstant specifying the propagation state of the rotational acceleration component about the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar3State

A SymbolicConstant specifying the propagation state of the rotational acceleration component about the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY*, TYPE=ACCELERATION (degree of freedom: 1, 2, 3, 4, 5, or 6)

AcousticPressureBC object

The AcousticPressureBC object stores the data for an acoustic pressure boundary condition.

The AcousticPressureBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

AcousticPressureBC(...)

This method creates a AcousticPressureBC object.

Path

```
mdb.models[name].AcousticPressureBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the acoustic pressure magnitude. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

An AcousticPressureBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing AcousticPressureBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticPressureBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing AcousticPressureBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the acoustic pressure magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The AcousticPressureBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

AcousticPressureBCState object

The AcousticPressureBCState object stores the propagating data for an acoustic pressure boundary condition in a step. One instance of this object is created internally by the *AcousticPressureBC* object for each step. The instance is also deleted internally by the *AcousticPressureBC* object.

The AcousticPressureBCState object has no constructor or methods.

The AcousticPressureBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The AcousticPressureBCState object has the following members:

magnitude

A Float specifying the acoustic pressure magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the acoustic pressure magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 8)

BoundaryConditionState object

The BoundaryConditionState object is the abstract base type for other BoundaryConditionState objects. The BoundaryConditionState object has no explicit constructor or methods. The members of the BoundaryConditionState object are common to all objects derived from the BoundaryConditionState object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The BoundaryConditionState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the BoundaryConditionState object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

ConcentrationBC object

The ConcentrationBC object stores the data for a concentration boundary condition.

The ConcentrationBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ConcentrationBC(...)

This method creates a ConcentrationBC object.

Path

```
mdb.models[name].ConcentrationBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the concentration magnitude. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

A ConcentrationBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConcentrationBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcentrationBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing ConcentrationBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the concentration magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcentrationBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ConcentrationBCState object

The ConcentrationBCState object stores the propagating data for a concentration boundary condition in a step. One instance of this object is created internally by the *ConcentrationBC* object for each step. The instance is also deleted internally by the *ConcentrationBC* object.

The ConcentrationBCState object has no constructor or methods.

The ConcentrationBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ConcentrationBCState object has the following members:

magnitude

A Float specifying the concentration magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the concentration magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 11)

ConnAccelerationBC object

The ConnAccelerationBC object stores the data for a connector acceleration boundary condition.

The ConnAccelerationBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ConnAccelerationBC(...)

This method creates an ConnAccelerationBC object on a wire region. Alternatively, the boundary condition may also be applied to a wire set referenced from an assembled fastener template model.

Path

```
mdb.models[name].ConnAccelerationBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

Optional arguments

region The wire region to which the boundary condition is applied. This argument is not valid when *fastenerName* and *fastenerSetName* are specified.

fastenerName A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

a1 A Float or a SymbolicConstant specifying the acceleration component in the connector's local 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *a1*, *a2*, *a3*, *ar1*, *ar2*, and *ar3* are optional arguments, at least one of them must be specified.

a2
A Float or a SymbolicConstant specifying the acceleration component in the connector's local 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

a3
A Float or a SymbolicConstant specifying the acceleration component in the connector's local 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar1
A Float or a SymbolicConstant specifying the rotational acceleration component in the connector's local 4-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar2
A Float or a SymbolicConstant specifying the rotational acceleration component in the connector's local 5-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ar3
A Float or a SymbolicConstant specifying the rotational acceleration component in the connector's local 6-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

Return value

A ConnAccelerationBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConnAccelerationBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnAccelerationBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConnAccelerationBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

a1 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

a2 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

a3 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar1 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 4-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar2 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 5-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ar3 A Float or a SymbolicConstant specifying the connector acceleration component in the connector's local 6-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConnAccelerationBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

fastenerName

A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName

A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ConnAccelerationBCState object

The ConnAccelerationBCState object stores the propagating data of a connector acceleration boundary condition in a step. One instance of this object is created internally by the *ConnAccelerationBC* object for each step. The instance is also deleted internally by the *ConnAccelerationBC* object.

The ConnAccelerationBCState object has no constructor or methods.

The ConnAccelerationBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ConnAccelerationBCState object has the following members:

a1

A Float specifying the connector acceleration component in the connector's local 1-direction.

a2

A Float specifying the connector acceleration component in the connector's local 2-direction.

a3

A Float specifying the connector acceleration component in the connector's local 3-direction.

ar1

A Float specifying the connector acceleration component in the connector's local 4-direction.

ar2

A Float specifying the connector acceleration component in the connector's local 5-direction.

ar3

A Float specifying the connector acceleration component in the connector's local 6-direction.

a1State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

a2State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

a3State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar1State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 4-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar2State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 5-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ar3State

A SymbolicConstant specifying the propagation state of the connector acceleration component in the connector's local 6-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **CONNECTOR MOTION*, TYPE=ACCELERATION (degree of freedom: 1, 2, 3, 4, 5, or 6)

ConnDisplacementBC object

The ConnDisplacementBC object stores the data for a connector displacement/rotation boundary condition.

The ConnDisplacementBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ConnDisplacementBC(...)

This method creates a ConnDisplacementBC object on a wire region. Alternatively, the boundary condition may also be applied to a wire set referenced from an assembled fastener template model.

Path

```
mdb.models[name].ConnDisplacementBC
```

Required arguments

name
A String specifying the boundary condition repository key.

createStepName
A String specifying the name of the step in which the boundary condition is created.

Optional arguments

region
The wire region to which the boundary condition is applied. This argument is not valid when *fastenerName* and *fastenerSetName* are specified.

fastenerName
A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName
A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

u1
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *u1*, *u2*, *u3*, *ur1*, *ur2*, and *ur3* are optional arguments, at least one of them must be specified.

u2
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

u3
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur1
A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 4-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur2
A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 5-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur3
A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 6-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

fixed
A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

buckleCase
A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

Return value

A ConnDisplacementBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConnDisplacementBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnDisplacementBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing ConnDisplacementBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

u1 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u2 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u3 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the connector's local 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur1 A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 4-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur2 A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 5-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur3 A Float, a Complex, or a SymbolicConstant specifying the rotational component in the connector's local 6-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

Return value
`None`.

Exceptions
None.

Members
The ConnDisplacementBC object can have the following members:

name
A String specifying the boundary condition repository key.

fixed
A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

buckleCase
A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

fastenerName
A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName
A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

category
A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region
A *Region* object specifying the region to which the boundary condition is applied.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ConnDisplacementBCState object

The ConnDisplacementBCState object stores the propagating data for a connector displacement/rotation boundary condition in a step. One instance of this object is created internally by the *ConnDisplacementBC* object for each step. The instance is also deleted internally by the *ConnDisplacementBC* object.

The ConnDisplacementBCState object has no constructor or methods.

The ConnDisplacementBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ConnDisplacementBCState object has the following members:

u1

A Float or a Complex specifying the displacement component in the connector's local 1-direction.

u2

A Float or a Complex specifying the displacement component in the connector's local 2-direction.

u3

A Float or a Complex specifying the displacement component in the connector's local 3-direction.

ur1

A Float or a Complex specifying the rotational component in the connector's local 4-direction.

ur2

A Float or a Complex specifying the rotational component in the connector's local 5-direction.

ur3

A Float or a Complex specifying the rotational component in the connector's local 6-direction.

u1State

A SymbolicConstant specifying the propagation state of the displacement component in the connector's local 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u2State

A SymbolicConstant specifying the propagation state of the displacement component in the connector's local 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u3State

A SymbolicConstant specifying the propagation state of the displacement component in the connector's local 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur1State

A SymbolicConstant specifying the propagation state of the rotational component in the connector's local 4-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur2State

A SymbolicConstant specifying the propagation state of the rotational component in the connector's local 5-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur3State

A SymbolicConstant specifying the propagation state of the rotational component in the connector's local 6-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **CONNECTOR MOTION*, TYPE=DISPLACEMENT (degree of freedom: 1, 2, 3, 4, 5, or 6)

ConnVelocityBC object

The ConnVelocityBC object stores the data for a connector velocity boundary condition.

The ConnVelocityBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ConnVelocityBC(...)

This method creates a ConnVelocityBC object on a wire region. Alternatively, the boundary condition may also be applied to a wire set referenced from an assembled fastener template model.

Path

```
mdb.models[name].ConnVelocityBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

Optional arguments

region The wire region to which the boundary condition is applied. This argument is not valid when *fastenerName* and *fastenerSetName* are specified.

fastenerName A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

v1 A Float or a SymbolicConstant specifying the velocity component in the connector's local 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *v1*, *v2*, *v3*, *vr1*, *vr2*, and *vr3* are optional arguments, at least one of them must be specified.

v2 A Float or a SymbolicConstant specifying the velocity component in the connector's local 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

v3
A Float or a SymbolicConstant specifying the velocity component in the connector's local 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr1
A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 4-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr2
A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 5-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr3
A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 6-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

Return value

A ConnVelocityBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConnVelocityBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnVelocityBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConnVelocityBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

v1 A Float or a SymbolicConstant specifying the velocity component in the connector's local 1-direction. Possible values for the SymbolicConstant are SET and FREED.

v2 A Float or a SymbolicConstant specifying the velocity component in the connector's local 2-direction. Possible values for the SymbolicConstant are SET and FREED.

v3 A Float or a SymbolicConstant specifying the velocity component in the connector's local 3-direction. Possible values for the SymbolicConstant are SET and FREED.

vr1 A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 4-direction. Possible values for the SymbolicConstant are SET and FREED.

vr2 A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 5-direction. Possible values for the SymbolicConstant are SET and FREED.

vr3 A Float or a SymbolicConstant specifying the rotational velocity component in the connector's local 6-direction. Possible values for the SymbolicConstant are SET and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConnVelocityBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

fastenerName

A String specifying the name of the assembled fastener to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName

A String specifying the assembled fastener template model set to which the boundary condition will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ConnVelocityBCState object

The ConnVelocityBCState object stores the propagating data for a velocity boundary condition in a step. One instance of this object is created internally by the *ConnVelocityBC* object for each step. The instance is also deleted internally by the *ConnVelocityBC* object.

The ConnVelocityBCState object has no constructor or methods.

The ConnVelocityBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ConnVelocityBCState object has the following members:

v1

A Float specifying the velocity component in the connector's local 1-direction.

v2

A Float specifying the velocity component in the connector's local 2-direction.

v3

A Float specifying the velocity component in the connector's local 3-direction.

vr1

A Float specifying the rotational velocity component in the connector's local 4-direction.

vr2

A Float specifying the rotational velocity component in the connector's local 5-direction.

vr3

A Float specifying the rotational velocity component in the connector's local 6-direction.

v1State

A SymbolicConstant specifying the propagation state of the velocity component in the connector's local 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v2State

A SymbolicConstant specifying the propagation state of the velocity component in the connector's local 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v3State

A SymbolicConstant specifying the propagation state of the velocity component in the connector's local 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr1State

A SymbolicConstant specifying the propagation state of the rotational velocity component in the connector's local 4–direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr2State

A SymbolicConstant specifying the propagation state of the rotational velocity component in the connector's local 5–direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr3State

A SymbolicConstant specifying the propagation state of the rotational velocity component in the connector's local 6–direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **CONNECTOR MOTION*, TYPE=VELOCITY (degree of freedom: 1, 2, 3, 4, 5, or 6)

DisplacementBaseMotionBC object

The DisplacementBaseMotionBC object stores the data for a displacement base motion boundary condition.

The DisplacementBaseMotionBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

DisplacementBaseMotionBC(...)

This method creates a DisplacementBaseMotionBC object.

Path

```
mdb.models[name].DisplacementBaseMotionBC
```

Required arguments

name
A String specifying the boundary condition repository key.

createStepName
A String specifying the name of the step in which the boundary condition is created.

dof
A SymbolicConstant specifying the constrained degree-of-freedom. Possible values for the SymbolicConstant are U1, U2, U3, UR1, UR2, UR3. The default value is U1.

Optional arguments

amplitudeScaleFactor
A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

centerOfRotation
A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation
A *CorrelationArray* object.

secondaryBase
A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

useComplex
A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A DisplacementBaseMotionBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing DisplacementBaseMotionBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DisplacementBaseMotionBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing DisplacementBaseMotionBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The DisplacementBaseMotionBC object can have the following members:

name

A String specifying the boundary condition repository key.

amplitudeScaleFactor

A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

useComplex

A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

centerOfRotation

A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation

A *CorrelationArray* object.

secondaryBase

A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

DisplacementBaseMotionBCState object

The DisplacementBaseMotionBCState object stores the propagating data for a velocity base motion boundary condition in a step. One instance of this object is created internally by the *DisplacementBaseMotionBC* object for each step. The instance is also deleted internally by the *DisplacementBaseMotionBC* object.

The DisplacementBaseMotionBCState object has no constructor or methods.

The DisplacementBaseMotionBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The DisplacementBaseMotionBCState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BASE MOTION*

DisplacementBC object

The DisplacementBC object stores the data for a displacement/rotation boundary condition.

The DisplacementBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

DisplacementBC(...)

This method creates a DisplacementBC object.

Path

```
mdb.models[name].DisplacementBC
```

Required arguments

name
A String specifying the boundary condition repository key.

createStepName
A String specifying the name of the step in which the boundary condition is created.

region
A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName
A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

u1
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *u1*, *u2*, *u3*, *ur1*, *ur2*, and *ur3* are optional arguments, at least one of them must be specified.

u2
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

u3
A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur1
A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur2
A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

ur3
A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

fixed
A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

buckleCase
A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

Return value

A DisplacementBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing DisplacementBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DisplacementBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing DisplacementBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

u1 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u2 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

u3 A Float, a Complex, or a SymbolicConstant specifying the displacement component in the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur1 A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 1-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur2 A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 2-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

ur3 A Float, a Complex, or a SymbolicConstant specifying the rotational displacement component about the 3-direction. Possible values for the SymbolicConstant are SET, UNCHANGED, and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

Return value

`None.`

Exceptions

None.

Members

The DisplacementBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

fixed

A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

buckleCase

A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

fieldName

A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

DisplacementBCState object

The DisplacementBCState object stores the propagating data for a displacement/rotation boundary condition in a step. One instance of this object is created internally by the *DisplacementBC* object for each step. The instance is also deleted internally by the *DisplacementBC* object.

The DisplacementBCState object has no constructor or methods.

The DisplacementBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The DisplacementBCState object has the following members:

u1
A Float or a Complex specifying the displacement component in the 1-direction.

u2
A Float or a Complex specifying the displacement component in the 2-direction.

u3
A Float or a Complex specifying the displacement component in the 3-direction.

ur1
A Float or a Complex specifying the rotational displacement component about the 1-direction.

ur2
A Float or a Complex specifying the rotational displacement component about the 2-direction.

ur3
A Float or a Complex specifying the rotational displacement component about the 3-direction.

u1State
A SymbolicConstant specifying the propagation state of the displacement component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u2State
A SymbolicConstant specifying the propagation state of the displacement component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

u3State
A SymbolicConstant specifying the propagation state of the displacement component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur1State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur2State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

ur3State
A SymbolicConstant specifying the propagation state of the rotational displacement component about the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY*, TYPE=DISPLACEMENT (degree of freedom: 1, 2, 3, 4, 5, or 6)

ElectricPotentialBC object

The ElectricPotentialBC object stores the data for an electrical potential boundary condition.

The ElectricPotentialBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ElectricPotentialBC(...)

This method creates an ElectricPotentialBC object.

Path

```
mdb.models[name].ElectricPotentialBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the electrical potential magnitude. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

An ElectricPotentialBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ElectricPotentialBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ElectricPotentialBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ElectricPotentialBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the electrical potential magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ElectricPotentialBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ElectricPotentialBCState object

The ElectricPotentialBCState object stores the propagating data for a electrical potential boundary condition in a step. One instance of this object is created internally by the *ElectricPotentialBC* object for each step. The instance is also deleted internally by the *ElectricPotentialBC* object.

The ElectricPotentialBCState object has no constructor or methods.

The ElectricPotentialBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ElectricPotentialBCState object has the following members:

magnitude

A Float specifying the electrical potential magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the electrical potential magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 9)

ElectricPotentialElectrolyteBC object

The ElectricPotentialElectrolyteBC object stores the data for an electrical potential in electrolyte boundary condition.

The ElectricPotentialElectrolyteBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

ElectricPotentialElectrolyteBC(...)

This method creates an ElectricPotentialElectrolyteBC object.

Path

```
mdb.models[name].ElectricPotentialElectrolyteBC
```

Required arguments

name
A String specifying the boundary condition repository key.

createStepName
A String specifying the name of the step in which the boundary condition is created.

region
A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName
A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude
A Float specifying the magnitude of the electrical potential of the electrolyte. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed
A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

An ElectricPotentialElectrolyteBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ElectricPotentialElectrolyteBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ElectricPotentialElectrolyteBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ElectricPotentialElectrolyteBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the electrical potential magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ElectricPotentialElectrolyteBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL, ELECTROCHEMICAL, and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

ElectricPotentialElectrolyteBCState object

The ElectricPotentialElectrolyteBCState object stores the propagating data for an electrical potential of electrolyte boundary condition in a step.
One instance of this object is created internally by the *ElectricPotentialElectrolyteBC* object for each step. The instance is also deleted internally by the *ElectricPotentialElectrolyteBC* object.

The ElectricPotentialElectrolyteBCState object has no constructor or methods.

The ElectricPotentialElectrolyteBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The ElectricPotentialElectrolyteBCState object has the following members:

magnitude

A Float specifying the electrical potential of electrolyte magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the electrical potential of electrolyte magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 9)

EulerianBC object

The EulerianBC object stores the data for an Eulerian boundary condition.

The EulerianBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

EulerianBC(...)

This method creates a EulerianBC object.

Path

```
mdb.models[name].EulerianBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

definition A SymbolicConstant specifying the flow conditions to be defined. Possible values are INFLOW, OUTFLOW, and BOTH. The default value is INFLOW.

inflowType A SymbolicConstant specifying the control of material flow into the Eulerian domain. Possible values are FREE, NONE, and VOID. The default value is FREE.

outflowType A SymbolicConstant specifying the control of flow of material out of the Eulerian domain. Possible values are ZERO_PRESSURE, FREE, NON_REFLECTING, and EQUILIBRIUM. The default value is ZERO_PRESSURE.

Return value

An EulerianBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing EulerianBC object in the step where it is created.

Required arguments

None.

Optional arguments

region A *Region* object specifying the region to which the boundary condition is applied.

definition A SymbolicConstant specifying the material flow conditions to be defined. Possible values are INFLOW, OUTFLOW, and BOTH. The default value is INFLOW.

inflowType A SymbolicConstant specifying the control of material flow into the Eulerian domain. Possible values are FREE, NONE, and VOID. The default value is FREE.

outflowType A SymbolicConstant specifying the control of material flow out of the Eulerian domain. Possible values are ZERO_PRESSURE, FREE, NON_REFLECTING, and EQUILIBRIUM. The default value is ZERO_PRESSURE.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing EulerianBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

definition A SymbolicConstant specifying the material flow conditions to be defined. Possible values are INFLOW, OUTFLOW, and BOTH. The default value is INFLOW.

inflowType A SymbolicConstant specifying the control of material flow into the Eulerian domain. Possible values are FREE, NONE, and VOID. The default value is FREE.

outflowType A SymbolicConstant specifying the control of material flow out of the Eulerian domain. Possible values are ZERO_PRESSURE, FREE, NON_REFLECTING, and EQUILIBRIUM. The default value is ZERO_PRESSURE.

Return value

`None.`

Exceptions

None.

Members

The EulerianBC object can have the following members:

name
A String specifying the boundary condition repository key.

definition
A SymbolicConstant specifying the flow conditions to be defined. Possible values are INFLOW, OUTFLOW, and BOTH. The default value is INFLOW.

inflowType
A SymbolicConstant specifying the control of material flow into the Eulerian domain. Possible values are FREE, NONE, and VOID. The default value is FREE.

outflowType
A SymbolicConstant specifying the control of flow of material out of the Eulerian domain. Possible values are ZERO_PRESSURE, FREE, NON_REFLECTING, and EQUILIBRIUM. The default value is ZERO_PRESSURE.

category
A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region
A *Region* object specifying the region to which the boundary condition is applied.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

EulerianBCState object

The EulerianBCState object stores the propagating data for an Eulerian boundary condition in a step. One instance of this object is created internally by the *EulerianBC* object for each step. The instance is also deleted internally by the *EulerianBC* object.

The EulerianBCState object has no constructor or methods.

The EulerianBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The EulerianBCState object has the following members:

definition

A SymbolicConstant specifying the material flow conditions to be defined. Possible values are INFLOW, OUTFLOW, and BOTH. The default value is INFLOW.

definitionState

A SymbolicConstant specifying the propagation state of the definition member. Possible values are UNSET, SET, and UNCHANGED.

inflowType

A SymbolicConstant specifying the material flow conditions to be defined. Possible values are FREE, NONE, and VOID. The default value is FREE.

inflowTypeState

A SymbolicConstant specifying the propagation state of the definition member. Possible values are UNSET, SET, and UNCHANGED.

outflowType

A SymbolicConstant specifying the material flow conditions to be defined. Possible values are ZERO_PRESSURE, FREE, NON_REFLECTING, and EQUILIBRIUM. The default value is ZERO_PRESSURE.

outflowTypeState

A SymbolicConstant specifying the propagation state of the definition member. Possible values are UNSET, SET, and UNCHANGED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE

- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **EULERIAN BOUNDARY*

EulerianMotionBC object

The EulerianMotionBC object stores the data for an Eulerian mesh motion boundary condition.

The EulerianMotionBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

EulerianMotionBC(...)

This method creates an EulerianMotionBC object.

Path

```
mdb.models[name].EulerianMotionBC
```

Required arguments

name	A String specifying the boundary condition repository key.
createStepName	A String specifying the name of the step in which the boundary condition is created.
instanceName	A String specifying the name of the Eulerian part instance.

Optional arguments

followRegion	A Boolean specifying whether the mesh will follow a regular surface region or an Eulerian surface. The default value is ON.
region	A *Region* object specifying the region to which the boundary condition is applied.
materialName	A String specifying the name of the Eulerian surface to follow. This argument applies only when *followRegion*=False.
ctrPosition1	A SymbolicConstant specifying the 1-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.
posPosition1	A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.
negPosition1	A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio1

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 1 direction. If *expansionRatio1*=`None`, then there is no upper limit. The default value is `None`.

contractRatio1

A Float specifying the lower bounds on the allowable scaling of the mesh in the 1 direction. The default value is 0.0.

ctrPosition2

A SymbolicConstant specifying the 2-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition2

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition2

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio2

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 2 direction. If *expansionRatio2*=`None`, then there is no upper limit. The default value is `None`.

contractRatio2

A Float specifying the lower bounds on the allowable scaling of the mesh in the 2 direction. The default value is 0.0.

ctrPosition3

A SymbolicConstant specifying the 3-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition3

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition3

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio3

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 3 direction. If *expansionRatio3*=`None`, then there is no upper limit. The default value is `None`.

contractRatio3

A Float specifying the lower bounds on the allowable scaling of the mesh in the 3 direction. The default value is 0.0.

allowContraction

A Boolean specifying whether the mesh is allowed to contract . The default value is ON.

aspectLimit

A Float specifying the maximum change in allowed aspect ratio (for any of the three mesh aspects, 1-2, 2-3, 3-1). The default value is 10.0.

vmaxFactor A Float specifying the multiplier for the mesh nodal velocity limit. The default value is 1.01.

volThreshold A Float specifying the lower bounds on the volume fraction when determining which nodes to include in the surface bounding box calculation for an Eulerian material surface. This argument applies only when *followRegion*=False. The default value is 0.5.

bufferSize `None` or a Float specifying the buffer between the surface box and the Eulerian section mesh bounding box. The default value is 2.0.

Return value

An EulerianMotionBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing EulerianMotionBC object in the step where it is created.

Required arguments

None.

Optional arguments

instanceName A String specifying the name of the Eulerian part instance.

followRegion A Boolean specifying whether the mesh will follow a regular surface region or an Eulerian surface. The default value is ON.

region A *Region* object specifying the region to which the boundary condition is applied.

materialName A String specifying the name of the Eulerian surface to follow. This argument applies only when *followRegion*=False.

ctrPosition1 A SymbolicConstant specifying the 1-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition1 A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition1 A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio1

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 1 direction. If *expansionRatio1*=`None`, then there is no upper limit. The default value is `None`.

contractRatio1

A Float specifying the lower bounds on the allowable scaling of the mesh in the 1 direction. The default value is 0.0.

ctrPosition2

A SymbolicConstant specifying the 2-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition2

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition2

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio2

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 2 direction. If *expansionRatio2*=`None`, then there is no upper limit. The default value is `None`.

contractRatio2

A Float specifying the lower bounds on the allowable scaling of the mesh in the 2 direction. The default value is 0.0.

ctrPosition3

A SymbolicConstant specifying the 3-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition3

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition3

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio3

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 3 direction. If *expansionRatio3*=`None`, then there is no upper limit. The default value is `None`.

contractRatio3

A Float specifying the lower bounds on the allowable scaling of the mesh in the 3 direction. The default value is 0.0.

allowContraction

A Boolean specifying whether the mesh is allowed to contract . The default value is ON.

aspectLimit

A Float specifying the maximum change in allowed aspect ratio (for any of the three mesh aspects, 1-2, 2-3, 3-1). The default value is 10.0.

vmaxFactor
A Float specifying the multiplier for the mesh nodal velocity limit. The default value is 1.01.

volThreshold
A Float specifying the lower bounds on the volume fraction when determining which nodes to include in the surface bounding box calculation for an Eulerian material surface. This argument applies only when *followRegion*=False. The default value is 0.5.

bufferSize
`None` or a Float specifying the buffer between the surface box and the Eulerian section mesh bounding box. The default value is 2.0.

Return value
`None`.

Exceptions
None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing EulerianMotionBC object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

ctrPosition1
A SymbolicConstant specifying the 1-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition1
A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition1
A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio1
`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 1 direction. If *expansionRatio1*=`None`, then there is no upper limit. The default value is `None`.

contractRatio1
A Float specifying the lower bounds on the allowable scaling of the mesh in the 1 direction. The default value is 0.0.

ctrPosition2
A SymbolicConstant specifying the 2-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition2

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition2

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio2

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 2 direction. If *expansionRatio2*=`None`, then there is no upper limit. The default value is `None`.

contractRatio2

A Float specifying the lower bounds on the allowable scaling of the mesh in the 2 direction. The default value is 0.0.

ctrPosition3

A SymbolicConstant specifying the 3-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition3

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition3

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio3

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 3 direction. If *expansionRatio3*=`None`, then there is no upper limit. The default value is `None`.

contractRatio3

A Float specifying the lower bounds on the allowable scaling of the mesh in the 3 direction. The default value is 0.0.

allowContraction

A Boolean specifying whether the mesh is allowed to contract . The default value is ON.

aspectLimit

A Float specifying the maximum change in allowed aspect ratio (for any of the three mesh aspects, 1-2, 2-3, 3-1). The default value is 10.0.

vmaxFactor

A Float specifying the multiplier for the mesh nodal velocity limit. The default value is 1.01.

volThreshold

A Float specifying the lower bounds on the volume fraction when determining which nodes to include in the surface bounding box calculation for an Eulerian material surface. This argument applies only when *followRegion*=False. The default value is 0.5.

bufferSize

`None` or a Float specifying the buffer between the surface box and the Eulerian section mesh bounding box. The default value is 2.0.

Return value
`None.`

Exceptions
None.

Members
The EulerianMotionBC object can have the following members:

name
A String specifying the boundary condition repository key.

followRegion
A Boolean specifying whether the mesh will follow a regular surface region or an Eulerian surface. The default value is ON.

ctrPosition1
A SymbolicConstant specifying the 1-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

ctrPosition2
A SymbolicConstant specifying the 2-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

ctrPosition3
A SymbolicConstant specifying the 3-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition1
A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

posPosition2
A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

posPosition3
A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition1
A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition2
A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition3
A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio1

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 1 direction. If *expansionRatio1*=`None`, then there is no upper limit. The default value is `None`.

expansionRatio2

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 2 direction. If *expansionRatio2*=`None`, then there is no upper limit. The default value is `None`.

expansionRatio3

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 3 direction. If *expansionRatio3*=`None`, then there is no upper limit. The default value is `None`.

contractRatio1

A Float specifying the lower bounds on the allowable scaling of the mesh in the 1 direction. The default value is 0.0.

contractRatio2

A Float specifying the lower bounds on the allowable scaling of the mesh in the 2 direction. The default value is 0.0.

contractRatio3

A Float specifying the lower bounds on the allowable scaling of the mesh in the 3 direction. The default value is 0.0.

allowContraction

A Boolean specifying whether the mesh is allowed to contract . The default value is ON.

aspectLimit

A Float specifying the maximum change in allowed aspect ratio (for any of the three mesh aspects, 1-2, 2-3, 3-1). The default value is 10.0.

vmaxFactor

A Float specifying the multiplier for the mesh nodal velocity limit. The default value is 1.01.

volThreshold

A Float specifying the lower bounds on the volume fraction when determining which nodes to include in the surface bounding box calculation for an Eulerian material surface. This argument applies only when *followRegion*=False. The default value is 0.5.

bufferSize

`None` or a Float specifying the buffer between the surface box and the Eulerian section mesh bounding box. The default value is 2.0.

instanceName

A String specifying the name of the Eulerian part instance.

materialName

A String specifying the name of the Eulerian surface to follow. This argument applies only when *followRegion*=False.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

EulerianMotionBCState object

The EulerianMotionBCState object stores the propagating data for an Eulerian mesh motion boundary condition in a step. One instance of this object is created internally by the *EulerianMotionBC* object for each step. The instance is also deleted internally by the *EulerianMotionBC* object.

The EulerianMotionBCState object has no constructor or methods.

The EulerianMotionBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The EulerianMotionBCState object has the following members:

ctrPosition1

A SymbolicConstant specifying the 1-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

ctrPosition2

A SymbolicConstant specifying the 2-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

ctrPosition3

A SymbolicConstant specifying the 3-direction translational constraint on the center of the Eulerian mesh. Possible values are FREE and FIXED. The default value is FREE.

posPosition1

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

posPosition2

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

posPosition3

A SymbolicConstant specifying the translational constraint on the positive (maximum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition1

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 1 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition2

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 2 direction. Possible values are FREE and FIXED. The default value is FREE.

negPosition3

A SymbolicConstant specifying the translational constraint on the negative (minimum) bounds of the mesh in the 3 direction. Possible values are FREE and FIXED. The default value is FREE.

expansionRatio1

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 1 direction. If *expansionRatio1*=`None`, then there is no upper limit. The default value is `None`.

expansionRatio2

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 2 direction. If *expansionRatio2*=`None`, then there is no upper limit. The default value is `None`.

expansionRatio3

`None` or a Float specifying the upper bounds on the allowable scaling of the mesh in the 3 direction. If *expansionRatio3*=`None`, then there is no upper limit. The default value is `None`.

contractRatio1

A Float specifying the lower bounds on the allowable scaling of the mesh in the 1 direction. The default value is 0.0.

contractRatio2

A Float specifying the lower bounds on the allowable scaling of the mesh in the 2 direction. The default value is 0.0.

contractRatio3

A Float specifying the lower bounds on the allowable scaling of the mesh in the 3 direction. The default value is 0.0.

allowContraction

A Boolean specifying whether the mesh is allowed to contract . The default value is ON.

aspectLimit

A Float specifying the maximum change in allowed aspect ratio (for any of the three mesh aspects, 1-2, 2-3, 3-1). The default value is 10.0.

vmaxFactor

A Float specifying the multiplier for the mesh nodal velocity limit. The default value is 1.01.

volThreshold

A Float specifying the lower bounds on the volume fraction when determining which nodes to include in the surface bounding box calculation for an Eulerian material surface. This argument applies only when *followRegion*=False. The default value is 0.5.

bufferSize

`None` or a Float specifying the buffer between the surface box and the Eulerian section mesh bounding box. The default value is 2.0.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **EULERIAN MESH MOTION*

FluidCavityPressureBC object

The FluidCavityPressureBC object stores the data for a fluid cavity pressure boundary condition.

The FluidCavityPressureBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

FluidCavityPressureBC(...)

This method creates a FluidCavityPressureBC object.

Path

```
mdb.models[name].FluidCavityPressureBC
```

Required arguments

name	A String specifying the boundary condition repository key.
createStepName	A String specifying the name of the step in which the boundary condition is created.
fluidCavity	A String specifying the name of a Fluid Cavity Interaction.

Optional arguments

magnitude	A Float specifying the fluid cavity pressure magnitude. The default value is 0.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
fixed	A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

A FluidCavityPressureBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing FluidCavityPressureBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidCavityPressureBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing FluidCavityPressureBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the fluid cavity pressure magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The FluidCavityPressureBC object can have the following members:

name A String specifying the boundary condition repository key.

fluidCavity A String specifying the name of a Fluid Cavity Interaction.

category A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

FluidCavityPressureBCState object

The FluidCavityPressureBCState object stores the propagating data for a fluid cavity pressure boundary condition in a step. One instance of this object is created internally by the *FluidCavityPressureBC* object for each step. The instance is also deleted internally by the *FluidCavityPressureBC* object.

The FluidCavityPressureBCState object has no constructor or methods.

The FluidCavityPressureBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The FluidCavityPressureBCState object has the following members:

magnitude

A Float specifying the fluid cavity pressure magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the fluid cavity pressure magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 8)

IonConcentrationBC object

The IonConcentrationBC object stores the data for an ion concentration boundary condition.

The IonConcentrationBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

IonConcentrationBC(...)

This method creates an IonConcentrationBC object.

Path

```
mdb.models[name].IonConcentrationBC
```

Required arguments

name A String specifying the repository key.

region A *Region* object specifying the region to which the predefined field is applied.

Optional arguments

distributionType A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this predefined field. The field argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

value A Float specifying the initial value of ion concentration.

Return value

An IonConcentrationBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing IonConcentrationBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IonConcentrationBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing IonConcentrationBC object in the specified step.

Required arguments

name A String specifying the repository key.

region A Region object specifying the region to which the predefined field is applied.

Optional arguments

distributionType A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

field A String specifying the name of the AnalyticalField or DiscreteField object associated with this predefined field. The field argument applies only when *distributionType* is FIELD or *distributionType* is DISCRETE_FIELD. The default value is an empty string.

value

value A Float specifying the initial value of ion concentration.

Return value

`None.`

Exceptions

None.

Members

The IonConcentrationBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL, ELECTROCHEMICAL, and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

IonConcentrationBCState object

The IonConcentrationBCState object stores the propagating data for an ion concentration boundary condition in a step. One instance of this object is created internally by the *IonConcentrationBC* object for each step. The instance is also deleted internally by the *IonConcentrationBC* object.

The IonConcentrationBCState object has no constructor or methods.

The IonConcentrationBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The IonConcentrationBCState object has the following members:

magnitude

A Float specifying the ion concentration magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the ion concentration magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 9)

MagneticVectorPotentialBC object

The MagneticVectorPotentialBC object stores the data for a magnetic vector potential boundary condition.

The MagneticVectorPotentialBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

MagneticVectorPotentialBC(...)

This method creates a MagneticVectorPotentialBC object.

Path

```
mdb.models[name].MagneticVectorPotentialBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

component1 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET

component2 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

component3 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A MagneticVectorPotentialBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing MagneticVectorPotentialBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MagneticVectorPotentialBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing MagneticVectorPotentialBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

component1 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 1-direction. Possible values for the SymbolicConstant are SET and UNCHANGED.

component2 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 2-direction. Possible values for the SymbolicConstant are SET and UNCHANGED.

component3 A Complex, or a SymbolicConstant specifying the magnetic vector potential component in the 3-direction. Possible values for the SymbolicConstant areSET and UNCHANGED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None`.

Exceptions

None.

Members

The MagneticVectorPotentialBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

category A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region A *Region* object specifying the region to which the boundary condition is applied.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

MaterialFlowBC object

The MaterialFlowBC object stores the data for a connector material flow boundary condition.

The MaterialFlowBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

MaterialFlowBC(...)

This method creates a MaterialFlowBC object.

Path

```
mdb.models[name].MaterialFlowBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the material flow magnitude. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

A MaterialFlowBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing MaterialFlowBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MaterialFlowBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing MaterialFlowBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the material flow magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The MaterialFlowBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

MaterialFlowBCState object

The MaterialFlowBCState object stores the propagating data for a connector material flow boundary condition in a step. One instance of this object is created internally by the *MaterialFlowBC* object for each step. The instance is also deleted internally by the *MaterialFlowBC* object.

The MaterialFlowBCState object has no constructor or methods.

The MaterialFlowBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The MaterialFlowBCState object has the following members:

magnitude

A Float specifying the material flow magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the material flow magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 10)

PorePressureBC object

The PorePressureBC object stores the data for a pore pressure boundary condition.

The PorePressureBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

PorePressureBC(...)

This method creates a PorePressureBC object.

Path

```
mdb.models[name].PorePressureBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the pore pressure magnitude. The default value is 0. The *magnitude* argument is optional if *distributionType*=USER_DEFINED.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

A PorePressureBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing PorePressureBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PorePressureBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing PorePressureBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the pore pressure magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The PorePressureBC object can have the following members:

name

A String specifying the boundary condition repository key.

distributionType

A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName

A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

PorePressureBCState object

The PorePressureBCState object stores the propagating data for a pore pressure boundary condition in a step. One instance of this object is created internally by the *PorePressureBC* object for each step. The instance is also deleted internally by the *PorePressureBC* object.

The PorePressureBCState object has no constructor or methods.

The PorePressureBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The PorePressureBCState object has the following members:

magnitude

A Float specifying the pore pressure magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the pore pressure magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 8)

RetainedNodalDofsBC object

The RetainedNodalDofsBC object stores the data for a retained nodal dofs boundary condition.

The RetainedNodalDofsBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

RetainedNodalDofsBC(...)

This method creates a RetainedNodalDofsBC object.

Path

```
mdb.models[name].RetainedNodalDofsBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

u1 A Boolean specifying whether to retain the degree of freedom in the 1-direction. The default value is OFF indicating that the degree of freedom is not retained.

u2 A Boolean specifying whether to retain the degree of freedom in the 2-direction. The default value is OFF indicating that the degree of freedom is not retained.

u3 A Boolean specifying whether to retain the degree of freedom in the 3-direction. The default value is OFF indicating that the degree of freedom is not retained.

ur1 A Boolean specifying whether to retain the rotational degree of freedom about the 1-direction. The default value is OFF indicating that the degree of freedom is not retained.

ur2 A Boolean specifying whether to retain the rotational degree of freedom about the 2-direction. The default value is OFF indicating that the degree of freedom is not retained.

ur3 A Boolean specifying whether to retain the rotational degree of freedom about the 3-direction. The default value is OFF indicating that the degree of freedom is not retained.

Return value

A RetainedNodalDofsBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing RetainedNodalDofsBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RetainedNodalDofsBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing RetainedNodalDofsBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

u1 A Boolean specifying whether to retain the degree of freedom in the 1-direction.

u2 A Boolean specifying whether to retain the degree of freedom in the 2-direction.

u3 A Boolean specifying whether to retain the degree of freedom in the 3-direction.

ur1 A Boolean specifying whether to retain the rotational degree of freedom about the 1-direction.

ur2 A Boolean specifying whether to retain the rotational degree of freedom about the 2-direction.

ur3 A Boolean specifying whether to retain the rotational degree of freedom about the 3-direction.

Return value

`None.`

Exceptions

None.

Members

The RetainedNodalDofsBC object can have the following members:

name
A String specifying the boundary condition repository key.

category
A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region
A *Region* object specifying the region to which the boundary condition is applied.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

SecondaryBaseBC object

The SecondaryBaseBC object stores the data for a secondary base boundary condition.

The SecondaryBaseBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

SecondaryBaseBC(...)

This method creates a SecondaryBaseBC object.

Path

```
mdb.models[name].SecondaryBaseBC
```

Required arguments

name	A String specifying the boundary condition repository key.
createStepName	A String specifying the name of the step in which the boundary condition is created.
regions	A *RegionArray* object specifying the region to which the boundary condition is applied. Note that the usual *region* is ignored. The default value is MODEL.
dofs	A sequence of sequences of Ints specifying the constrained degrees-of-freedom.

Optional arguments

None.

Return value

A SecondaryBaseBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SecondaryBaseBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SecondaryBaseBC* method, except for the *name* and *createStepName* arguments.

Return value

`None`.

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing SecondaryBaseBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

None.

Return value

`None`.

Exceptions

None.

Members

The SecondaryBaseBC object can have the following members:

name A String specifying the boundary condition repository key.

dofs A tuple of tuples of Ints specifying the constrained degrees-of-freedom.

regions A *RegionArray* object specifying the region to which the boundary condition is applied. Note that the usual *region* is ignored. The default value is MODEL.

category A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region A *Region* object specifying the region to which the boundary condition is applied.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

SecondaryBaseBCState object

The SecondaryBaseBCState object stores the propagating data for a secondary base boundary condition in a step. One instance of this object is created internally by the *SecondaryBaseBC* object for each step. The instance is also deleted internally by the *SecondaryBaseBC* object.

The SecondaryBaseBCState object has no constructor or methods.

The SecondaryBaseBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The SecondaryBaseBCState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY*, BASE NAME

SubmodelBC object

The SubmodelBC object stores the data for a submodel boundary condition.

The SubmodelBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

SubmodelBC(...)

This method creates a SubmodelBC object.

Path

```
mdb.models[name].SubmodelBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

dof A sequence of Ints specifying the degrees of freedom to which the boundary condition is applied.

globalStep A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step.

timeScale A Boolean specifying whether to scale the time variable for the driven nodes' amplitude functions to match the submodel analysis step time. The default value is OFF.

shellThickness A Float specifying the thickness of the shell in the global model. This argument is required for shell-to-solid submodeling and is not applicable to other submodels. The default value is 0.0.

Optional arguments

globalDrivingRegion A String specifying the element set in the global model that will be searched for elements whose responses will be used to drive the submodel. An empty string indicates that the entire global model will be searched. The default value is an empty string.

absoluteExteriorTolerance `None` or a Float specifying the absolute value by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is `None`.

exteriorTolerance

`None` or a Float specifying the fraction of the average element size in the global model by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is 0.05.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

globalIncrement

An Int specifying the increment number in the global model step from which the solution will be used to specify the values of the driven variables. If *globalIncrement*=0, the solution from the last increment will be used. The *globalIncrement* argument is applicable only for linear perturbation steps. The default value is 0.

centerZoneSize

A Float specifying the thickness of the center zone size around the shell midsurface. The default value is `None`.

intersectionOnly

A Boolean specifying whether to ignore driven nodes that lie outside the region of elements of the global model after accounting for the exterior search tolerance. The default value is OFF.

Return value

A SubmodelBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing SubmodelBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SubmodelBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing SubmodelBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is ON.

dof A sequence of Ints specifying the degrees of freedom to which the boundary condition is applied. The *dof* argument is applicable only if *fixed*=OFF.

globalStep A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step. The *globalStep* argument is applicable only if *fixed*=OFF.

globalIncrement An Int specifying the increment number in the global model step at which the solution will be used to specify the values of the driven variables. If *globalIncrement*=0, the solution from the last increment will be used. The *globalIncrement* argument is applicable only for linear perturbation steps and if *fixed*=OFF. The default value is 0.

centerZoneSize A Float specifying the thickness of the center zone size around the shell midsurface. The default value is `None`.

The *centerZoneSize* argument is applicable only if *fixed*=OFF.

Return value

`None.`

Exceptions

None.

Members

The SubmodelBC object can have the following members:

name A String specifying the boundary condition repository key.

shellThickness A Float specifying the thickness of the shell in the global model. This argument is required for shell-to-solid submodeling and is not applicable to other submodels. The default value is 0.0.

absoluteExteriorTolerance `None` or a Float specifying the absolute value by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is `None`.

exteriorTolerance `None` or a Float specifying the fraction of the average element size in the global model by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is 0.05.

globalDrivingRegion

A String specifying the element set in the global model that will be searched for elements whose responses will be used to drive the submodel. An empty string indicates that the entire global model will be searched. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

SubmodelBCState object

The SubmodelBCState object stores the propagating data for a Submodel boundary condition in a step. One instance of this object is created internally by the *SubmodelBC* object for each step. The instance is also deleted internally by the *SubmodelBC* object.

The SubmodelBCState object has no constructor or methods.

The SubmodelBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The SubmodelBCState object has the following members:

dofState

A SymbolicConstant specifying the propagation state of the *dof* member. Possible values are SET and UNCHANGED.

globalStepState

A SymbolicConstant specifying the propagation state of the *globalStep* member. Possible values are SET and UNCHANGED.

globalIncrement

An Int specifying the increment number in the global model step at which the solution will be used to specify the values of the driven variables. This argument is applicable only for linear perturbation steps.

globalIncrementState

A SymbolicConstant specifying the propagation state of the *globalIncrement* member. Possible values are SET and UNCHANGED.

centerZoneSize

`None` or a Float specifying the thickness of the center zone size around the shell midsurface. The default value is `None`.

centerZoneSizefState

A SymbolicConstant specifying the propagation state of the *centerZoneSize* member. Possible values are SET and UNCHANGED.

scale

`None` or a Float specifying a scaling value applied to the applied displacements at the interface. The default value is 1.0.

scaleState

A SymbolicConstant specifying the propagation state of the *scale* member. Possible values are SET and UNCHANGED.

globalStep

A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step.

dof

A tuple of Ints specifying the degrees of freedom to which the boundary condition is applied.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **SUBMODEL*
- **BOUNDARY*, SUBMODEL

TemperatureBC object

The TemperatureBC object stores the data for a temperature boundary condition.

The TemperatureBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

TemperatureBC(...)

This method creates a TemperatureBC object.

Path

```
mdb.models[name].TemperatureBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

magnitude A Float specifying the temperature magnitude. The default value is 0.

dof A sequence of Ints specifying the degrees of freedom to which the boundary condition is applied. The default value is (11,).

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fixed A Boolean specifying whether the boundary condition should remain fixed at the current values at the start of the step. The default value is OFF.

Return value

A TemperatureBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing TemperatureBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TemperatureBC* method, except for the *name* and *createStepName* arguments.

Return value

```
None.
```

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing TemperatureBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

magnitude A Float or the SymbolicConstant FREED specifying the temperature magnitude.

dof A sequence of Ints specifying the degrees of freedom to which the boundary condition is applied.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

```
None.
```

Exceptions

None.

Members

The TemperatureBC object can have the following members:

name
A String specifying the boundary condition repository key.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName
A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category
A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region
A *Region* object specifying the region to which the boundary condition is applied.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

TemperatureBCState object

The TemperatureBCState object stores the propagating data for a temperature boundary condition in a step. One instance of this object is created internally by the *TemperatureBC* object for each step. The instance is also deleted internally by the *TemperatureBC* object.

The TemperatureBCState object has no constructor or methods.

The TemperatureBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The TemperatureBCState object has the following members:

magnitude

A Float specifying the temperature magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the temperature magnitude. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

dofState

A SymbolicConstant specifying the propagation state of the *dof* member. Possible values are SET and UNCHANGED.

dof

A tuple of Ints specifying the degrees of freedom to which the boundary condition is applied.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude
A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY* (degree of freedom: 11 for solids; 11, 12, etc. for shells)

TypeBC object

The TypeBC object stores the data for several types of predefined boundary conditions that are commonly used in stress/displacement analyses.

The TypeBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

EncastreBC(...)

This method creates an encastre TypeBC object.

Path

```
mdb.models[name].EncastreBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

PinnedBC(...)

This method creates a pinned TypeBC object.

Path

```
mdb.models[name].PinnedBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

XsymmBC(...)

This method creates a TypeBC object that specifies symmetry about the *X*-axis.

Path

```
mdb.models[name].XsymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

YsymmBC(...)

This method creates a TypeBC object that specifies symmetry about the *Y*-axis.

Path

```
mdb.models[name].YsymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

ZsymmBC(...)

This method creates a TypeBC object that specifies symmetry about the *Z*-axis.

Path

```
mdb.models[name].ZsymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

XasymmBC(...)

This method creates a TypeBC object that specifies antisymmetry about the *X*-axis.

Path

```
mdb.models[name].XasymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

YasymmBC(...)

This method creates a TypeBC object that specifies antisymmetry about the *Y*-axis.

Path

```
mdb.models[name].YasymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

ZasymmBC(...)

This method creates a TypeBC object that specifies antisymmetry about the *Z*-axis.

Path

```
mdb.models[name].ZasymmBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value

A TypeBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing TypeBC object in the step where it is created.

Required arguments

None.

Optional arguments

region A *Region* object specifying the region to which the boundary condition is applied.

typeName A SymbolicConstant specifying the predefined boundary condition type. Possible values are XSYMM, YSYMM, ZSYMM, XASYMM, YASYMM, ZASYMM, PINNED, and ENCASTRE.

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

Return value
`None`.

Exceptions
None.

`setValuesInStep(...)`

This method always returns a value error for a TypeBC; it is inherited from the *BoundaryCondition* object.

Required arguments
stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments
typeName A SymbolicConstant specifying the predefined boundary condition type. Possible values are XSYMM, YSYMM, ZSYMM, XASYMM, YASYMM, ZASYMM, PINNED, and ENCASTRE.

Return value
`None`.

Exceptions
Value Error: A Symmetry/Antisymmetry/Encastre BC cannot be edited in a propagated step.

Members
The TypeBC object can have the following members:

name A String specifying the boundary condition repository key.

buckleCase A SymbolicConstant specifying how the boundary condition is defined in a **BUCKLE* analysis. Possible values are NOT_APPLICABLE, STRESS_PERTURBATION, BUCKLING_MODES, and PERTURBATION_AND_BUCKLING. The default value is NOT_APPLICABLE.

category A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

TypeBCState object

The TypeBCState object stores the propagating data for a predefined boundary condition in a step. One instance of this object is created internally by the *TypeBC* object for each step. The instance is also deleted internally by the *TypeBC* object.

The TypeBCState object has no constructor or methods.

The TypeBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The TypeBCState object has the following members:

typeName

A SymbolicConstant specifying the predefined boundary condition type. Possible values are XSYMM, YSYMM, ZSYMM, XASYMM, YASYMM, ZASYMM, PINNED, and ENCASTRE.

typeNameState

A SymbolicConstant specifying the propagation state of the predefined boundary condition type. The only possible value is UNCHANGED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY*, TYPE=XSYMM, YSYMM, ZSYMM, XASYMM, YASYMM, ZASYMM, PINNED, or ENCASTRE

VelocityBaseMotionBC object

The VelocityBaseMotionBC object stores the data for a velocity base motion boundary condition.

The VelocityBaseMotionBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

VelocityBaseMotionBC(...)

This method creates a VelocityBaseMotionBC object.

Path

```
mdb.models[name].VelocityBaseMotionBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

dof A SymbolicConstant specifying the constrained degree-of-freedom. Possible values for the SymbolicConstant are U1, U2, U3, UR1, UR2, UR3. The default value is U1.

Optional arguments

amplitudeScaleFactor A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

centerOfRotation A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation A *CorrelationArray* object.

secondaryBase A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

useComplex A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A VelocityBaseMotionBC object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing VelocityBaseMotionBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VelocityBaseMotionBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing VelocityBaseMotionBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The VelocityBaseMotionBC object can have the following members:

name

A String specifying the boundary condition repository key.

amplitudeScaleFactor

A Float specifying the scale factor for the amplitude curve. The default value is 1.0.

useComplex

A Boolean specifying whether to define the imaginary (out-of-plane) portion of the base motion record given by amplitude definition. The default value is OFF.

centerOfRotation

A *ModelDot* object specifying a tuple containing one center of rotation. The default value is the global origin. This argument applies only when *dof*=UR1, UR2, or UR3.

correlation

A *CorrelationArray* object.

secondaryBase

A String specifying the name of the *SecondaryBaseBC* object associated with this boundary condition. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

VelocityBaseMotionBCState object

The VelocityBaseMotionBCState object stores the propagating data for a velocity base motion boundary condition in a step. One instance of this object is created internally by the *VelocityBaseMotionBC* object for each step. The instance is also deleted internally by the *VelocityBaseMotionBC* object.

The VelocityBaseMotionBCState object has no constructor or methods.

The VelocityBaseMotionBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The VelocityBaseMotionBCState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BASE MOTION*

VelocityBC object

The VelocityBC object stores the data for a velocity boundary condition.

The VelocityBC object is derived from the *BoundaryCondition* object.

Access

```
import load
mdb.models[name].boundaryConditions[name]
```

VelocityBC(...)

This method creates a VelocityBC object.

Path

```
mdb.models[name].VelocityBC
```

Required arguments

name A String specifying the boundary condition repository key.

createStepName A String specifying the name of the step in which the boundary condition is created.

region A *Region* object specifying the region to which the boundary condition is applied.

Optional arguments

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

v1 A Float or a SymbolicConstant specifying the velocity component in the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.



Note:

Although *v1*, *v2*, *v3*, *vr1*, *vr2*, and *vr3* are optional arguments, at least one of them must be specified.

v2 A Float or a SymbolicConstant specifying the velocity component in the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

v3 A Float or a SymbolicConstant specifying the velocity component in the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr1 A Float or a SymbolicConstant specifying the rotational velocity component about the 1-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr2
A Float or a SymbolicConstant specifying the rotational velocity component about the 2-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

vr3
A Float or a SymbolicConstant specifying the rotational velocity component about the 3-direction. Possible values for the SymbolicConstant are UNSET and SET. The default value is UNSET.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the boundary condition has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

distributionType
A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

Return value

A VelocityBC object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing VelocityBC object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VelocityBC* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing VelocityBC object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the boundary condition is modified.

Optional arguments

v1 A Float or a SymbolicConstant specifying the velocity component in the 1-direction. Possible values for the SymbolicConstant are SET and FREED.

v2 A Float or a SymbolicConstant specifying the velocity component in the 2-direction. Possible values for the SymbolicConstant are SET and FREED.

v3 A Float or a SymbolicConstant specifying the velocity component in the 3-direction. Possible values for the SymbolicConstant are SET and FREED.

vr1 A Float or a SymbolicConstant specifying the rotational velocity component about the 1-direction. Possible values for the SymbolicConstant are SET and FREED.

vr2 A Float or a SymbolicConstant specifying the rotational velocity component about the 2-direction. Possible values for the SymbolicConstant are SET and FREED.

vr3 A Float or a SymbolicConstant specifying the rotational velocity component about the 3-direction. Possible values for the SymbolicConstant are SET and FREED.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the boundary condition is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The VelocityBC object can have the following members:

name A String specifying the boundary condition repository key.

distributionType A SymbolicConstant specifying how the boundary condition is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

fieldName A String specifying the name of the *AnalyticalField* object associated with this boundary condition. The *fieldName* argument applies only when *distributionType*=FIELD. The default value is an empty string.

category

A SymbolicConstant specifying the category of the boundary condition. Possible values are MECHANICAL and THERMAL.

region

A *Region* object specifying the region to which the boundary condition is applied.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the boundary condition's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

VelocityBCState object

The VelocityBCState object stores the propagating data for a velocity boundary condition in a step. One instance of this object is created internally by the *VelocityBC* object for each step. The instance is also deleted internally by the *VelocityBC* object.

The VelocityBCState object has no constructor or methods.

The VelocityBCState object is derived from the *BoundaryConditionState* object.

Access

```
import load
mdb.models[name].steps[name].boundaryConditionStates[name]
```

Members

The VelocityBCState object has the following members:

v1
A Float specifying the velocity component in the 1-direction.

v2
A Float specifying the velocity component in the 2-direction.

v3
A Float specifying the velocity component in the 3-direction.

vr1
A Float specifying the rotational velocity component about the 1-direction.

vr2
A Float specifying the rotational velocity component about the 2-direction.

vr3
A Float specifying the rotational velocity component about the 3-direction.

v1State
A SymbolicConstant specifying the propagation state of the velocity component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v2State
A SymbolicConstant specifying the propagation state of the velocity component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

v3State
A SymbolicConstant specifying the propagation state of the velocity component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr1State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 1-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr2State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 2-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

vr3State
A SymbolicConstant specifying the propagation state of the rotational velocity component about the 3-direction. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the amplitude reference. Possible values are UNSET, SET, UNCHANGED, FREED, and MODIFIED.

status

A SymbolicConstant specifying the propagation state of the *BoundaryConditionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- PROPAGATED_FROM_BASE_STATE
- MODIFIED_FROM_BASE_STATE
- DEACTIVATED_FROM_BASE_STATE
- BUILT_INTO_MODES

amplitude

A String specifying the name of the amplitude reference. The String is empty if the boundary condition has no amplitude reference.

Corresponding analysis keywords

- **BOUNDARY*, TYPE=VELOCITY (degree of freedom: 1, 2, 3, 4, 5, or 6)

Calibration commands

The calibration commands are used for material calibration.

In this section:

- *Calibration object*
- *Behavior object*
- *DataSet object*

Calibration object

A Calibration object is the object used to specify a material calibration. The Calibration object stores the data that is used for specifying materials from test data.

Access

```
import calibration
mdb.models[name].calibrations[name]
```

Calibration(...)

This method creates a Calibration object.

Path

```
mdb.models[name].Calibration
```

Required arguments

name A String specifying the name of the new calibration.

Optional arguments

None.

Return value

A Calibration object.

Exceptions

InvalidNameError.

Members

The Calibration object has members with the same names and descriptions as the arguments to the *Calibration* method.

In addition, the Calibration object can have the following members:

dataSets A *DataSet* object.

behaviors A *Behavior* object.

Behavior object

The Behavior object specifies the method used for calibrating a material.

Access

```
import calibration
mdb.models[name].calibrations[name].behaviors[name]
```

Behavior(...)

This method creates a Behavior object.

Path

```
mdb.models[name].calibrations[name].Behavior
```

Required arguments

name A String specifying the name of the new behavior.

typeName A String specifying the type of the new Behavior. Values can be "ElasIsoBehavior", "ElasPlasIsoBehavior", "FeFpBehavior", or a user plug-in behavior type.

Optional arguments

None.

Return value

A Behavior object.

Exceptions

InvalidNameError.

setValues(...)

This method modifies the data for an existing behavior object.

Required arguments

None.

Optional arguments

E Young's modulus. Only valid if the behavior type is ElasIsoBehavior.

nu
Poisson's ratio. Only valid if the behavior type is ElasIsoBehavior.

ds1Name
The name of the first data set. Only valid if the behavior type is ElasIsoBehavior or ElasPlasIsoBehavior

ds2Name
The name of the second data set. Only valid if the behavior type is ElasIsoBehavior or ElasPlasIsoBehavior

materialName
Material Name.

yieldPoint
Stress/strain value for the material yield point.Only valid if the behavior type is ElasPlasIsoBehavior

ultimatePoint
Stress/strain value for the material ultimate point.Only valid if the behavior type is ElasPlasIsoBehavior

plasticPoints
Stress/strain values for the plastic portion of material curve. Only valid if the behavior type is ElasPlasIsoBehavior

PoissonsRatio
Poisson's Ratio. Only valid if the behavior type is ElasPlasIsoBehavior

elasticModulus
Young's Modulus for the elastic portion of the material curve. Only valid if the behavior type is ElasPlasIsoBehavior

plasticPointsRange
Extent of the material plastic points. Only valid if the behavior type is ElasPlasIsoBehavior

name
Name of the behavior.

uniaxialName
Name of the uniaxial dataset. Only valid if the behavior type is FeFpBehavior

biaxialName
Name of the biaxial dataset. Only valid if the behavior type is FeFpBehavior

interpolation
'linear' specifies linear interpolation between data points, otherwise 'logarithmic'. Only valid if the behavior type is FeFpBehavior

uniWeight
Uniaxial weight factor, uniWeight + biWeight should equal 1.0. Only valid if the behavior type is FeFpBehavior

biWeight
Biaxial weight factor, uniWeight + biWeight should equal 1.0. Only valid if the behavior type is FeFpBehavior

uMullinsReload
A List of strings, specifying names of reloading DataSet objects obtained from uniaxial test data. Only valid if the behavior is of type FeFpBehavior

uMullinsUnload
A List of strings, specifying names of reloading DataSet objects obtained from uniaxial test data. Only valid if the behavior is of type FeFpBehavior

uPYieldPoint
A tuple specifying the coordinates of yield point of the permanent data set. Only valid if the behavior is of type FeFpBehavior

uPermSet
A List of strings, specifying names of permanent DataSet objects obtained from uniaxial test data. Only valid if the behavior is of type FeFpBehavior

uPrimary
A string specifying name of Primary DataSet object.Only valid if the behavior is of type FeFpBehavior

bMullinsReload
A List of strings, specifying names of reloading DataSet objects obtained from biaxial test data. Only valid if the behavior is of type FeFpBehavior

bMullinsUnload
A List of strings, specifying names of unloading DataSet objects obtained from biaxial test data. Only valid if the behavior is of type FeFpBehavior

bPYieldPoint
A tuple specifying the coordinates of yield point of the permanent data set. Only valid if the behavior is of type FeFpBehavior

bPermSet
A List of strings, specifying names of permanent DataSet objects obtained from biaxial test data. Only valid if the behavior is of type FeFpBehavior

bPrimary
A string specifying name of Primary DataSet object. Only valid if the behavior is of type FeFpBehavior

Return value
`None.`

Exceptions
None.

`mapToMaterial(...)`

This method appends the calibration data obtained from the DataSet object to an existing material object. In the case of ElasIsoBehavior, it appends the young's modulus and poisson's ratio. For ElasPlasIsoBehavior it appends the young's modulus, poisson's ratio and plastic points range and for FeFpBehavior it appends plastic points range and Mullins effect properties.

Required arguments
materialName
A String specifying the name of the existing material

Optional arguments
None.

Return value
`None.`

Exceptions

None.

`compute_E(...)`

This method computes the value of young's modulus from the existing DataSet object. The method is only valid for ElasIsoBehavior type of behavior.

Required arguments

dataSet A *DataSet* object.

Optional arguments

None.

Return value

A tuple consisting of a and b values of the regression line(y = ax + b), coefficient of determination(r-squared) value and the start and end-points of the line.

Exceptions

None.

`compute_nu(...)`

This method computes the value of Poisson's Ratio from the existing DataSet object. The method is only valid for ElasIsoBehavior and ElasPlasIsoBehavior type of behavior.

Required arguments

dataSet A *DataSet* object.

Optional arguments

None.

Return value

A tuple consisting of a and b values of the regression line(y = ax + b), coefficient of determination(r-squared) value and the start and end-points of the line.

Exceptions

None.

`compute_ultimatePoint(...)`

This method computes the coordinates of the Ultimate point from the existing DataSet object. The method is only valid for ElasPlasIsoBehavior type of behavior.

Required arguments

dataSet A *DataSet* object.

Optional arguments

None.

Return value

Coordinates of the ultimate point.

Exceptions

None.

`compute_elasticModulus(...)`

This method computes the value of the elastic modulus from the yieldpoint value. The method is only valid for ElasPlasIsoBehavior type of behavior.

Required arguments

yieldPoint A tuple consisting of coordinates of the yieldpoint.

Optional arguments

None.

Return value

A float specifying the value of elastic modulus.

Exceptions

None.

`compute_plasticPoints(...)`

This method extracts the coordinates of the plastic Points. The method is only valid for ElasPlasIsoBehavior type of behavior.

Required arguments

dataSet A *DataSet* object.

slider_val A float specifying the number of values to be taken.

start_index A float specifying the lower limit of the range.

end_index A float specifying the upper limit of the range.

Optional arguments

yp Coordinates of the yieldpoint. The default value is (0,0).

Return value

A sequence of coordinates of the plastic points.

Exceptions

None.

`xyDataDissect(...)`

This method extracts primary, unload, reload and permanent DataSet objects from the existing DataSet object.The method is only valid for FeFpBehavior type of behavior.

Required arguments

dsName A string specifying the name of the uniaxial/biaxial test dataset.

modelName A string specifying the name of the model to which the calibration behavior belongs.

calibrationName A string specifying the name of the Calibration object to which the behavior belongs.

Optional arguments

biaxial A boolean specifying whether the test data is biaxial or uniaxial. The default value is True.

Return value

A sequence of strings specifying names of the DataSet objects containing loading, unloading, reloading and primary datasets.

Exceptions

None.

Members

The Behavior object has members with the same names and descriptions as the arguments to the *Behavior* method and *setValues* method.

In addition, the Behavior object can have the following members:

modelName A string specifying the name of the model to which the behavior belongs.

calibrationName A string specifying the name of calibration to which the behavior belongs.

biAxialAllName

A String specifying the name of the dataset containing all the raw data in the test data file. Only valid if the behavior is of type FeFpBehavior

uniAxialAllName

A String specifying the name of the dataset containing all the raw data in the test data file. Only valid if the behavior is of type FeFpBehavior

DataSet object

The DataSetobject specifies material test data.

Access

```
import calibration
mdb.models[name].calibrations[name].dataSets[name]
```

DataSet(...)

This method creates a DataSet object.

Path

```
mdb.models[name].calibrations[name].DataSet
```

Required arguments

name A String specifying the name of the new dataset.

Optional arguments

data A sequence of pairs of Floats specifying data set type pairs. Possible values are for stress/strain, force/displacement, or transverse strain/axial strain pairs.

type A String specifying the type of the new dataset. Values can be "STRESS/STRAIN", "FORCE/DISPLACEMENT", or "AXIALSTRAIN/TRANSVERSESTRAIN". The default value is "STRESS/STRAIN".

form A String specifying the form of the new dataset. Values can be "NOMINAL" or "TRUE". The default value is "NOMINAl".

Return value

A DataSet object.

Exceptions

InvalidNameError.

setValues(...)

This method modifies the data for an existing DataSet object.

Required arguments

None.

Optional arguments

data A sequence of pairs of Floats specifying data set type pairs.

Return value

`None.`

Exceptions

None.

Members

The DataSet object has members with the same names and descriptions as the arguments to the *DataSet* method.

Canvas commands

Canvas commands are used to create, position, and modify canvas objects.
The Canvas commands need only the `abaqus` module to be imported; no additional modules need to be imported.

In this section:

- *AttributeColorMap object*
- *DrawingArea object*
- *Layer object*
- *Viewport object*
- *ImageOptions object*
- *MovieOptions object*
- *Highlight commands*

AttributeColorMap object

The AttributeColorMap object is used to store values and attributes associated with AttributeColorMap type objects. AttributeColorMap objects can be modified using the methods described below. The methods accessed via the *Viewport* object cause the AttributeColorMap object to be updated in the session.viewports[name].colorMappings repository.

Access

```
session.viewports[name].colorMappings[name]
```

setDefaults()

This method resets the AttributeColorMap object to its default state.

Arguments

None.

Return value

None.

Exceptions

None.

setValues(...)

This method modifies the AttributeColorMap object.

Required arguments

None.

Optional arguments

At least one of the following must be provided:

overrides

A Dictionary object specifying a color mapping. Each key is of String type and specifies an attribute in the map; the corresponding values specify the color definition to apply to that attribute in the form (0|1, wire color, edge color, face color). The 0|1 defines the active status for the attribute. For example:

```
overrides={
                          'Part-1':(1,'#00FF00', '#00CCFF',
                          '#00FF00')}
```

defaultOverrides

A Dictionary object specifying a custom color mapping similar to overrides. For example:

```
defaultOverrides={
                          'Copper':(1,''#00FF00', '#00CCFF',
                          '#00FF00')}
```

The color mapping can contain keys that have not been created. When the key is created, it gets the appropriate values from this mapping.

Return value
`None.`

Exceptions
None.

updateOverrides(...)

This method specifies additional overrides to be added to the current object definition.

Required arguments
None.

Optional arguments
At least one of the following must be provided:

overrides A Dictionary object specifying a color mapping. Each key is of String type and specifies an attribute in the map; the corresponding values specify the color definition to apply to that attribute in the form (0|1, wire color, edge color, face color). The 0|1 defines the active status for the attribute. For example:

```
overrides={
                          'Part-1':(1,'#00FF00', '#00CCFF',
                          '#00FF00')}
```

defaultOverrides A Dictionary object specifying a custom color mapping similar to overrides. For example:

```
defaultOverrides={
                          'Copper':(1,''#00FF00', '#00CCFF',
                          '#00FF00')}
```

The color mapping can contain keys that have not been created. When the key is created, it gets the appropriate values from this mapping.

Return value
`None.`

Exceptions
None.

Members

The AttributeColorMap object has the following members:

mapType A SymbolicConstant specifying the type of AttributeColorMap . Possible values are MATERIAL_MAP, SECTION_MAP, PART_MAP, ELSET_MAP, AVERAGING_REGION_MAP, and ELTYPE_MAP.

overrides A Dictionary object specifying a color mapping. Each key is of String type and specifies an attribute in the map; the corresponding values specify the color definition to apply to that attribute

in the form (0|1, wire color, edge color, face color). The 0|1 defines the active status for the attribute. For example:

```
overrides={
                          'Part-1':(1,'#00FF00', '#00CCFF',
                          '#00FF00')}
```

defaultOverrides

A Dictionary object specifying a custom color mapping similar to overrides. For example:

```
defaultOverrides={
                          'Copper':(1,''#00FF00', '#00CCFF',
                          '#00FF00')}
```

The color mapping can contain keys that have not been created. When the key is created, it gets the appropriate values from this mapping.

attributeColors

A Dictionary object specifying the color settings of each attribute as described in the *updateOverrides* method.

DrawingArea object

The DrawingArea object specifies the location and size of the drawing area used for placement of viewports.

Access

```
session.drawingArea
```

Members

The DrawingArea object has the following members:

width	A Float specifying the width in millimeters.
height	A Float specifying the height in millimeters.
origin	A pair of Floats specifying the coordinates of the bottom left hand corner in millimeters.

Layer object

Objects can be superimposed by displaying them in different layers of a viewport.

Access

```
session.viewports[name].layers[name]
```

Layer(...)

This method creates a Layer object in the Layer repository.

Path

```
session.viewports[name].Layer
```

Required arguments

name A String specifying the repository key.

Optional arguments

copyViewName A String specifying the name of the layer to copy.

Return value

A Layer object.

Exceptions

None.

moveBefore(...)

This method moves the layer object before another object in the layer repository.

Required arguments

name A String specifying the name of the other Layer object.

Optional arguments

None.

Return value

```
None.
```

Exceptions

None.

moveAfter(...)

This method moves the layer object after another object in the layer repository.

Required arguments

name A String specifying the name of the other Layer object.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Layer object has members with the same names and descriptions as the arguments to the *Layer* method.

In addition, the Layer object can have the following members:

displayedObject A Displayable object specifying the object to be displayed. The Displayable type is an abstract generalization. The concrete possible types are *Part*, *Assembly*, *ConstrainedSketch*, *Odb*, or *XYPlot*.

view A *View* object specifying the object that controls viewing of the layer.

odbDisplay An *OdbDisplay* object specifying the display options for the *Odb* object.

partDisplay A *PartDisplayOptions* object specifying the display options for the *Part* object.

assemblyDisplay An *AssemblyDisplayOptions* object specifying the display options for the *Assembly* object.

Viewport object

A viewport is the container for the graphics generated by the application. TheViewport object stores the various settings that determine how objects are displayed within that viewport.

Access

```
session.viewports[name]
```

Viewport(...)

This method creates a Viewport object with the specified origin and dimensions.

Path

```
session.Viewport
```

Required arguments

name A String specifying the repository key.

Optional arguments

origin A pair of Floats specifying the *X*- and *Y*-coordinates in millimeters in the canvas coordinate system of the lower left corner of the viewport. The default origin is (0, 0).

width A Float specifying the width in millimeters of the viewport. Possible values are $30 \leq$ *width* $\leq$ (*maxWidth*). The default value is 120.0.



Note:

The maximum value of width (*maxWidth*) is the width of the screen in millimeters.

height A Float specifying the height in millimeters of the viewport. This height includes the title bar. Possible values are $30 \leq$ *height* $\leq$ (*maxHeight*). The default value is 80.0.



Note:

The maximum value of height (*maxHeight*) is the height of the screen in millimeters.

border A Boolean specifying whether the viewport border is visible in a printed image. The default value is ON.

titleBar A Boolean specifying whether the viewport title should be displayed in a printed image. The default value is ON.

If *border* = OFF, the title will not be visible, even if *titleBar* =ON.

titleStyle A SymbolicConstant specifying which title to use for the viewport title. Possible values are CUSTOM and SYSTEM. The default value is SYSTEM.

If *titleStyle* = CUSTOM, *customTitleString* will be used. If *titleStyle*= SYSTEM, a system-generated string will be used.

customTitleString A String specifying the viewport title when *titleStyle* =CUSTOM. The default value is an empty string.

Return value

A Viewport object.

Exceptions

RangeError.

If the user attempts to delete the only viewport:	SystemError: the current viewport may not be deleted.
If *width* is out of range:	RangeError: width must be a Float in the range: 30 <= width <= *maxWidth*
If *height* is out of range:	RangeError: height must be a Float in the range: 30 <= width <= *maxHeight*

`bringToFront()`

This method moves the Viewport object to the front.

Arguments

None.

Return value

`None.`

Exceptions

None.

`disableMultipleColors()`

This method disables applying multiple color mappings that was enabled using `enableMultipleColors`

Arguments

None.

Return value

`None.`

Exceptions

None.

disableRefresh()

This method disables Viewport refresh. Some methods that require the Viewport to be up-to-date will override this setting. It is advisable to use this method sparingly.

Arguments

None.

Return value

`None.`

Exceptions

None.

disableColorCodeUpdates()

This method disables Viewport updates and internal computations triggered because of color coding. Performance improvement will be significant when color coding is ON and repeating operations are performed using a script each of which requires color code updates. No benefit will be had when color coding is OFF.

Arguments

None.

Return value

`None.`

Exceptions

None.

enableMultipleColors()

This method enables multiple color mappings to be applied at the same time. It also ensures that the Viewport is updated correctly when *initialColor* is set.

Arguments

None.

Return value

`None.`

Exceptions

None.

enableRefresh()

This method enables Viewport refresh disabled using `disableRefresh`.

Arguments

None.

Return value

`None.`

Exceptions

None.

enableColorCodeUpdates()

This method enables Viewport color code updates disabled using `disableColorCodeUpdates`.

Arguments

None.

Return value

`None.`

Exceptions

None.

getActiveElementLabels(...)

This method returns the element labels currently active in the viewport based on the current display group. The element labels are printed only when the *displayedObject* member in the Viewport object is set to an Odb. The `getActiveElementLabels` method has the following arguments:

Required arguments

None.

Optional arguments

useCut A Boolean flag to specify if any active cutting plane is to be considered in determining active elements.

printResults A Boolean flag to specify if the active element labels are to be printed to the replay file.

Return value

A Dictionary object of element labels, keyed by OdbInstance name. Returns `None` if the *displayedObject* member is not an Odb object.

Exceptions

None.

getActiveNodeLabels(...)

This method returns the node labels currently active in the viewport based on the current display group. The node labels are printed only when the *displayedObject* member in the Viewport object is set to an Odb. The `getActiveNodeLabels` method has the following arguments:

Required arguments

None.

Optional arguments

useCut A Boolean flag to specify if any active cutting plane is to be considered in determining active nodes.

printResults A Boolean flag to specify if the active node labels are to be printed to the replay file.

Return value

A Dictionary object of node labels, keyed by OdbInstance name. Returns `None` if the *displayedObject* member is not an Odb object.

Exceptions

None.

getPrimVarMinMaxLoc()

This method returns a dictionary containing the minimum, maximum and their location for the current primary variable. A contour plot should be displayed in the current viewport or else the method will return *None*.

Arguments

None.

Return value

A dictionary with keys 'minPartInstanceName', 'minElementLabel', 'minNodeLabel', 'minPosition', 'maxPartInstanceName', 'maxElementLabel', 'maxNodeLabel', 'maxPosition'

Exceptions

None.

makeCurrent()

This method makes theViewport object the current viewport.

Arguments

None.

Return value
`None.`

Exceptions

None.

`maximize()`

This method maximizes the Viewport object to fill the drawing area.

Arguments

None.

Return value
`None.`

Exceptions

None.

`minimize()`

This method minimizes the Viewport object to appear as an abbreviated title bar.

Arguments

None.

Return value
`None.`

Exceptions

None.

`offset(...)`

This method modifies the current *X*-*Y* location of the viewport by the specified distance.

Required arguments

None.

Optional arguments

deltaX
A Float specifying the offset in millimeters of the *X*-component of the viewport origin. The default value is 0.

deltaY
A Float specifying the offset in millimeters of the *Y*-component of the viewport origin. The default value is 0.

Return value
`None.`

Exceptions
None.

restore()

This method restores a maximized or minimized Viewport object to its previous size and location.

Arguments
None.

Return value
`None.`

Exceptions
None.

sendToBack()

This method moves the Viewport object to the back.

Arguments
None.

Return value
`None.`

Exceptions
None.

setColor(...)

This method specifies the color assignment using *initialColor* and *translucency*. If *initialColor* has a value of 'As is', *translucency* has no effect. The `setColor` method has the following arguments:

Required arguments

initialColor A string specifying the initial color applied to the objects.

Optional arguments

translucency A float in the range of 0.0 to 1.0 specifying how translucent the objects drawn using *initialColor* needs to be.

Return value
`None.`

Exceptions
None.

`setColor(...)`

This method specifies the color assignment using attributes specified by an *AttributeColorMap* object. The `setColor` method has the following arguments:

Required arguments

colorMapping An *AttributeColorMap* object. Possible values are any AttributeColorMap object.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

`setColor(...)`

This method specifies the color of a *Leaf* object.

Required arguments

leaf A *Leaf* object. Possible values are any Leaf object.

Optional arguments
At least one of the following must be provided:

edgeColorWireHide A String specifying the color to be used for drawing the edges of the elements contained in *leaf* when the render style is wireframe or hidden.

edgeColorFillShade A String specifying the color to be used for drawing the edges of the elements contained in *leaf* when the render style is filled or shaded.

fillColor A String specifying the color to be used for drawing the faces of the elements contained in *leaf* when the render style is filled or shaded.

nodeSymbolColor A String specifying the color to be used for drawing the nodes contained in *leaf*.

nodeSymbolType

A SymbolicConstant specifying the node symbol types for the nodes contained in *leaf*. Possible values areFILLED_CIRCLE, FILLED_SQUARE, FILLED_DIAMOND, FILLED_TRI, HOLLOW_CIRCLE, HOLLOW_SQUARE, HOLLOW_DIAMOND, HOLLOW_TRI, CROSS, and XMARKER. The default value is HOLLOW_CIRCLE.

nodeSymbolSize

A SymbolicConstant specifying the node symbol size for the nodes contained in *leaf*. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

Return value

`None.`

Exceptions

None.

`forceRefresh()`

This method causes the Viewport to refresh immediately. It is provided to allow scripts to refresh the Viewport before the script terminates. Normally, there would only be a single cumulative refresh that takes place immediately after the script completes.

Arguments

None.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the Viewport object. The arguments to `setValues` are the same as the arguments to the *`Viewport`* method, except for the *name* argument. In addition, the `setValues` method has the following arguments:

Required arguments

None.

Optional arguments

displayedObject

A Displayable object specifying the object to be displayed. The Displayable type is an abstract generalization. The concrete possible types are *Part*, *Assembly*, *ConstrainedSketch*, *Odb*, *PlyStackPlot*, or *XYPlot*. If *displayedObject*=`None`, Abaqus displays an empty viewport.

displayMode

A SymbolicConstant specifying the display mode of the viewport. Possible values are:

- SINGLE, specifying a single *displayedObject*.

- OVERLAY, specifying one or more layers to be displayed simultaneously—each layer contains one *displayedObject*.

visibleLayers
A sequence of Strings specifying the names of layers that will be displayed in the viewport when *displayMode* = OVERLAY.

viewManipLayers
A SymbolicConstant specifying which layer or layers will be controlled by the view manipulation tools when *displayMode*=OVERLAY. Possible values are ALL and CURRENT.

currentLayer
A String specifying which layer is affected by options settings when *displayMode* =OVERLAY. The current layer is also the only layer affected by view manipulations when*viewManipLayers* =CURRENT.

layerOffset
A Float specifying a factor to be used in offsetting layers in the screen Z direction. Possible values are –1 to 1. A negative value reverses the apparent order in which the layers are plotted.

Return value
`None.`

Exceptions
RangeError.

`addDrawings(...)`

This method identifies the names of *Drawing* objects to be rendered in the Viewport.

Required arguments
None.

Optional arguments
names
A sequence of String values identifying keys in the session.drawings repository.

Return value
`None.`

Exceptions
ValueError.

`removeDrawings(...)`

This method identifies the names of *Drawing* objects to no longer be rendered in the Viewport.

Required arguments
None.

Optional arguments

names A sequence of String values identifying keys in the Viewport sequence.

Return value

`None.`

Exceptions

ValueError.

`timeDisplay(...)`

This method refreshes the Viewport display *numFrames* times and then checks to see if *numSeconds* seconds have elapsed. If not, it will continue refreshing the Viewport until the time has elapsed. At completion, the actual number of refreshes (frames) rendered and elapsed time will be reported along with the calculated frames-per-second (fps).

Required arguments

None.

Optional arguments

numFrames An Int specifying the minimum number of times to refresh the Viewport. The default value is 0.

numSeconds An Int specifying the minimum number of seconds to spend refreshing the Viewport. The default value is 10.

degreesPerFrame A Float specifying the number of degrees to rotate the model view about its Z axis before each refresh. The default value is 0.0.

Return value

`None.`

Exceptions

None.

Members

The Viewport object has members with the same names and descriptions as the arguments to the *Viewport* method.

In addition, the Viewport object can have the following members:

displayMode A SymbolicConstant specifying the display mode of the viewport. Possible values are:

- SINGLE, specifying a single *displayedObject*.
- OVERLAY, specifying one or more layers to be displayed simultaneously—each layer contains one *displayedObject*.

viewManipLayers

A SymbolicConstant specifying which layer or layers will be controlled by the view manipulation tools when *displayMode*=OVERLAY. Possible values are ALL and CURRENT.

layerOffset

A Float specifying a factor to be used in offsetting layers in the screen Z direction. Possible values are –1 to 1. A negative value reverses the apparent order in which the layers are plotted.

windowState

A SymbolicConstant specifying the current state of a viewport. Possible values are NORMAL, MAXIMIZED, and MINIMIZED.

currentWidth

A Float specifying the width in millimeters of the current viewport, regardless of the value of *windowState*.

currentHeight

A Float specifying the height in millimeters of the current viewport, regardless of the value of *windowState*, and including the title bar.

applyLinkedCommands

A Boolean specifying whether the viewport is linked for synchronization. The default value is ON.

activeColorModes

A SymbolicConstant specifying the currently active color mappings. Possible values are:

- DEFAULT_COLORS
- PART_GEOM_MAP_COLORS
- ASSEMBLY_MAP_COLORS
- PART_MAP_COLORS
- INSTANCE_MAP_COLORS
- INSTANCE_TYPE_MAP_COLORS
- SECTION_MAP_COLORS
- MATERIAL_MAP_COLORS
- LOAD_MAP_COLORS
- BC_MAP_COLORS
- INTERACTION_MAP_COLORS
- CONSTRAINT_MAP_COLORS
- SET_MAP_COLORS
- SURFACE_MAP_COLORS
- INTERNAL_SET_MAP_COLORS
- INTERNAL_SURFACE_MAP_COLORS
- DISPLAY_GRP_MAP_COLORS
- SELECTION_GRP_MAP_COLORS
- ELTYPE_MAP_COLORS
- PLOT_MAP_COLORS
- MESH_MAP_COLORS

The default value is DEFAULT_COLORS.

colorMode

A SymbolicConstant specifying the last applied color mapping. Possible values are:

- DEFAULT_COLORS
- PART_GEOM_MAP_COLORS
- ASSEMBLY_MAP_COLORS
- PART_MAP_COLORS
- INSTANCE_MAP_COLORS
- INSTANCE_TYPE_MAP_COLORS
- SECTION_MAP_COLORS
- MATERIAL_MAP_COLORS
- LOAD_MAP_COLORS
- BC_MAP_COLORS
- INTERACTION_MAP_COLORS
- CONSTRAINT_MAP_COLORS
- SET_MAP_COLORS
- SURFACE_MAP_COLORS
- INTERNAL_SET_MAP_COLORS
- INTERNAL_SURFACE_MAP_COLORS
- DISPLAY_GRP_MAP_COLORS
- SELECTION_GRP_MAP_COLORS
- ELTYPE_MAP_COLORS
- PLOT_MAP_COLORS
- MESH_MAP_COLORS

The default value is DEFAULT_COLORS.

translucency

A Float specifying the translucency that will be applied to objects colored using *initialColor* and it needs to be set along with *initialColor*. If *initialColor* is set to 'As is' then translucency will have no effect.

animationConnect

A Boolean specifying whether an animation is connected to the viewport.

colorMappings

A repository of *AttributeColorMap* objects specifying the objects cannot be constructed but the following attribute maps are supported:

- "type"
- "Element set"
- "Material"
- "Section"
- "Default"
- "Part"

- "Part instance"
- "Element type"
- "Averaging region"
- "Assembly"
- "Property"
- "Set"
- "Surface"
- "Skin"
- "Profile"
- "Part shape"
- "Part status"
- "Part geometry"
- "Meshability"
- "Instance type"
- "Load"
- "Boundary condition"
- "Interaction"
- "Constraint"
- "Interaction type"
- "Constraint type"
- "Display group"
- "Selection group"
- "Interaction property"
- "Connector"
- "Connector type"
- "Connector property"
- "Internal set"
- "Internal surface"
- "mapColors"
- "autoColors"
- "overrides"
- "defaultAutoColors"
- "defaultOverrides"
- "objectToCopy"
- "colorMapping"
- "colorMappings"
- "colorMode"
- "attributeColors"
- "updateOverrides"

- "colorCodeOverride"
- "initialColor"
- "Layup"
- "Ply"

initialColor
A String specifying the color that will be applied to all objects in the viewport at the start of color coding. The possible values are 'As is', 'Default' or a string with a hexadecimal representation of a color.

currentLayer
A String specifying which layer is affected by options settings when *displayMode* =OVERLAY. The current layer is also the only layer affected by view manipulations when*viewManipLayers* =CURRENT.

displayedObject
A Displayable object specifying the object to be displayed. The Displayable type is an abstract generalization. The concrete possible types are *Part*, *Assembly*, *ConstrainedSketch*, *Odb*, *PlyStackPlot*, or *XYPlot*. If *displayedObject*=`None`, Abaqus displays an empty viewport.

layers
A repository of *Layer* objects specifying the key to the repository is a String with the name of the layer.

view
A *View* object specifying the object that controls viewing of the viewport content.

odbDisplay
An *OdbDisplay* object specifying the display options for the *Odb* object.

partDisplay
A *PartDisplayOptions* object specifying the display options for the *Part* object.

assemblyDisplay
An *AssemblyDisplayOptions* object specifying the display options for the *Assembly* object.

viewportAnnotationOptions
A *ViewportAnnotationOptions* object.

detailPlotOptions
A *DetailPlotOptions* object.

annotationsToPlot
An *AnnotationsToPlotArray* object.

visibleLayers
A tuple of Strings specifying the names of layers that will be displayed in the viewport when *displayMode* = OVERLAY.

currentOrigin
A pair of Floats specifying the *X*- and *Y*-coordinates in millimeters in the canvas coordinate system of the lower left corner of the current viewport, regardless of the value of *windowState*.

iconOrigin
A pair of Floats specifying the *X*- and *Y*-coordinates in millimeters of the lower left corner of the current viewport from a coordinate system having its origin in the lower left corner of the drawing area. This origin refers to the viewport location when *windowState* =MINIMIZED.

lightOptions A *LightOptions* object.

imageOptions An *ImageOptions* object.

movieOptions A *MovieOptions* object.

animationController An *AnimationController* object.

drawings A tuple of Strings specifying keys to the session.drawings repository. The default value is an empty sequence.

ImageOptions object

The ImageOptions object stores settings that control how an image is rendered in a particular viewport. ImageOptions objects are accessed from the image options associated with a particular viewport.

The ImageOptions object has no constructor; Abaqus creates the *imageOptions* member for a viewport when the viewport is created using the values in the *imageOptions* member of the current viewport.

Access

```
session.viewports[name].imageOptions
```

`setValues(...)`

This method modifies the ImageOptions object.

Required arguments

None.

Optional arguments

imageName

A String specifying the name of the image. A list of valid image names is in the *images* repository in the *session* object.

showImage

A Boolean specifying whether an image should be displayed in the viewport background. The default value is OFF.

positionMethod

A SymbolicConstant specifying which positioning method is used to determine how the image will be scaled and positioned in the viewport. Possible values are:

- FIT_TO_VIEWPORT, specifying a display mode where the image is scaled to fit in the viewport using the specified *fitMethod*.
- AUTO_ALIGN, specifying a display mode where the image is scaled as specified by *xScale* and *yScale* and then positioned in the viewport using the specified *alignment*.
- MANUAL, specifying a display mode where the image is scaled as specified by *xScale* and *yScale* and then positioned in the viewport using the specified *origin*.

The default value is FIT_TO_VIEWPORT.

fitMethod

A SymbolicConstant specifying which type of fit is performed to scale and position the image in the viewport when *positionMethod* =FIT_TO_VIEWPORT. Possible values are:

- BEST_FIT, specifying a mode where the image is scaled to completely fit within the viewport.
- FIT_WIDTH, specifying a mode where the image width is scaled to match the viewport width.
- FIT_HEIGHT, specifying a mode where the image height is scaled to match the viewport height.

The default value is BEST_FIT.

alignment

A SymbolicConstant specifying the relative position of the image in the viewport when *positionMethod* =AUTO_ALIGN. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is CENTER.

xScale

A Float specifying the scale applied to the image width. The *xScale* argument is ignored when *positionMethod* =FIT_TO_VIEWPORT. The default value is 1.0.

When *xScale* is negative, the image is mirrored about its y-axis but its position is not affected.

yScale

A Float specifying the scale applied to the image height. The *yScale* argument is ignored when *positionMethod* =FIT_TO_VIEWPORT. The default value is 1.0.

When *yScale* is negative, the image is mirrored about its x-axis but its position is not affected.

origin

A pair of Floats specifying the *X*- and *Y*-offsets in millimeters from the lower-left corner of the viewport. The *origin* argument is ignored unless *positionMethod* =MANUAL. The default value is (0, 0).

translucency

A Float specifying the translucency factor to use when displaying the image. Possible values are $0.0 \leq$ *translucency* ≤ 1.0 with 0.0 being invisible and 1.0 being opaque. The default value is 1.0.

options

`None` or an ImageOptions object specifying the object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in the *options* member. The default value is `None`.

Return value

`None`.

Exceptions

RangeError.

Members

The ImageOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

MovieOptions object

The MovieOptions object stores settings that control how the movie background for an animation is rendered in a particular viewport. MovieOptions objects are accessed from the movie options associated with a particular viewport.

The MovieOptions object has no constructor; Abaqus creates the *movieOptions* member for a viewport when the viewport is created using the values in the *movieOptions* member of the current viewport.

Access

```
session.viewports[name].movieOptions
```

`setValues(...)`

This method modifies the MovieOptions object.

Required arguments

None.

Optional arguments

movieName

A String specifying the name of the movie. A list of valid movie names is in the *movies* repository in the *session* object.

showMovie

A Boolean specifying whether a movie should be displayed in the viewport during an animation. The default value is OFF.

positionMethod

A SymbolicConstant specifying which positioning method is used to determine how the movie frames will be scaled and positioned in the viewport. Possible values are:

- FIT_TO_VIEWPORT, specifying a display mode where the movie frame is scaled to fit in the viewport using the specified *fitMethod*.
- AUTO_ALIGN, specifying a display mode where the movie frame is scaled as specified by *xScale* and *yScale* and then positioned in the viewport using the specified *alignment*.
- MANUAL, specifying a display mode where the movie frame is scaled as specified by *xScale* and *yScale* and then positioned in the viewport using the specified *origin*.

The default value is FIT_TO_VIEWPORT.

fitMethod

A SymbolicConstant specifying which type of fit is performed to scale and position the movie frame in the viewport when *positionMethod* =FIT_TO_VIEWPORT. Possible values are:

- BEST_FIT, specifying a mode where the movie frame is scaled to completely fit within the viewport.
- FIT_WIDTH, specifying a mode where the movie frame width is scaled to match the viewport width.
- FIT_HEIGHT, specifying a mode where the movie frame height is scaled to match the viewport height.

The default value is BEST_FIT.

alignment A SymbolicConstant specifying the relative position of the movie frame in the viewport when *positionMethod* =AUTO_ALIGN. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is CENTER.

xScale A Float specifying the scale applied to the movie frame width. The *xScale* argument is ignored when *positionMethod* =FIT_TO_VIEWPORT. The default value is 1.0.

When *xScale* is negative, the movie frame is mirrored about its y-axis but its position is not affected.

yScale A Float specifying the scale applied to the movie frame height. The *yScale* argument is ignored when *positionMethod* =FIT_TO_VIEWPORT. The default value is 1.0.

When *yScale* is negative, the movie frame is mirrored about its x-axis but its position is not affected.

origin A pair of Floats specifying the*X*- and *Y*-offsets in millimeters from the lower-left corner of the viewport. The *origin* argument is ignored unless *positionMethod* =MANUAL. The default value is (0, 0).

translucency A Float specifying the translucency factor to use when displaying the movie frame. Possible values are $0.0 \leq$ *translucency* ≤ 1.0 with 0.0 being invisible and 1.0 being opaque. The default value is 1.0.

options `None` or a MovieOptions object specifying the object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in the *options* member. The default value is `None`.

Return value

`None`.

Exceptions

RangeError.

Members

The MovieOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

Highlight commands

The Highlight commands are used to highlight objects in the current viewport and to remove the highlighting.

highlight

This method highlights an object in the current viewport.

Path

highlight

Required arguments

object An object specifying the object in the current viewport to be highlighted. You can specify only a single object. Abaqus/CAE highlights only the edges of a face when highlighting a surface and a face together. The following objects are supported:

For the MDB

- Vertex
- Edge
- Face
- Surface
- Cell
- Node
- Element
- Element face
- Element edge
- Feature
- Datum
- Instance
- Set
- Load
- Boundary condition
- Predefined field
- Display group

For the ODB

- Node
- Element
- Display group

unhighlight

This method removes highlighting from an object in the current viewport.

Path

unhighlight

Required arguments

object

An object specifying the object in the current viewport from which the highlighting will be removed. You can specify only a single object. See *highlight* for a list of supported objects.

Connector commands

A connector describes the relative motions between two points. A connector also describes the behavior associated with the relative motion.

In this section:

- *ConnectorBehaviorOption object*
- *CDCTerm object*
- *ConnectorDamage object*
- *ConnectorDamping object*
- *ConnectorElasticity object*
- *ConnectorFailure object*
- *ConnectorFriction object*
- *ConnectorLock object*
- *ConnectorOptions object*
- *ConnectorPlasticity object*
- *ConnectorPotential object*
- *ConnectorStop object*
- *DerivedComponent object*
- *TangentialBehavior object*

ConnectorBehaviorOption object

The ConnectorBehaviorOption object is the abstract base type for other ConnectorBehaviorOption objects. The ConnectorBehaviorOption object has no explicit constructor. The members of the ConnectorBehaviorOption object are common to all objects derived from the ConnectorBehaviorOption.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

Members

The ConnectorBehaviorOption object has no members.

CDCTerm object

The CDCTerm object is used to create contributing terms for a *DerivedComponent* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i]
mdb.models[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i]
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.cdcTerms[i]
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i]
session.odbs[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i]
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.cdcTerms[i]
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i]
```

CDCTerm(...)

This method creates a CDCTerm object.

Path

```
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.CDCTerm
mdb.models[name].sections[name].behaviorOptions[i].derivedComponent\
.CDCTerm
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.CDCTerm
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.CDCTerm
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.CDCTerm
session.odbs[name].sections[name].behaviorOptions[i].derivedComponent\
.CDCTerm
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.CDCTerm
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.CDCTerm
```

Required arguments

intrinsicComponents A sequence of Ints specifying the components of relative motion for which the contributing term is defined. Possible values are $1 \leq intrinsicComponents \leq 6$. Only available components can be specified if the DerivedComponent object is being referenced by a Potential object. This

is not the case if the DerivedComponent object is referenced by a ConnectorFriction object directly. The default value is an empty sequence.

table

A sequence of sequences of Floats specifying components numbers and temperature and field values. Each sequence of the table data specifies:

- The first intrinsic component number.
- If applicable, the second intrinsic component number.
- Etc.
- If applicable, the first independent component number.
- If applicable, the second independent component number.
- Etc.
- If applicable, the temperature value.
- If applicable, the value of the first field variable.
- If applicable, the value of the second field variable.
- Etc.

The default value is an empty sequence.

Optional arguments

termOperator

A SymbolicConstant specifying the method for combining contributing terms: square root of a sum of the squares, direct sum, or Macauley sum. Possible values are RSS, SUM, and MACAULEY. The default value is RSS.

termSign

A SymbolicConstant specifying the overall sign for the contributing term. Possible values are POSITIVE and NEGATIVE. The default value is POSITIVE.

localDependency

A Boolean specifying whether the table data depend on either components of relative position or components of constitutive relative motion. The default value is OFF.

indepCompType

A SymbolicConstant specifying whether localDependency refers to components of relative position or components of constitutive relative motion. Possible values are POSITION and MOTION. The default value is POSITION.

The *indepCompType* argument applies only if *localDependency*=ON.

indepComponents

A sequence of Ints specifying the independent components included in the derived component definition. Possible values are $1 \leq$*indepComponents*≤ 6. Only available components can be specified. The *indepComponents* argument applies only if *localDependency*=ON. The default value is an empty sequence.

tempDependency

A Boolean specifying whether the table data depend on temperature. The default value is OFF.

fieldDependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Return value

A CDCTerm object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the CDCTerm object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CDCTerm* method.

Return value

`None.`

Exceptions

ValueError.

Members

The CDCTerm object has members with the same names and descriptions as the arguments to the *CDCTerm* method.

In addition, the CDCTerm object can have the following member:

options A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

ConnectorDamage object

The ConnectorDamage object defines damage behavior for one or more components of a connector's relative motion.

The ConnectorDamage object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorDamage(...)

This method creates a connector damage behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorDamage
import odbConnectorBehavior
odbConnectorBehavior.ConnectorDamage
```

Required arguments

None.

Optional arguments

coupling

A SymbolicConstant specifying whether or not the behavior is coupled. Possible values are UNCOUPLED and COUPLED. The default value is UNCOUPLED.

criterion

A SymbolicConstant specifying the damage initiation criterion to be used. Possible values are FORCE, MOTION, and PLASTIC_MOTION. The default value is FORCE.

initiationTemperature

A Boolean specifying whether the initiation data depend on temperature. The default value is OFF.

initiationPotentialOperator

A SymbolicConstant specifying the contribution operator for the initiation potential contributions. Possible values are SUM and MAXIMUM. The default value is SUM.

This argument is only if *coupling*=COUPLED and if *criterion*=FORCE or MOTION.

initiationPotentialExponent

A Float specifying the number equal to the inverse of the overall exponent in the initiation potential definition. The default value is 2.0.

This argument is applicable only if *coupling*=COUPLED, when *initiationPotentialOperator*=SUM, and when *criterion*=FORCE or MOTION.

initiationDependencies

An Int specifying the number of field variable dependencies for the initiation data. The default value is 0.

evolution

A Boolean specifying whether damage evolution data will be used. The default value is ON.

evolutionType

A SymbolicConstant specifying the type of damage evolution to be specified. Possible values are MOTION_TYPE and ENERGY_TYPE. The default value is MOTION_TYPE.

This argument is applicable only if *evolution*=ON.

softening

A SymbolicConstant specifying the damage evolution law to be specified. Possible values are LINEAR, EXPONENTIAL, and TABULAR. The default value is LINEAR.

This argument is applicable only if *evolution*=ON and when *evolutionType*=MOTION_TYPE.

useAffected

A Boolean specifying whether or not *affectedComponents* will be specified. If *useAffected*=OFF, then only the components of relative motion specified by *components* will undergo damage. The default value is OFF.

This argument is applicable only if *evolution*=ON.

degradation

A SymbolicConstant specifying the contribution of each damage mechanism when more than one damage mechanism is defined. Possible values are MAXIMUM and MULTIPLICATIVE. The default value is MAXIMUM.

This argument is applicable if *evolution*=ON.

evolutionTemperature

A Boolean specifying whether the evolution data depend on temperature. The default value is OFF.

This argument is applicable only if *evolution*=ON.

evolutionDependencies

An Int specifying the number of field variable dependencies for the evolution data. The default value is 0.

This argument is applicable only if *evolution*=ON.

evolutionPotentialOperator

A SymbolicConstant specifying the contribution operator for the evolution potential contributions. Possible values are SUM and MAXIMUM. The default value is SUM.

This argument is applicable only if *coupling*=COUPLED, when *evolution*=ON, when *evolutionType*=MOTION_TYPE, and when *criterion*=FORCE or MOTION.

evolutionPotentialExponent

A Float specifying the number equal to the inverse of the overall exponent in the evolution potential definition. The default value is 2.0.

This argument is applicable only if *coupling*=COUPLED, when *evolution*=ON, when *evolutionPotentialOperator*=SUM, when *evolutionType*=MOTION, and when *criterion*=FORCE or MOTION.

initiationPotentials

A *ConnectorPotentialArray* object specifying one *ConnectorPotential* object for each initiation potential contribution. This member can be specified only if *coupling*=COUPLED and if *criterion*=FORCE or MOTION.

evolutionPotentials

A *ConnectorPotentialArray* object specifying one *ConnectorPotential* object for each evolution potential contribution). This member can be specified only if *coupling*=COUPLED, if *evolution*=ON, if *evolutionType*=MOTION, and if *criterion*=FORCE or MOTION.

initiationTable

A sequence of sequences of Floats specifying the initiation properties. The default value is an empty sequence.

Items in the *initiationTable* data are described below.

evolutionTable

A sequence of sequences of Floats specifying the evolution properties. The default value is an empty sequence.

Items in the *evolutionTable* data are described below. This argument is only applicable if *evolution*=ON.

affectedComponents

A sequence of Ints specifying the components of relative motion that will be damaged. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument is applicable only if *evolution*=ON and *useAffected*=ON. The default value is an empty sequence.

components

A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument can be specified only if *coupling*=UNCOUPLED. The default value is an empty sequence.

Table data

Table data for *initiationTable*:

If *criterion*=FORCE, then each sequence of the table data specifies the following:

- Lower (compression) limiting force or moment. Use -1.0E+36 to indicate an unspecified lower limit.
- Upper (tension) limiting force or moment. Use 1.0E+36 to indicate an unspecified upper limit. At least one limit, lower or upper, must be specified.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *criterion*=MOTION, then each sequence of the table data specifies the following:

- Lower (compression) limiting connector constitutive relative displacement or rotation. Use -1.0E+36 to indicate an unspecified lower limit.
- Upper (tension) limiting connector constitutive relative displacement or rotation. Use 1.0E+36 to indicate an unspecified upper limit. At least one limit, lower or upper, must be specified.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *criterion*=PLASTIC_MOTION, then each sequence of the table data specifies the following:

- Relative equivalent plastic displacement/rotation at which damage will be initiated.
- Mode mix ratio (only if *coupling*=COUPLED).
- Relative equivalent plastic displacement/rotation rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Table data for *evolutionTable*:

If *evolutionType*=MOTION and *softening*=LINEAR, then each sequence of the table data specifies the following:

- Post-initiation equivalent relative plastic motion at ultimate failure if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation) at ultimate failure.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=MOTION and *softening*=EXPONENTIAL, then each sequence of the table data specifies the following:

- Post-initiation equivalent relative plastic motion at ultimate failure if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation) at ultimate failure.
- Exponential law parameter.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=MOTION and *softening*=TABULAR, then each sequence of the table data specifies the following:

- Damage variable (cannot be less than 0 or greater than 1).
- Post-initiation equivalent relative plastic motion if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation).
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=ENERGY, then each sequence of the table data specifies the following:

- Total energy dissipated by damage at ultimate failure.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorDamage object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorDamage object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorDamage* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorDamage object has members with the same names and descriptions as the arguments to the *ConnectorDamage* method.

In addition, the ConnectorDamage object can have the following members:

initiationOptions
A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the damage initiation table.

evolutionOptions
A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the damage evolution table.

Corresponding analysis keywords

- **CONNECTOR DAMAGE INITIATION*, **CONNECTOR DAMAGE EVOLUTION*, **CONNECTOR POTENTIAL*

ConnectorDamping object

The ConnectorDamping object defines damping behavior for one or more components of a connector's relative motion.

The ConnectorDamping object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorDamping(...)

This method creates a connector damping behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorDamping
import odbConnectorBehavior
odbConnectorBehavior.ConnectorDamping
```

Required arguments

None.

Optional arguments

type

A SymbolicConstant specifying if the damping type is viscous or structural. Possible values are VISCOUS and STRUCTURAL. The default value is VISCOUS.

behavior

A SymbolicConstant specifying if the damping behavior is linear or nonlinear. If *type*=VISCOUS, possible values are LINEAR and NONLINEAR. If *type*=STRUCTURAL, the possible value is LINEAR.

coupling

A SymbolicConstant specifying whether the damping behavior is coupled between the connector's components of relative motion. If *behavior*=LINEAR, possible values are UNCOUPLED and COUPLED. If *behavior*=NONLINEAR, possible values are UNCOUPLED, COUPLED_POSITION, and COUPLED_MOTION. Possible values are UNCOUPLED, COUPLED, COUPLED_POSITION, and COUPLED_MOTION. The default value is UNCOUPLED.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0. This argument is applicable only if *type*=VISCOUS.

temperatureDependency

A Boolean specifying whether the behavior data depend on temperature. The default value is OFF. This argument is applicable only if *type*=VISCOUS.

frequencyDependency A Boolean specifying whether the behavior data depend on frequency. This value is applicable only if *behavior*= LINEAR and *coupling*=UNCOUPLED. The default value is OFF.

table A sequence of sequences of Floats specifying damping properties. Items in the table data are described below. The default value is an empty sequence.

independentComponents A sequence of Ints specifying the list of independent components that are included in the definition of the connector damping data. This argument is applicable only if *behavior*=NONLINEAR and *coupling*=COUPLED_POSITION or COUPLED_MOTION. When this argument is applicable, at least one value must be specified. Only available components can be specified. The default value is an empty sequence.

components A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Table data

If *type*=VISCOUS, *behavior*=LINEAR, and *coupling*=UNCOUPLED, each sequence of the table data specifies the following:

- Damping coefficient (force or moment per relative velocity).
- Frequency (cycles/time), if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=VISCOUS, *behavior*=NONLINEAR, and *coupling*=UNCOUPLED, each sequence of the table data specifies the following:

- Force or moment.
- Relative displacement or rotation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=VISCOUS, *behavior*=LINEAR, and *coupling*=COUPLED, the table data specify the symmetric portion of the damping matrix for the specified components followed by any temperature data and then any field data. For example, if components 2, 3, and 5 are specified, the table portion of the command is as follows:

```
table=( (C22, C23, C25, C33, C35, C55,), )
```

The following items should be specified as comma-separated data:

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *type*=VISCOUS, *behavior*=NONLINEAR, and *coupling*=COUPLED_POSITION or COUPLED_MOTION, each sequence of the table data specifies the following:

- Force or moment for the directions in the *components* list.
- Relative velocity for the directions in the *components* list.
- Relative position or angle for the first *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the first *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Relative position or angle for the second *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the second *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Etc. up to the Nth *independentComponents* direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=STRUCTURAL, *behavior*=LINEAR, and *coupling*=UNCOUPLED, each sequence of the table data specifies the following:

- Damping coefficient (force or moment per relative velocity).
- Frequency (cycles/time), if the data depend on frequency.
- Etc.

If *type*=STRUCTURAL, *behavior*=LINEAR, and *coupling*=COUPLED, the table data specify the symmetric portion of the damping matrix for the specified components. For example, if components 2, 3, and 5 are specified, the table portion of the command is as follows:

```
table=( (S22, S23, S25, S33, S35, S55,), )
```

Return value

A ConnectorDamping object.

Exceptions

ValueError and TextError.

setValues(...)

This method modifies the ConnectorDamping object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorDamping* method.

Return value
`None.`

Exceptions
ValueError.

Members
The ConnectorDamping object has members with the same names and descriptions as the arguments to the *ConnectorDamping* method.

In addition, the ConnectorDamping object can have the following member:

options
A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords
- **CONNECTOR DAMPING*

ConnectorElasticity object

The ConnectorElasticity object defines elastic behavior for one or more components of a connector's relative motion.

The ConnectorElasticity object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorElasticity(...)

This method creates a connector elasticity behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorElasticity
import odbConnectorBehavior
odbConnectorBehavior.ConnectorElasticity
```

Required arguments

None.

Optional arguments

behavior

A SymbolicConstant specifying whether the elastic behavior is linear, nonlinear, or rigid. Possible values are LINEAR, NONLINEAR, and RIGID. The default value is LINEAR.

coupling

A SymbolicConstant specifying whether the elastic behavior is coupled between the connector's components of relative motion. If *behavior*=LINEAR, then possible values are UNCOUPLED and COUPLED. If *behavior*=NONLINEAR, then possible values are UNCOUPLED, COUPLED_POSITION, and COUPLED_MOTION. Possible values are UNCOUPLED, COUPLED, COUPLED_POSITION, and COUPLED_MOTION. The default value is UNCOUPLED.

This argument is not applicable if *behavior*=RIGID.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

This argument is not applicable if *behavior*=RIGID.

temperatureDependency

A Boolean specifying whether the behavior data depend on temperature. The default value is OFF.

This argument is not applicable if *behavior*=RIGID.

frequencyDependency A Boolean specifying whether the behavior data depend on frequency. This value is applicable only if *behavior*=LINEAR and *coupling*=UNCOUPLED. The default value is OFF.

This argument is not applicable if *behavior*=RIGID.

table A sequence of sequences of Floats specifying elasticity properties. Items in the table data are described below. This argument is not applicable if *behavior*=RIGID. The default value is an empty sequence.

independentComponents A sequence of Ints specifying the list of independent components that are included in the definition of the connector elasticity data. This argument is applicable only if *behavior*=NONLINEAR and *coupling*=COUPLED_POSITION or COUPLED_MOTION. If this argument is applicable, at least one value must be specified. Only available components can be specified. The default value is an empty sequence.

components A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Table data

If *behavior*=LINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Elastic stiffness (per each specified force/moment component).
- Frequency (cycles/time), if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=NONLINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Force or moment.
- Relative displacement or rotation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=LINEAR and *coupling*=COUPLED, the table data specify the symmetric portion of the elastic stiffness matrix for the specified components followed by any temperature data and then any field data. For example, if components 2, 3, and 5 are specified, the table portion of the command is as follows:

```
table=( (D22, D23, D25, D33, D35, D55,),
)
```

The following items should then be specified as comma-separated data:

- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=NONLINEAR and *coupling*=COUPLED_POSITION or COUPLED_MOTION, each sequence of the table data specifies the following:

- Force or moment for the directions in the *components* list.
- Relative position or angle for the first *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the first *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Relative position or angle for the second *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the second *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Etc. up to the Nth *independentComponents* direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorElasticity object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorElasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorElasticity* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorElasticity object has members with the same names and descriptions as the arguments to the *ConnectorElasticity* method.

In addition, the ConnectorElasticity object can have the following member:

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords

- **CONNECTOR ELASTICITY*

ConnectorFailure object

The ConnectorFailure object defines failure criteria for one or more components of a connector's relative motion.

The ConnectorFailure object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorFailure(...)

This method creates a connector failure behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorFailure
import odbConnectorBehavior
odbConnectorBehavior.ConnectorFailure
```

Required arguments

None.

Optional arguments

releaseComponent The SymbolicConstant ALL or an Int specifying the motion components that fail. If an Int is specified, only that motion component fails when the failure criteria are satisfied. If *releaseComponent*=ALL, all motion components fail. The default value is ALL.

minMotion `None` or a Float specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is `None`.

maxMotion `None` or a Float specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is `None`.

minForce `None` or a Float specifying the lower bound of the force or moment in the directions of the specified components at which locking occurs, or no lower bound. The default value is `None`.

maxForce `None` or a Float specifying the upper bound of the force or moment in the directions of the specified components at which locking occurs, or no upper bound. The default value is `None`.

components A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorFailure object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorFailure object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorFailure* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorFailure object has members with the same names and descriptions as the arguments to the *ConnectorFailure* method.

Corresponding analysis keywords

- **CONNECTOR FAILURE*

ConnectorFriction object

The ConnectorFriction object defines Coulomb-like or hysteretic friction behavior for one or more components of a connector's relative motion.

The ConnectorFriction object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorFriction(...)

This method creates a connector friction behavior option for a *ConnectorSection* object. Depending upon the arguments provided, the friction behavior can be Coulomb-like or hysteretic in nature.

Path

```
import connectorBehavior
connectorBehavior.ConnectorFriction
import odbConnectorBehavior
odbConnectorBehavior.ConnectorFriction
```

Required arguments

None.

Optional arguments

frictionModel

A SymbolicConstant specifying the desired frictional response model. Possible values are PREDEFINED and USER_CUSTOMIZED. The default value is PREDEFINED.

slipStyle

A SymbolicConstant specifying the method of indicating the slip direction: either specified or computed based upon the force potential data. Possible values are SPECIFY and COMPUTE. The default value is SPECIFY.

This argument is applicable only if *frictionModel*=USER_CUSTOMIZED.

tangentDirection

`None` or an Int specifying the direction for which the frictional behavior is specified. Possible values are $1 \leq$*tangentDirection*≤ 6, indicating an available component of relative motion. This argument applies only if *frictionModel*=USER_CUSTOMIZED and if *slipStyle*=SPECIFY. The default value is `None`.

stickStiffness

`None` or a Float specifying the stick stiffness associated with the frictional behavior in the direction specified by *tangentDirection*. If this argument is omitted, Abaqus computes an appropriate number for the stick stiffness. The default value is `None`.

componentType

A SymbolicConstant specifying the type of the *independentComponents*. Possible values are POSITION, MOTION, and NO_INDEPENDENT_COMPONENTS. The default value is NO_INDEPENDENT_COMPONENTS.

slipDependency

A Boolean specifying whether the table data depend on accumulated slip. The default value is OFF.

This argument applies only if *frictionModel*=USER_CUSTOMIZED.

temperatureDependency

A Boolean specifying whether the table data depend on temperature. The default value is OFF.

This argument applies only if *frictionModel*=USER_CUSTOMIZED.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

This argument applies only if *frictionModel*=USER_CUSTOMIZED.

useContactForceComponent

A Boolean specifying whether the contact force component will be defined. The default value is OFF.

This argument applies only if *frictionModel*=USER_CUSTOMIZED.

contactForceStyle

A SymbolicConstant specifying the method of indicating the contact force component direction: either specified or computed based on upon a DerivedComponent. Possible values are COMPONENT_NUMBER and DERIVED_COMPONENT. The default value is COMPONENT_NUMBER.

This argument is applicable only if *frictionModel*=USER_CUSTOMIZED and if *useContactForceComponent*=ON.

contactForceComponent

An Int specifying the contact force component direction. This argument applies only if *frictionModel*=USER_CUSTOMIZED, if *useContactForceComponent*=ON, and if *contactForceStyle*=COMPONENT_NUMBER. The default value is 0.

forcePotentialOperator

A SymbolicConstant specifying the contribution operator for the force potential contributions. Possible values are SUM and MAXIMUM. The default value is SUM.

This argument is applicable only if *frictionModel*=USER_CUSTOMIZED and if *slipStyle*=COMPUTE.

forcePotentialExponent

A Float specifying the number equal to the inverse of the overall exponent in the force potential definition. The default value is 2.0.

This argument is applicable only if *frictionModel*=USER_CUSTOMIZED, if *slipStyle*=COMPUTE, and if *forcePotentialOperator*=SUM.

connectorPotentials

A *ConnectorPotentialArray* object specifying one *ConnectorPotential* object for each force potential contribution. This member can be specified only if *frictionModel*=USER_CUSTOMIZED, and if *slipStyle*=COMPUTE.

table

A sequence of sequences of Floats specifying friction properties. The default value is an empty sequence.

If *frictionModel*=PREDEFINED, each sequence of the table data specifies:

- If applicable, the first geometric scaling constant relevant to frictional interactions.
- Etc., up to as many geometric scaling constants as are associated with this connection type.
- Internal contact force/moment generating friction in the first predefined slip direction.
- If applicable, internal contact force/moment generating friction in the second predefined slip direction.
- Connector constitutive relative motion in the direction specified by *independentComponent*.
- Accumulated slip in the first predefined slip direction, if the data depend on accumulated slip.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *frictionModel*=USER_CUSTOMIZED, each sequence of the table data specifies:

- Effective radius of the cylindrical or spherical surface over which frictional slip occurs in the connector associated with frictional effects in the direction specified by *tangentDirection*. This radius is relevant only if the connection type includes an available rotational component of relative motion and *tangentDirection*=SLIP_DIRECTION.
- Internal contact force/moment generating friction in the direction specified by *tangentDirection*.
- Connector constitutive relative motion in the direction specified by *independentComponent*.
- Accumulated slip in the direction specified by *tangentDirection*, if the data depend on accumulated slip.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

independentComponents

A sequence of Ints specifying the independent components. Possible values are 1 $\leq$ *independentComponents* $\leq$ 6. In addition, each independent component value must be unique. The *independentComponents* argument applies only if *frictionModel*=USER_CUSTOMIZED. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorFriction object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorFriction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorFriction* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorFriction object has members with the same names and descriptions as the arguments to the *ConnectorFriction* method.

In addition, the ConnectorFriction object can have the following members:

tangentialBehavior

A *TangentialBehavior* object.

derivedComponent

A *DerivedComponent* object specifying the *DerivedComponent* used to compute the contact force component direction. This argument applies only if *frictionModel*=USER_CUSTOMIZED, if *useContactForceComponent*=ON, and if *contactForceStyle*=DERIVED_COMPONENT.

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords

- **CONNECTOR FRICTION*, **FRICTION*, **CONNECTOR POTENTIAL*, **CONNECTOR DERIVED COMPONENT*

ConnectorLock object

The ConnectorLock object defines locking criteria for one or more available components of a connector's relative motion.

The ConnectorLock object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorLock(...)

This method creates a connector lock behavior option for a *ConnectorSection*.

Path

```
import connectorBehavior
connectorBehavior.ConnectorLock
import odbConnectorBehavior
odbConnectorBehavior.ConnectorLock
```

Required arguments

None.

Optional arguments

lockingComponent The SymbolicConstant ALL or an Int specifying the motion components that are locked. If an Int is specified, only that motion component is locked when the locking criteria are satisfied. If *lockingComponent*=ALL, all motion components are locked. The default value is ALL.

minMotion `None` or a Float specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is `None`.

maxMotion `None` or a Float specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is `None`.

minForce `None` or a Float specifying the lower bound of the force or moment in the directions of the specified components at which locking occurs, or no lower bound. The default value is `None`.

maxForce `None` or a Float specifying the upper bound of the force or moment in the directions of the specified components at which locking occurs, or no upper bound. The default value is `None`.

components A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorLock object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorLock object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorLock* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorLock object has members with the same names and descriptions as the arguments to the *ConnectorLock* method.

Corresponding analysis keywords

**CONNECTOR LOCK*

ConnectorOptions object

The ConnectorOptions object is used to define various options for connector behaviors. It can be used only in conjunction with *CDCTerm*, *ConnectorDamage*, *ConnectorDamping*, *ConnectorElasticity*, *ConnectorFriction*, and *ConnectorPlasticity* objects. Because the *ConnectorDamage* object contains two separate ConnectorOptions repositories (one for damage initiation and one for damage evolution), there are two ConnectorOptions constructors associated with that behavior—initiationOptions and evolutionOptions. The *ConnectorPlasticity* object also contains two separate ConnectorOptions repositories (one for isotropic hardening and one for kinematic hardening), so there are two ConnectorOptions constructors associated with that behavior—isotropicOptions and kinematicOptions.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i].options
mdb.models[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i].options
mdb.models[name].sections[name].behaviorOptions[i].evolutionOptions
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.cdcTerms[i].options
mdb.models[name].sections[name].behaviorOptions[i].initiationOptions
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i].options
mdb.models[name].sections[name].behaviorOptions[i].isotropicOptions
mdb.models[name].sections[name].behaviorOptions[i].kinematicOptions
mdb.models[name].sections[name].behaviorOptions[i].options
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i].options
session.odbs[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i].options
session.odbs[name].sections[name].behaviorOptions[i].evolutionOptions
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.cdcTerms[i].options
session.odbs[name].sections[name].behaviorOptions[i].initiationOptions
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i].options
session.odbs[name].sections[name].behaviorOptions[i].isotropicOptions
session.odbs[name].sections[name].behaviorOptions[i].kinematicOptions
session.odbs[name].sections[name].behaviorOptions[i].options
```

ConnectorOptions(...)

This method creates a connector options object to be used in conjunction with an allowable connector behavior option, derived component term, or connector section.

Path

```
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i]\
```

```
.evolutionPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
mdb.models[name].sections[name].behaviorOptions[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i].derivedComponent\
.cdcTerms[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent.cdcTerms[i].ConnectorOptions
session.odbs[name].sections[name].behaviorOptions[i].ConnectorOptions
```

Required arguments

None.

Optional arguments

useBehRegSettings

A Boolean specifying whether or not to use the behavior-level settings for regularization options. This argument is applicable only for an Abaqus/Explicit analysis. The default value is ON.

regularize

A Boolean specifying whether or not the tabular data will be regularized. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=OFF. The default value is ON.

defaultTolerance

A Boolean specifying whether or not the analysis default regularization tolerance will be used. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=OFF and *regularize*=ON. The default value is ON.

regularization

A Float specifying the regularization increment to be used. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=OFF, *regularize*=ON, and *defaultTolerance*=OFF. The default value is 0.03.

defaultRateFactor

A Boolean specifying whether or not the analysis default rate filter factor will be used. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic motion criteria. The default value is ON.

rateFactor

A Float specifying the rate filter factor to be used. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic motion criteria. This argument is also applicable only if *defaultRateFactor*=OFF. The default value is 0.9.

interpolation

A SymbolicConstant specifying the type of interpolation increment to be used on rate-dependent tabular data. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic

motion criteria. Possible values are LINEAR and LOGARITHMIC. The default value is LINEAR.

useBehExtSettings
A Boolean specifying whether or not to use the behavior-level settings for extrapolation options. The default value is ON.

extrapolation
A SymbolicConstant specifying the extrapolation technique to be used. This argument is applicable only if *useBehExtSettings*=OFF. Possible values are CONSTANT and LINEAR. The default value is CONSTANT.

Return value

A ConnectorOptions object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorOptions object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorOptions* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorOptions object has members with the same names and descriptions as the arguments to the *ConnectorOptions* method.

Corresponding analysis keywords

- **CONNECTOR BEHAVIOR*, **CONNECTOR DAMAGE INITIATION*, **CONNECTOR DAMAGE EVOLUTION*, **CONNECTOR DAMPING*, **CONNECTOR DERIVED COMPONENT*, **CONNECTOR ELASTICITY*, **CONNECTOR FRICTION*, and **CONNECTOR PLASTICITY*

ConnectorPlasticity object

The ConnectorPlasticity object defines plastic behavior for one or more components of a connector's relative motion.

The ConnectorPlasticity object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorPlasticity(...)

This method creates a connector plasticity behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorPlasticity
import odbConnectorBehavior
odbConnectorBehavior.ConnectorPlasticity
```

Required arguments

None.

Optional arguments

coupling

A SymbolicConstant specifying whether or not the behavior is coupled. Possible values are UNCOUPLED and COUPLED. The default value is UNCOUPLED.

isotropic

A Boolean specifying whether isotropic hardening data will be used. The default value is ON.

If *isotropic*=OFF, then *kinematic* must be specified as ON.

isotropicType

A SymbolicConstant specifying the type of isotropic hardening to be specified. Possible values are TABULAR and EXPONENTIAL_LAW. The default value is TABULAR.

This argument is applicable only if *isotropic*=ON.

isotropicTemperature

A Boolean specifying whether the isotropic data depend on temperature. The default value is OFF.

This argument is applicable only if *isotropic*=ON.

isotropicDependencies

An Int specifying the number of field variable dependencies for the isotropic data. The default value is 0.

This argument is applicable only if *isotropic*=ON.

kinematic

A Boolean specifying whether kinematic hardening data will be used. The default value is OFF.

If *kinematic*=OFF, then *isotropic* must be specified as ON.

kinematicType

A SymbolicConstant specifying the type of kinematic hardening to be specified. Possible values are HALF_CYCLE, STABILIZED, and PARAMETERS. The default value is HALF_CYCLE.

This argument is applicable only if *kinematic*=ON.

kinematicTemperature

A Boolean specifying whether the kinematic data depend on temperature. The default value is OFF.

This argument is applicable only if *kinematic*=ON.

kinematicDependencies

An Int specifying the number of field variable dependencies for the kinematic data. The default value is 0.

This argument is applicable only if *kinematic*=ON.

forcePotentialOperator

A SymbolicConstant specifying the contribution operator for the force potential contributions. Possible values are SUM and MAXIMUM. The default value is SUM.

This argument is applicable only if *coupling*=COUPLED.

forcePotentialExponent

A Float specifying the number equal to the inverse of the overall exponent in the force potential definition. The default value is 2.0.

This argument is applicable only if *coupling*=COUPLED and if *forcePotentialOperator*=SUM.

connectorPotentials

A *ConnectorPotentialArray* object specifying one *ConnectorPotential* object for each force potential contribution. This member can be specified only if *coupling*=COUPLED.

isotropicTable

A sequence of sequences of Floats specifying isotropic plasticity properties. Items in the *isotropicTable* data are described below. This argument is applicable only if *isotropic*=ON. The default value is an empty sequence.

kinematicTable

A sequence of sequences of Floats specifying kinematic plasticity properties. Items in the *kinematicTable* data are described below. This argument is applicable only if *kinematic*=ON. The default value is an empty sequence.

components

A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument can be specified only if *coupling*=UNCOUPLED. The default value is an empty sequence.

Table data

Table data for *isotropicTable*:

If *isotropicType*=TABULAR, then each sequence of the table data specifies the following:

- Equivalent yield force or moment defining the size of the elastic range.
- Equivalent relative plastic motion.
- Equivalent relative plastic motion rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *isotropicType*=EXPONENTIAL_LAW, then each sequence of the table data specifies the following:

- Equivalent force or moment defining the size of the elastic range at zero plastic motion.
- Isotropic hardening parameter $Q_{\mathbf{inf}}$.
- Isotropic hardening parameter $\boldsymbol{b}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Table data for *kinematicTable*:

If *kinematicType*=HALF_CYCLE, then each sequence of the table data specifies the following:

- Yield force or moment.
- Connector relative plastic motion.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *kinematicType*=STABILIZED, then each sequence of the table data specifies the following:

- Yield force or moment.
- Connector relative plastic motion.
- Connector relative constitutive motion range.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *kinematicType*=PARAMETERS, then each sequence of the table data specifies the following:

- Yield force or moment at zero relative plastic motion.
- Kinematic hardening parameter C.
- Kinematic hardening parameter γ. Set γ=0 to specify linear Ziegler kinematic hardening.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorPlasticity object.

Exceptions

ValueError and TextError.

setValues(...)

This method modifies the ConnectorPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorPlasticity* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorPlasticity object has members with the same names and descriptions as the arguments to the *ConnectorPlasticity* method.

In addition, the ConnectorPlasticity object can have the following members:

isotropicOptions
A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the isotropic hardening table.

kinematicOptions
A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the kinematic hardening table.

Corresponding analysis keywords

- **CONNECTOR PLASTICITY*, **CONNECTOR HARDENING*, **CONNECTOR POTENTIAL*

ConnectorPotential object

The ConnectorPotential object is used to define a restricted set of mathematical functions to represent yield or limiting surfaces in the space spanned by connector available components. It can be used only in conjunction with *ConnectorDamage*, *ConnectorFriction*, and *ConnectorPlasticity* objects. Because the *ConnectorDamage* object contains two separate ConnectorPotential repositories (one for damage initiation and one for damage evolution), there are two ConnectorPotential constructors associated with that behavior—IniPotential and EvoPotential.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i]
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i]
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i]
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i]
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i]
```

ConnectorPotential(...)

This method creates a connector potential object to be used in conjunction with an allowable connector behavior option.

Path

```
mdb.models[name].sections[name].behaviorOptions[i].ConnectorPotential
session.odbs[name].sections[name].behaviorOptions[i]\
.ConnectorPotential
```

Required arguments

None.

Optional arguments

componentStyle

A SymbolicConstant specifying whether a component number or the name of the *DerivedComponent* object will be used in the contribution. Possible values are COMPONENT_NUMBER and DERIVED_COMPONENT. The default value is COMPONENT_NUMBER.

componentNumber

An Int specifying the component number used in the contribution. This argument is applicable only if *componentStyle*=COMPONENT_NUMBER. Possible values are 1 $\leq$*componentNumber*$\leq$ 6. Only available components can be specified. The default value is 0.

sign
A SymbolicConstant specifying the sign of the contribution. Possible values are POSITIVE and NEGATIVE. The default value is POSITIVE.

scaleFactor
A Float specifying the scaling factor for the contribution. The default value is 1.0.

positiveExponent
A Float specifying the positive exponent for the contribution. The default value is 2.0.

This argument is ignored if the potential operator of the invoking behavior option is set to MAXIMUM.

shiftFactor
A Float specifying the shift factor for the contribution. The default value is 0.0.

hFunction
A SymbolicConstant specifying the H function of the contribution: either absolute value, Macauley bracket, or the identity function. Possible values are ABS, MACAULEY, and IDENTITY. The default value is ABS.

The value of IDENTITY can be used only if *positiveExponent*=1.0 and the potential exponent of the invoking behavior option is also 1.0 (i.e., the potential operator of the invoking behavior option must be SUM).

Return value

A ConnectorPotential object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorPotential object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorPotential* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorPotential object has members with the same names and descriptions as the arguments to the *ConnectorPotential* method.

In addition, the ConnectorPotential object can have the following member:

derivedComponent A *DerivedComponent* object specifying the *DerivedComponent* used in the contribution. This argument is applicable only if *componentStyle*=DERIVED_COMPONENT.

Corresponding analysis keywords

- **CONNECTOR POTENTIAL*

ConnectorStop object

The ConnectorStop object defines connector stops for one or more components of a connector's relative motion.

The ConnectorStop object is derived from the *ConnectorBehaviorOption* object.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]
```

ConnectorStop(...)

This method creates a connector stop behavior option for a *ConnectorSection* object.

Path

```
import connectorBehavior
connectorBehavior.ConnectorStop
import odbConnectorBehavior
odbConnectorBehavior.ConnectorStop
```

Required arguments

None.

Optional arguments

minMotion

`None` or a Float specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is `None`.

maxMotion

`None` or a Float specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is `None`.

components

A sequence of Ints specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorStop object.

Exceptions

ValueError and TextError.

setValues(...)

This method modifies the ConnectorStop object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorStop* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorStop object has members with the same names and descriptions as the arguments to the *ConnectorStop* method.

Corresponding analysis keywords

- **CONNECTOR STOP*

DerivedComponent object

A DerivedComponent object describes user-customized components for use in defining *ConnectorFriction* and *Potential* objects.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent
mdb.models[name].sections[name].behaviorOptions[i].derivedComponent
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].derivedComponent
session.odbs[name].sections[name].behaviorOptions[i].derivedComponent
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].derivedComponent
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].derivedComponent
```

DerivedComponent()

This method creates a DerivedComponent object.

Path

```
mdb.models[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].DerivedComponent
mdb.models[name].sections[name].behaviorOptions[i].DerivedComponent
mdb.models[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].DerivedComponent
mdb.models[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].DerivedComponent
session.odbs[name].sections[name].behaviorOptions[i]\
.connectorPotentials[i].DerivedComponent
session.odbs[name].sections[name].behaviorOptions[i].DerivedComponent
session.odbs[name].sections[name].behaviorOptions[i]\
.evolutionPotentials[i].DerivedComponent
session.odbs[name].sections[name].behaviorOptions[i]\
.initiationPotentials[i].DerivedComponent
```

Arguments

None.

Return value

A DerivedComponent object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the DerivedComponent object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DerivedComponent* method.

Return value

`None.`

Exceptions

ValueError.

Members

The DerivedComponent object can have the following member:

cdcTerms A *CDCTermArray* object.

Corresponding analysis keywords

- **CONNECTOR DERIVED COMPONENT*

TangentialBehavior object

The TangentialBehavior object specifies tangential behavior for a connector friction behavior option.

Access

```
import section
mdb.models[name].sections[name].behaviorOptions[i].tangentialBehavior
import odbSection
session.odbs[name].sections[name].behaviorOptions[i]\
.tangentialBehavior
```

TangentialBehavior(...)

This method creates a TangentialBehavior object.

Path

```
mdb.models[name].sections[name].behaviorOptions[i].TangentialBehavior
session.odbs[name].sections[name].behaviorOptions[i]\
.TangentialBehavior
```

Required arguments

None.

Optional arguments

formulation
A SymbolicConstant specifying the friction coefficient formulation. Possible values are PENALTY and EXPONENTIAL_DECAY. The default value is PENALTY.

slipRateDependency
A Boolean specifying whether the data depend on slip rate. The default value is OFF.

pressureDependency
A Boolean specifying whether the data depend on contact pressure. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variables for the data. The default value is 0.

exponentialDecayDefinition
A SymbolicConstant specifying the exponential decay definition for the data. Possible values are COEFFICIENTS and TEST_DATA. The default value is COEFFICIENTS.

shearStressLimit
`None` or a Float specifying no upper limit or the friction coefficient shear stress limit. The default value is `None`.

maximumElasticSlip
A SymbolicConstant specifying the method for modifying the allowable elastic slip. Possible values are FRACTION and ABSOLUTE_DISTANCE. The default value is FRACTION.

This argument applies only to Abaqus/Standard analyses.

fraction
A Float specifying the ratio of the allowable maximum elastic slip to a characteristic model dimension. The default value is 10^{-4}.

This argument applies only to Abaqus/Standard analyses.

absoluteDistance
`None` or a Float specifying the absolute magnitude of the allowable elastic slip. The default value is `None`.

This argument applies only to Abaqus/Standard analyses.

table
A sequence of sequences of Floats specifying the tangential properties. Items in the table data are described below. The default value is an empty sequence.

Table data

If *formulation*=PENALTY, the table data specify the following:

- Friction coefficient in the slip direction, μ.
- Slip rate, if the data depend on slip rate.
- Contact pressure, if the data depend on contact pressure.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=COEFFICIENTS, the table data specify the following:

- Static friction coefficient, μ_s.
- Kinetic friction coefficient, μ_k.
- Decay coefficient, d_c.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=TEST_DATA, the table data specify the following:

- Static coefficient of friction.
- Dynamic friction coefficient measured at the reference slip rate, $\dot{\gamma}_2$.
- Reference slip rate, $\dot{\gamma}_2$, used to measure the dynamic friction coefficient.
- Kinetic friction coefficient, μ_∞. This value corresponds to the asymptotic value of the friction coefficient at infinite slip rate, $\dot{\gamma}_\infty$.

Return value

A TangentialBehavior object.

Exceptions

None.

`setValues(...)`

This method modifies the TangentialBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TangentialBehavior* method.

Return value

`None.`

Exceptions

None.

Members

The TangentialBehavior object has members with the same names and descriptions as the arguments to the *TangentialBehavior* method.

Corresponding analysis keywords

- **FRICTION*

Constraint commands

The Constraint commands define constraints between regions of the model.

In this section:

- *Constraint object*
- *AdjustPoints object*
- *Coupling object*
- *DisplayBody object*
- *EmbeddedRegion object*
- *Equation object*
- *MultipointConstraint object*
- *RigidBody object*
- *ShellSolidCoupling object*
- *Tie object*

Constraint object

The Constraint object is the abstract base type for other Constraint objects. The Constraint object has no explicit constructor. The members of the Constraint object are common to all objects derived from the Constraint.

Access

```
import interaction
mdb.models[name].constraints[name]
```

`resume()`

This method resumes the constraint that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the constraint.

Arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

This method allows you to delete existing constraints.

Required arguments

indices A sequence of Ints specifying the index of each constraint to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Constraint object has the following members:

name

A String specifying the constraint repository key.

suppressed

A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

AdjustPoints object

The AdjustPoints constraint object is used to adjust points (nodes) to a surface.

The AdjustPoints object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

AdjustPoints(...)

This method creates an AdjustPoints object.

Path

```
mdb.models[name].AdjustPoints
```

Required arguments

name	A String specifying the constraint repository key.
surface	A *Region* object specifying the surface to which the *controlPoints* are adjusted.
controlPoints	A *Region* object specifying the constraint control points.

Optional arguments

None.

Return value

An AdjustPoints object.

Exceptions

None.

`setValues(...)`

This method modifies the AdjustPoints object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AdjustPoints* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The AdjustPoints object has members with the same names and descriptions as the arguments to the *AdjustPoints* method.

In addition, the AdjustPoints object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **ADJUST*

Coupling object

The Coupling object defines a constraint between a group of coupling nodes located on a region and a reference point.

The Coupling object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

Coupling(...)

This method creates a Coupling object.

Path

```
mdb.models[name].Coupling
```

Required arguments

controlPoint A *Region* object specifying the constraint control point.

couplingType A SymbolicConstant specifying the coupling constraint type. Possible values are KINEMATIC, DISTRIBUTING, STRUCTURAL, and UNIFORM.

influenceRadius The SymbolicConstant WHOLE_SURFACE or a Float specifying the influence radius.

name A String specifying the constraint repository key.

rotationalCouplingType A SymbolicConstant specifying the rotational coupling constraint type. Possible values are ROTATIONAL_CONTINUUM and ROTATIONAL_STRUCTURAL.

surface A *Region* object specifying the surface on which the coupling nodes are located.

Optional arguments

adjust A Boolean specifying if the control point will be adjusted (moved) to the surface. The point will be adjusted in the direction normal to the specified surface. The default value is OFF.

alpha A Float specifying the value of the thermal expansion coefficient. The default value is 0.0.

The *alpha* argument applies only when *couplingType*=KINEMATIC.

localCsys `None` or a *DatumCsys* object specifying the initial orientation of the local coordinate system for the coupling's degrees of freedom. If *localCsys*=`None`, the coupling is defined in the global coordinate system. The default value is `None`.

temperatureDOFType An integer specifying the temperature degree of freedom to be coupled. Possible values range from 1 to 20. The default value is 0.

The *temperatureDOFType* argument applies only when *couplingType*=UNIFORM and *uniformConstraint*=DOF_TEMPERATURE.

uniformConstraint

A SymbolicConstant specifying the nonmechanical degrees of freedom of surface nodes to be coupled to the same degree of freedom of a reference node. Possible values are WARP_AMPLITUDE, PORE_PRESSURE, ELECTRIC_POTENTIAL, CONNECTOR_MATERIAL_FLOW, DOF_TEMPERATURE, SLURRY_CONCENTRATION, ELECTROLYTE_ELECTRIC_POTENTIAL, ION_CONCENTRATION, and SPECIES_CONCENTRATION.

The *uniformConstraint* argument applies only when *couplingType*=UNIFORM.

u1

A Boolean specifying if the displacement component in the 1-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *u1* argument applies only when *couplingType*=KINEMATIC.

u2

A Boolean specifying if the displacement component in the 2-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *u2* argument applies only when *couplingType*=KINEMATIC.

u3

A Boolean specifying if the displacement component in the 3-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *u3* argument applies only when *couplingType*=KINEMATIC.

ur1

A Boolean specifying if the rotational displacement component about the 1-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *ur1* argument applies only when *couplingType*=KINEMATIC.

ur2

A Boolean specifying if the rotational displacement component about the 2-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *ur2* argument applies only when *couplingType*=KINEMATIC.

ur3

A Boolean specifying if the rotational displacement component about the 3-direction is constrained to the reference node for a kinematic coupling constraint. The default value is ON.

The *ur3* argument applies only when *couplingType*=KINEMATIC.

weightingMethod

A SymbolicConstant specifying an optional weighting method used for calculating the distributing weight factors. Possible values are UNIFORM, LINEAR, QUADRATIC, and CUBIC. The default value is UNIFORM.

The *weightingMethod* argument applies only when *couplingType*=DISTRIBUTING.

Return value

A Coupling object.

Exceptions

None.

`setValues(...)`

This method modifies the Coupling object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Coupling* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Coupling object has members with the same names and descriptions as the arguments to the *Coupling* method.

In addition, the Coupling object has the following member:

suppressed A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **COUPLING*

DisplayBody object

The DisplayBody object defines a constraint such that the specified instance is used for display only and does not take part in the analysis. However it will still be visible during postprocessing and its position at any frame will be defined by the translation and rotation of the specified control points.

The DisplayBody object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

DisplayBody(...)

This method creates a DisplayBody object.

Path

```
mdb.models[name].DisplayBody
```

Required arguments

name A String specifying the constraint repository key.

instance A *PartInstance* object specifying the part instance that is to be used for display only.

controlPoints A *ModelDotArray* object specifying the motion of the *PartInstance*. The control points may be Vertex, ReferencePoint, or MeshNode objects. Their motion will control the motion of the PartInstance. If this argument is set to an empty sequence, the PartInstance will remain fixed in space during the analysis. The sequence can have either one object or three objects.

Optional arguments

None.

Return value

A DisplayBody object.

Exceptions

None.

setValues(...)

This method modifies the DisplayBody object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DisplayBody* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DisplayBody object has members with the same names and descriptions as the arguments to the *DisplayBody* method.

In addition, the DisplayBody object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **DISPLAY BODY*

EmbeddedRegion object

The EmbeddedRegion object allows you to embed a region of the model within a “host” region of the model or within the whole model.

The EmbeddedRegion object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

EmbeddedRegion(...)

This method creates a EmbeddedRegion object.

Path

```
mdb.models[name].EmbeddedRegion
```

Required arguments

name
A String specifying the constraint repository key.

embeddedRegion
A *Region* object specifying the body region to be embedded.

hostRegion
A *Region* object specifying the host region. A value of `None` indicates that the host region is the whole model.

Optional arguments

weightFactorTolerance
A Float specifying a small value below which the weighting factors will be zeroed out. The default value is 10^{-6}.

toleranceMethod
A SymbolicConstant specifying the method used to determine the embedded element tolerance. Possible values are ABSOLUTE, FRACTIONAL, and BOTH. The default value is BOTH.

absoluteTolerance
A Float specifying the absolute value by which a node on the embedded region may lie outside the host region. If *absoluteTolerance*=0.0, the *fractionalTolerance* value will be used. The default value is 0.0.

This argument applies only when *toleranceMethod*=ABSOLUTE or BOTH.

fractionalTolerance
A Float specifying the fractional value by which a node on the embedded region may lie outside the host region. The fractional value is based on the average element size within the host region. The default value is 0.05.

If both tolerance arguments are specified, the smaller value will be used.

This argument applies only when *toleranceMethod*=FRACTIONAL or BOTH.

Return value

An EmbeddedRegion object.

Exceptions

None.

setValues(...)

This method modifies the EmbeddedRegion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EmbeddedRegion* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The EmbeddedRegion object has members with the same names and descriptions as the arguments to the *EmbeddedRegion* method.

In addition, the EmbeddedRegion object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **EMBEDDED ELEMENT*

Equation object

The Equation object defines a linear multi-point constraint between a set of degrees of freedom.

The Equation object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

Equation(...)

This method creates an Equation object.

Path

```
mdb.models[name].Equation
```

Required arguments

name A String specifying the constraint repository key.

terms A sequence of (Float, String, Int, Int) sequences specifying a coefficient, Set name, degree of freedom, and coordinate system ID. The coordinate system ID is optional.

Optional arguments

None.

Return value

An Equation object.

Exceptions

If *terms* does not contain more than one entry: Equation must have two or more terms.

`setValues(...)`

This method modifies the Equation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Equation* method, except for the *name* argument.

Return value

```
None.
```

Exceptions

If *terms* does not contain more than one entry: Equation must have two or more terms.

Members

The Equation object has members with the same names and descriptions as the arguments to the *Equation* method.

In addition, the Equation object has the following member:

suppressed A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **EQUATION*

MultipointConstraint object

The MultipointConstraint object defines a constraint between a group of MultipointConstraint nodes located on a region and a reference point.

The MultipointConstraint object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

MultipointConstraint(...)

This method creates a MultipointConstraint object.

Path

```
mdb.models[name].MultipointConstraint
```

Required arguments

name A String specifying the constraint repository key.

surface A *Region* object specifying the surface on which the MultipointConstraint nodes are located.

controlPoint A *Region* object specifying the constraint control point.

mpcType A SymbolicConstant specifying the MPC type of the constraint. Possible values are BEAM_MPC, ELBOW_MPC, PIN_MPC, LINK_MPC, TIE_MPC, and USER_MPC.

Optional arguments

csys `None` or a *DatumCsys* object specifying the initial orientation of the local coordinate system for the MultipointConstraint's degrees of freedom. If *localCsys*=`None`, the MultipointConstraint is defined in the global coordinate system. The default value is `None`.

userType An Int specifying to differentiate between different constraint types in a user-defined MultipointConstraint. The default value is 0.

The *userType* argument applies only when *mpcType*=USER_MPC.

userMode A SymbolicConstant specifying the mode of the constraint when it is user-defined. Possible values are DOF_MODE_MPC and NODE_MODE_MPC. The default value is DOF_MODE_MPC.

The *userMode* argument applies only when *mpcType*=USER_MPC.

Return value

A MultipointConstraint object.

Exceptions

None.

setValues(...)

This method modifies the MultipointConstraint object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MultipointConstraint* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The MultipointConstraint object has members with the same names and descriptions as the arguments to the *MultipointConstraint* method.

In addition, the MultipointConstraint object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **MPC*

RigidBody object

The RigidBody object constrains all the degrees of freedom on the specified regions to the degree of freedom of its associated reference point.

The RigidBody object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

RigidBody(...)

This method creates a RigidBody object.

Path

```
mdb.models[name].RigidBody
```

Required arguments

name A String specifying the constraint repository key.

refPointRegion A *Region* object specifying the reference point.

Optional arguments

bodyRegion `None` or a *Region* object specifying the elements constrained to the movement of the reference point. The default value is `None`.

tieRegion `None` or a *Region* object specifying the nodes tied to the movement of the reference point. The default value is `None`.

pinRegion `None` or a *Region* object specifying the nodes pinned to the movement of the reference point. The default value is `None`.

surfaceRegion `None` or a *Region* object specifying the analytic surface constrained to the movement of the reference point. The default value is `None`.

refPointAtCOM A Boolean specifying whether the analysis product should recompute the reference point position to be at the center of mass. The default value is OFF.

isothermal A Boolean specifying whether the temperature degree of freedom should be constrained. The default value is OFF.

Return value

A RigidBody object.

Exceptions

None.

setValues(...)

This method modifies the RigidBody object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RigidBody* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The RigidBody object has members with the same names and descriptions as the arguments to the *RigidBody* method.

In addition, the RigidBody object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **RIGID BODY*

ShellSolidCoupling object

The ShellSolidCoupling object defines two surfaces to be tied together for the duration of a simulation.

The ShellSolidCoupling object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

ShellSolidCoupling(...)

This method creates a ShellSolidCoupling object.

Path

```
mdb.models[name].ShellSolidCoupling
```

Required arguments

name
A String specifying the constraint repository key.

shellEdge
A *Region* object specifying the name of the shell edge surface.

solidFace
A *Region* object specifying the name of the solid surface.

Optional arguments

positionToleranceMethod
A SymbolicConstant specifying the method used to determine the position tolerance. Possible values are COMPUTED and SPECIFIED. The default value is COMPUTED.

positionTolerance
A Float specifying the position tolerance. The default value is 0.0.

The *positionTolerance* argument applies only when *positionToleranceMethod*=SPECIFIED.



Note:

Abaqus will not constrain nodes on the solid face region outside the position tolerance.

influenceDistanceMethod
A SymbolicConstant specifying the method used to determine the influence distance. Possible values are DEFAULT and SPECIFIED. The default value is DEFAULT.

influenceDistance
A Float specifying the influence distance. The *influenceDistance* argument applies only when *influenceDistanceMethod*=SPECIFIED. The default value is 0.0.

Return value

A ShellSolidCoupling object.

Exceptions

None.

setValues(...)

This method modifies the ShellSolidCoupling object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShellSolidCoupling* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShellSolidCoupling object has members with the same names and descriptions as the arguments to the *ShellSolidCoupling* method.

In addition, the ShellSolidCoupling object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **SHELL TO SOLID COUPLING*

Tie object

The Tie object defines two surfaces to be tied together for the duration of a simulation.

The Tie object is derived from the *Constraint* object.

Access

```
import interaction
mdb.models[name].constraints[name]
```

Tie(...)

This method creates a Tie object.

Path

```
mdb.models[name].Tie
```

Required arguments

name	A String specifying the constraint repository key.
main	A *Region* object specifying the name of the main surface.
secondary	A *Region* object specifying the name of the secondary surface.

Optional arguments

adjust	A Boolean specifying whether initial positions of tied secondary nodes are adjusted to lie on the main surface. The default value is ON.
positionToleranceMethod	A SymbolicConstant specifying the method used to determine the position tolerance. Possible values are COMPUTED and SPECIFIED. The default value is COMPUTED.
positionTolerance	A Float specifying the position tolerance. The *positionTolerance* argument applies only when *positionToleranceMethod*=SPECIFIED. The default value is 0.0.
tieRotations	A Boolean specifying whether rotation degrees of freedom should be tied. The default value is ON.
tieTemperature	A Boolean specifying whether temperature degrees of freedom should be tied. The default value is ON.
tiePore	A Boolean specifying whether pore pressure degrees of freedom should be tied. The default value is ON.
electricPotential	A Boolean specifying whether electric potential in solid degrees of freedom should be tied. The default value is ON.

fluidElectricPotential A Boolean specifying whether electric potential in fluid degrees of freedom should be tied. The default value is ON.

ionConcentration A Boolean specifying whether ion concentration in fluid degrees of freedom should be tied. The default value is ON.

constraintRatioMethod A SymbolicConstant specifying the method used to determine the constraint ratio. Possible values are DEFAULT and SPECIFIED. The default value is DEFAULT.

constraintRatio A Float specifying the fractional distance between the main reference surface and the secondary node at which the translational constraint should act. The *constraintRatio* argument applies only when *constraintRatioMethod*=SPECIFIED. The default value is 0.0.

constraintEnforcement A SymbolicConstant specifying the discretization method. Possible values are SOLVER_DEFAULT, NODE_TO_SURFACE, and SURFACE_TO_SURFACE. The default value is SOLVER_DEFAULT.

thickness A Boolean specifying whether shell element thickness is considered. The default value is ON.

Return value

A Tie object.

Exceptions

None.

swapSurfaces()

This method switches the main and secondary surfaces of a tied constraint. This command is valid only during the step in which the interaction is created.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the Tie object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Tie* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Tie object has members with the same names and descriptions as the arguments to the *Tie* method.

In addition, the Tie object has the following member:

suppressed
A Boolean specifying whether the constraint is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **TIE*

CustomKernel module

The `customKernel` module augments the mdb, odb, and session objects with an object called customData. The customData object can contain custom data that you created using standard Python objects; for example, dictionaries, strings, numbers, and classes. In addition, the customData object can contain objects that you created using the classes described in this section.

All objects that you create under the mdb.customData and odb[odbName].customData object will be stored in the Abaqus/CAE model and output database, respectively, when you save the database; all objects that you create under the session.customData object will be lost when you exit Abaqus/CAE.

The customData object is an instance of the RepositorySupport class; for more information, see *RepositorySupport*. The classes described in this section register with the GUI infrastructure when you create the class. As a result, if you write a custom GUI script, you can query these objects from the GUI, and you can also be notified when the contents of these objects change so you can update the GUI accordingly. For more information, see *Extending the Abaqus Scripting Interface* and *Accessing kernel data from the GUI*.

In this section:

- *CommandRegister object*
- *RepositorySupport object*
- *RegisteredDictionary object*
- *RegisteredList object*
- *RegisteredTuple object*

CommandRegister object

This class allows you to derive a general class that can be queried from the GUI and is capable of notifying the GUI when the contents of the class change.

Access

```
import customKernel
```

CommandRegister()

This class allows you to derive a general class that can be queried from the GUI and is capable of notifying the GUI when the contents of the class change.

Path

customKernel.CommandRegister

Arguments

None.

Return value

A CommandRegister object.

Exceptions

None.

Members

The CommandRegister object has no members.

RepositorySupport object

The RepositorySupport is a base class from which you can derive your own classes that are designed to contain custom repositories. Instances of this class can be queried from the GUI and are capable of notifying the GUI when the contents of the instance change.

The RepositorySupport object is derived from the *CommandRegister* object.

Access

```
import customKernel
mdb.customData
session.customData
session.odbs[name].customData
```

RepositorySupport()

This method creates a RepositorySupport object.

Path

customKernel.RepositorySupport

Arguments

None.

Return value

A RepositorySupport object.

Exceptions

None.

Repository(...)

This method installs a repository on the class. The repository is an instance of a RegisteredDictionary class. Refer to *RegisteredDictionary* for details on its methods.

The objects stored in the repository are assumed to have an attribute called *name* that stores the key used to access the object in the repository. The name attribute will be modified by the `changeKey` method.

Required arguments

name	A String specifying the name of the repository.
constructors	A constructor or sequence of constructors specifying which classes will store their instances in the repository.

Optional arguments

None.

Return value

```
None.
```

Exceptions

None.

Members

The RepositorySupport object has no members.

RegisteredDictionary object

This class allows you to create a dictionary that can be queried from the GUI and is capable of notifying the GUI when the contents of the dictionary change. The keys to a RegisteredDictionary must be either strings or integers.

The RegisteredDictionary object is derived from the *CommandRegister* object.

Access

```
import customKernel
```

RegisteredDictionary()

This method creates a RegisteredDictionary object.

Path

customKernel.RegisteredDictionary

Arguments

None.

Return value

A RegisteredDictionary object.

Exceptions

None.

Methods()

The RegisteredDictionary object supports the same methods as a Python dictionary. In addition, the RegisteredDictionary object supports the `changeKey` method.

Arguments

None.

Return value

`None.`

Exceptions

None.

changeKey(...)

This method changes the name of a key in the dictionary.

Required arguments

fromName	A String or an integer specifying the name of the key to be changed.
toName	A String or an integer specifying the new name for the key.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The RegisteredDictionary object has no members.

RegisteredList object

This class allows you to create a list that can be queried from the GUI and is capable of notifying the GUI when the contents of the list change.

The RegisteredList object is derived from the *CommandRegister* object.

Access

```
import customKernel
```

RegisteredList()

This method creates a RegisteredList object.

Path

customKernel.RegisteredList

Arguments

None.

Return value

A RegisteredList object.

Exceptions

None.

Methods()

The RegisteredList object supports the same methods as a standard Python list object.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The RegisteredList object has no members.

RegisteredTuple object

This class allows you to create a tuple that can be queried from the GUI and is capable of notifying the GUI when the contents of any of the tuple's members change.

The RegisteredTuple object is derived from the *CommandRegister* object.

Access

```
import customKernel
```

RegisteredTuple(...)

This method creates a RegisteredTuple object.

Path

customKernel.RegisteredTuple

Required arguments

tuple A tuple of objects. These objects must be derived from the CommandRegister class.

Optional arguments

None.

Return value

A RegisteredTuple object.

Exceptions

None.

Methods()

The RegisteredTuple object supports the same methods as a standard Python list object.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The RegisteredTuple object has no members.

Datum commands

Datum commands return Feature objects and inherit the methods of Feature objects.
For more details, see *Feature commands*. Datums can be created using methods on a Part or Assembly object.

Each command also creates a Datum object in the corresponding datum repository. The Datum object is used as an argument to other commands, such as Part and Partition commands.

In this section:

- *Datum object*
- *DatumAxis object*
- *DatumCsys object*
- *DatumPlane object*
- *DatumPoint object*

Datum object

The Datum object is the abstract base type for other Datum objects. The Datum object has no explicit constructor. The methods and members of the Datum object are common to all objects derived from the Datum.

Access

```
import part
mdb.models[name].parts[name].datums[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].datums[i]
mdb.models[name].rootAssembly.datums[i]
mdb.models[name].rootAssembly.instances[name].datums[i]
mdb.models[name].rootAssembly.modelInstances[i].datums[i]
```

Members

The Datum object has no members.

DatumAxis object

The DatumAxis object has no direct constructor; it is created when a *Feature* object is created. For example, the DatumAxisByCylFace method creates a Feature object that creates a DatumAxis object.

The DatumAxis object is derived from the *Datum* object.

Access

```
import part
mdb.models[name].parts[name].datums[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].datums[i]
mdb.models[name].rootAssembly.datums[i]
mdb.models[name].rootAssembly.instances[name].datums[i]
```

Members

The DatumAxis object has the following members:

pointOn

A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the datum.

direction

A tuple of Floats specifying a sequence of three Floats specifying the direction of the axis.

DatumCsys object

The DatumCsys object has no direct constructor; it is created when a *Feature* object is created. For example, the DatumCsysByOffset method creates a Feature object that creates a DatumCsys object.

The DatumCsys object is derived from the *Datum* object.

Access

```
import part
mdb.models[name].parts[name].datums[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].datums[i]
mdb.models[name].rootAssembly.connectorOrientations[i].localCsys1
mdb.models[name].rootAssembly.connectorOrientations[i].localCsys2
mdb.models[name].rootAssembly.datums[i]
mdb.models[name].rootAssembly.instances[name].datums[i]
import odbAccess
session.odbs[name].rootAssembly.connectorOrientations[i].localCsys1
session.odbs[name].rootAssembly.connectorOrientations[i].localCsys2
```

`globalToLocal(...)`

This method transforms specified coordinates in the global coordinate system into this local coordinate system.

Required arguments

coordinates	A tuple of three Floats representing the coordinates in the global coordinate system.

Optional arguments

None.

Return value

A tuple of three Floats representing the coordinates in this local coordinate system.

Exceptions

None.

`localToGlobal(...)`

This method transforms specified coordinates in this local coordinate system into the global coordinate system.

Required arguments

coordinates	A tuple of three Floats representing the coordinates in this local coordinate system.

Optional arguments

None.

Return value

A tuple of three Floats representing the coordinates in the global coordinate system.

Exceptions

None.

Members

The DatumCsys object has the following members:

coordSysType
A SymbolicConstant specifying the type of the coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

origin
A *DatumPoint* object specifying the origin of the coordinate system.

axis1
A *DatumAxis* object specifying the 1-direction of the coordinate system.

axis2
A *DatumAxis* object specifying the 2-direction of the coordinate system.

axis3
A *DatumAxis* object specifying the 3-direction of the coordinate system.

DatumPlane object

The DatumPlane object has no direct constructor; it is created when a *Feature* object is created. For example, the DatumPlaneByPrincipalPlane method creates a Feature object that creates a DatumPlane object.

The DatumPlane object is derived from the *Datum* object.

Access

```
import part
mdb.models[name].parts[name].datums[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].datums[i]
mdb.models[name].rootAssembly.datums[i]
mdb.models[name].rootAssembly.instances[name].datums[i]
```

Members

The DatumPlane object has the following members:

pointOn A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the datum.

normal A tuple of Floats specifying a sequence of three Floats specifying the normal.

DatumPoint object

The DatumPoint object has no direct constructor; it is created when a *Feature* object is created. For example, the DatumPointByCoordinate method creates a Feature object that creates a DatumPoint object.

The DatumPoint object is derived from the *Datum* object.

Access

```
import part
mdb.models[name].parts[name].datums[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].datums[i]
mdb.models[name].rootAssembly.datums[i]
mdb.models[name].rootAssembly.instances[name].datums[i]
```

Members

The DatumPoint object has the following member:

pointOn A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point located on the datum.

Display Group commands

Display group commands are used to select a subset of the entities displayed in the viewport.

In this section:

- *DisplayGroup object*
- *DisplayGroupInstance object*
- *DisplayGroupInstanceRepository object*
- *Leaf object*
- *LeafFromConstraintNames object*
- *LeafFromDatums object*
- *LeafFromDisplayGroup object*
- *LeafFromElementLabels object*
- *LeafFromElementSets object*
- *LeafFromElementVarRange object*
- *LeafFromGeometry object*
- *LeafFromInstance object*
- *LeafFromInstanceElementLabels object*
- *LeafFromInstanceNodeLabels object*
- *LeafFromMeshElementLabels object*
- *LeafFromMeshNodeLabels object*
- *LeafFromMeshSurfaceSets object*
- *LeafFromModelElemLabels object*
- *LeafFromModelNodeLabels object*
- *LeafFromNodeLabels object*
- *LeafFromNodeSets object*
- *LeafFromNodeVarRange object*
- *LeafFromOdbElementLayups object*
- *LeafFromOdbElementMaterials object*
- *LeafFromOdbElementPick object*
- *LeafFromOdbElementPlies object*
- *LeafFromOdbElementSections object*
- *LeafFromOdbElementTypes object*
- *LeafFromOdbNodePick object*
- *LeafFromOdbEdgePick object*
- *LeafFromPartElementLabels object*
- *LeafFromPartInstance object*
- *LeafFromPartNodeLabels object*
- *LeafFromReferencePoint object*
- *LeafFromSets object*
- *LeafFromSurfaceSets object*
- *LeafFromSurfaceVarRange object*

DisplayGroup object

DisplayGroup objects are used to select a subset of the entities displayed in the viewport.

Access

```
session.displayGroups[name]
import assembly
session.viewports[name].assemblyDisplay.displayGroup
session.viewports[name].layers[name].assemblyDisplay.displayGroup
import visualization
session.viewports[name].layers[name].odbDisplay.displayGroup
import part
session.viewports[name].layers[name].partDisplay.displayGroup
session.viewports[name].odbDisplay.displayGroup
session.viewports[name].partDisplay.displayGroup
```

DisplayGroup(...)

This method creates a DisplayGroup object.

Path

session.DisplayGroup

Required arguments

name A String specifying the repository key.

leaf A *Leaf* object specifying the items in the display group.

Optional arguments

None.

Return value

A DisplayGroup object.

Exceptions

InvalidNameError.

add(...)

This method adds the specified items to the display group.

Required arguments

leaf A *Leaf* object specifying the items to add to the display group.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`either(...)`

This method redefines the display group to be only those items that are not shared by the *leaf* argument and by the display group.

Required arguments

leaf A *Leaf* object specifying the items to be excluded from the display group.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`intersect(...)`

This method redefines the display group to be only those items that are shared by the *leaf* argument and the display group.

Required arguments

leaf A *Leaf* object specifying the items to be included in the display group.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

redoLast()

This method redoes the last undone operation on the display group.

Arguments

None.

Return value

`None.`

Exceptions

None.

remove(...)

This method removes the specified items from the display group.

Required arguments

leaf A *Leaf* object specifying the items to remove from the display group.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

replace(...)

This method replaces the contents of the display group with the specified items.

Required arguments

leaf A *Leaf* object specifying the items with which to replace the current display group contents.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

undoLast()

This method undoes the last operation performed on the display group.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The DisplayGroup object has the following members:

canUndo
A Boolean specifying whether Undo is possible or not.

canRedo
A Boolean specifying whether Redo is possible or not.

name
A String specifying the repository key.

module
A SymbolicConstant specifying the module in which the display group has been created. The possible values are PART, ASSEMBLY, PART_ASSEMBLY, ODB, and ALL.

modelName
A String specifying the name of the model to which the display group belongs when the module is part- or assembly-based.

partName
A String specifying the name of the part to which the display group belongs when the module is part-based.

DisplayGroupInstance object

A DisplayGroupInstance object stores the IDs of the entities displayed in a viewport. The DisplayGroupInstance object has no constructor. When you set a display group to be plotted in a viewport, Abaqus/CAE creates a DisplayGroupInstance object for each display group and places it in the *DisplayGroupInstanceRepository* object.

Access

```
import assembly
session.viewports[name].assemblyDisplay.displayGroupInstances[name]
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name]
import visualization
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name]
import part
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name]
session.viewports[name].odbDisplay.displayGroupInstances[name]
session.viewports[name].partDisplay.displayGroupInstances[name]
```

`nodes()`

This method is used to obtain the list of nodes present in the DisplayGroupInstance object. It returns a Dictionary object keyed by part instance names, the value of which is a list of user node labels belonging to the part instance and contained in the DisplayGroupInstance object. This method is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of *OdbDisplay* object.

Arguments

None.

Return value

A Dictionary object.

Exceptions

None.

`elements()`

This method returns the list of elements present in the DisplayGroupInstance object. The `elements` method returns a Dictionary object that uses part instance names for the keys. The value of the items in the Dictionary object is a List of user element labels that belong to the part instance and are contained in the DisplayGroupInstance object. This method is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of the *OdbDisplay* object.

Arguments

None.

Return value

A Dictionary object.

Exceptions

None.

setValues(...)

This method modifies the DisplayGroupInstance object. The `setValues` method is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of the *OdbDisplay* object.

Required arguments

None.

Optional arguments

lockOptions A Boolean specifying whether the display options stored on the DisplayGroupInstance object should be synchronized with changes to the viewport display options. This member is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of the *OdbDisplay* object. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The DisplayGroupInstance object can have the following members:

name A String specifying the repository key.

lockOptions A Boolean specifying whether the display options stored on the DisplayGroupInstance object should be synchronized with changes to the viewport display options. This member is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of the *OdbDisplay* object. The default value is OFF.

odbDisplayOptions An *OdbDisplayOptions* object specifying this member is available only for DisplayGroupInstance objects that are members of the DisplayGroupInstance repository member of the *OdbDisplay* object.

DisplayGroupInstanceRepository object

The DisplayGroupInstanceRepository object stores *DisplayGroupInstance* objects. In addition to all the standard Python repository methods, the *DisplayGroupInstance* repository defines additional methods as described below.

Access

```
import visualization
session.viewports[name].layers[name].odbDisplay.displayGroupInstances
session.viewports[name].odbDisplay.displayGroupInstances
```

`syncOptions(...)`

This method synchronizes the display options stored on the *OdbDisplay* object with the display options stored on the *DisplayGroupInstance* object.

Required arguments

name A String specifying the repository key.

Optional arguments

updateInstances A Boolean specifying whether to synchronize the display options on all the *DisplayGroupInstance* objects stored in the DisplayGroupInstanceRepository for which *lockOptions* is OFF. The default value of *updateInstances* is ON.

Return value

`None.`

Exceptions

None.

Members

The DisplayGroupInstanceRepository object has no members.

Leaf object

Leaf objects are used to specify the items in a display group. Leaf objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

Leaf objects have similarities to Set objects; however, Leaf objects are evaluated when the DisplayGroup expression is evaluated, and they can have SymbolicConstant values (which are also evaluated when the DisplayGroup expression is evaluated).

Access

```
import displayGroupMdbToolset
import displayGroupOdbToolset
```

Leaf(...)

This method creates a Leaf object.

Path

```
Leaf
```

Required arguments

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

Optional arguments

None.

Return value

A Leaf object.

Exceptions

None.

Members

The Leaf object has members with the same names and descriptions as the arguments to the *Leaf* method.

LeafFromConstraintNames object

The LeafFromConstraintNames object can be used whenever a Leaf object is expected as an argument.

A *Leaf object* is used used to specify the items in a display group. Leaf objects are constructed as temporary objects that are used as arguments to DisplayGroup (*DisplayGroup object*) commands.

The LeafFromConstraintNames object is derived from the Leaf object.

Access

```
import displayGroupOdbToolset
```

LeafFromConstraintNames(...)

This method creates a Leaf object from a sequence of constraint objects. Leaf objects specify the items in a display group.

Path

```
LeafFromConstraintNames
```

Required arguments

name A sequence of tuples of name objects.

type A SymbolicConstant specifying the Leaf type. Possible values are TIE, SHELL_TO_SOLID_COUPLING, DISTRIBUTING_COUPLING, KINEMATIC_COUPLING, RIGID_BODY, and MPC.

Optional arguments

None.

Return value

A LeafFromConstraintNames object.

Exceptions

None.

Members

The LeafFromConstraintNames object has members with the same names and descriptions as the arguments to the *LeafFromConstraintNames* method.

In addition, the LeafFromConstraintNames object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are TIE, SHELL_TO_SOLID_COUPLING, DISTRIBUTING_COUPLING, KINEMATIC_COUPLING, RIGID_BODY, and MPC.

LeafFromDatums object

The LeafFromDatums object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromDatums object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromDatums(...)

This method creates a *Leaf* object from a sequence of datum objects. Leaf objects specify the items in a display group.

Path

```
LeafFromDatums
```

Required arguments

datumSeq A sequence of datum objects.

Optional arguments

None.

Return value

A LeafFromDatums object.

Exceptions

None.

Members

The LeafFromDatums object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromDisplayGroup object

The LeafFromDisplayGroup object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromDisplayGroup object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
import displayGroupOdbToolset
```

LeafFromDisplayGroup(...)

This method creates a *Leaf* object from a sequence of Display Group objects.

Path

```
LeafFromDisplayGroup
```

Required arguments

displayGroup A *DisplayGroupArray* object.

Optional arguments

None.

Return value

A LeafFromDisplayGroup object.

Exceptions

None.

Members

The LeafFromDisplayGroup object has members with the same names and descriptions as the arguments to the *LeafFromDisplayGroup* method.

In addition, the LeafFromDisplayGroup object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromElementLabels object

The LeafFromElementLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromElementLabels object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromElementLabels(...)

This method creates a *Leaf* object from a sequence of element labels that belong to a single part instance.

Path

```
LeafFromElementLabels
```

Required arguments

partInstanceName

A String specifying the name of the part instance to which *elementLabels* refers.

elementLabels

A sequence of Strings specifying expressions that denote element labels. The expression can be any of the following:

- An Int specifying a single element label; for example, `1`.
- A String specifying a single element label; for example, `'7'`.
- A String specifying a sequence of element labels; for example, `'3:5'` and `'3:15:3'`.

Optional arguments

None.

Return value

A LeafFromElementLabels object.

Exceptions

None.

Members

The LeafFromElementLabels object has members with the same names and descriptions as the arguments to the *LeafFromElementLabels* method.

In addition, the LeafFromElementLabels object has the following member:

leafType

A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromElementSets object

The LeafFromElementSets object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromElementSets object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromElementSets(...)

This method creates a *Leaf* object from a sequence of element sets.

Path

```
LeafFromElementSets
```

Required arguments

elementSets A sequence of Strings specifying element sets or a String specifying a single element set.

Optional arguments

None.

Return value

A LeafFromElementSets object.

Exceptions

None.

Members

The LeafFromElementSets object has members with the same names and descriptions as the arguments to the *LeafFromElementSets* method.

In addition, the LeafFromElementSets object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromElementVarRange object

The LeafFromElementVarRange object can be used whenever a Leaf object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromElementVarRange object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromElementVarRange(...)

This method creates a *Leaf* object from elements with values lying in a variable range.

Path

```
LeafFromElementVarRange
```

Required arguments

None.

Optional arguments

minimumRange
A Float specifying the minimum value for the variable range. The default value is −3.40282346639E38.

maximumRange
A Float specifying the maximum value for the variable range. The default value is 3.40282346639e+038.

insideRange
A Boolean specifying the method used to evaluate the range. If *insideRange*=ON, the range falls inside the specified minimum and maximum values. The default value is ON.

Return value

A LeafFromElementVarRange object.

Exceptions

None.

Members

The LeafFromElementVarRange object has members with the same names and descriptions as the arguments to the *LeafFromElementVarRange* method.

In addition, the LeafFromElementVarRange object has the following member:

leafType
A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromGeometry object

The LeafFromGeometry object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromGeometry object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromGeometry(...)

This method creates a Leaf object from a sequence of edge, face and cell geometry objects. Any combination of edge, face or cell arguments is allowed however the arguments must specify at least one object--it is not permissible to create an empty leaf.

Path

```
LeafFromGeometry
```

Required arguments

None.

Optional arguments

edgeSeq A sequence of geometry edges.

faceSeq A sequence of geometry faces.

cellSeq A sequence of geometry cells.

Return value

A LeafFromGeometry object.

Exceptions

If at least one of the sequences is not passed to this method: Cannot define empty leaf.

Members

The LeafFromGeometry object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromInstance object

The LeafFromInstance object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromInstance object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromInstance(...)

This method creates a Leaf object from a sequence of part instance objects.

Path

```
LeafFromInstance
```

Required arguments

instances A *PartInstance* object or a Sequence of *PartInstance* objects.

Optional arguments

None.

Return value

A LeafFromInstance object.

Exceptions

If an invalid argument is passed to this method: Cannot define empty leaf.

Members

The LeafFromInstance object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromInstanceElementLabels object

The LeafFromInstanceElementLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromInstanceElementLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromInstanceElementLabels(...)

This method creates a *Leaf* object from a sequence of Strings specifying the element labels. Leaf objects specify the items in a display group.

Path

```
LeafFromInstanceElementLabels
```

Required arguments

elementLabels A sequence of sequences specifying element labels. Each inner sequence consists of a *PartInstance* object followed by a sequence of Strings specifying element labels.

Optional arguments

None.

Return value

A LeafFromInstanceElementLabels object.

Exceptions

None.

Members

The LeafFromInstanceElementLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromInstanceNodeLabels object

The LeafFromInstanceNodeLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromInstanceNodeLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromInstanceNodeLabels(...)

This method creates a *Leaf* object from a sequence of Strings specifying the node labels. Leaf objects specify the items in a display group.

Path

```
LeafFromInstanceNodeLabels
```

Required arguments

nodeLabels
A sequence of sequences specifying node labels. Each inner sequence consists of a *PartInstance* object followed by a sequence of Strings specifying node labels.

Optional arguments

None.

Return value

A LeafFromInstanceNodeLabels object.

Exceptions

None.

Members

The LeafFromInstanceNodeLabels object has the following member:

leafType
A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromMeshElementLabels object

The LeafFromMeshElementLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromMeshElementLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromMeshElementLabels(...)

This method creates a *Leaf* object from a sequence of mesh element objects. Leaf objects specify the items in a display group.

Path

```
LeafFromMeshElementLabels
```

Required arguments

elementSeq A sequence of *MeshElement* objects specifying elements.

Optional arguments

None.

Return value

A LeafFromMeshElementLabels object.

Exceptions

None.

Members

The LeafFromMeshElementLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromMeshNodeLabels object

The LeafFromMeshNodeLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromMeshNodeLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromMeshNodeLabels(...)

This method creates a *Leaf* object from a sequence of mesh node objects. Leaf objects specify the items in a display group.

Path

```
LeafFromMeshNodeLabels
```

Required arguments

nodeSeq A sequence of *MeshNode* objects specifying nodes.

Optional arguments

None.

Return value

A LeafFromMeshNodeLabels object.

Exceptions

None.

Members

The LeafFromMeshNodeLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromMeshSurfaceSets object

The LeafFromMeshSurfaceSets object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromMeshSurfaceSets object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromMeshSurfaceSets(...)

This method creates a *Leaf* object from a sequence of surface sets.

Path

```
LeafFromMeshSurfaceSets
```

Required arguments

surfaceSets A sequence of Surface objects.

Optional arguments

None.

Return value

A LeafFromMeshSurfaceSets object.

Exceptions

None.

Members

The LeafFromMeshSurfaceSets object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromModelElemLabels object

The LeafFromModelElemLabels object can be used whenever a Leaf object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromModelElemLabels object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromModelElemLabels(...)

This method creates a *Leaf* object from a sequence of element labels spanning several part instances.

Path

```
LeafFromModelElemLabels
```

Required arguments

elementLabels A sequence of Strings specifying expressions that denote element labels per part instance in the model. Each part instance element expression is a sequence of a String specifying the part instance name and a sequence of element expressions; for example, `(('partInstance1',(1,'7','3:15;3'),), ('partInstance2','8'),))`. The element expressions can be any of the following:

- An Int specifying a single element label; for example, `1`.
- A String specifying a single element label; for example, `'7'`.
- A String specifying a sequence of element labels; for example, `'3:5'` and `'3:15:3'`.

Optional arguments

None.

Return value

A LeafFromModelElemLabels object.

Exceptions

None.

Members

The LeafFromModelElemLabels object has members with the same names and descriptions as the arguments to the *LeafFromModelElemLabels* method.

In addition, the LeafFromModelElemLabels object has the following member:

leafType

A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromModelNodeLabels object

The LeafFromModelNodeLabels object can be used whenever a Leaf object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromModelNodeLabels object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromModelNodeLabels(...)

This method creates a *Leaf* object from a sequence of node labels spanning several part instances.

Path

```
LeafFromModelNodeLabels
```

Required arguments

nodeLabels A sequence of Strings specifying expressions that denote node labels per part instance in the model. Each part instance node expression is a sequence of a String specifying the part instance name and a sequence of node expressions; for example, `(('partInstance1',(1,'7','3:15;3'),),('partInstance2','8'),))`. The node expressions can be any of the following:

- An Int specifying a single node label; for example, `1`.
- A String specifying a single node label; for example, `'7'`.
- A String specifying a sequence of node labels; for example, `'3:5'` and `'3:15:3'`.

Optional arguments

None.

Return value

A LeafFromModelNodeLabels object.

Exceptions

None.

Members

The LeafFromModelNodeLabels object has members with the same names and descriptions as the arguments to the *LeafFromModelNodeLabels* method.

In addition, the LeafFromModelNodeLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromNodeLabels object

The LeafFromNodeLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromNodeLabels object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromNodeLabels(...)

This method creates a *Leaf* object from a sequence of node labels that belong to a single part instance.

Path

```
LeafFromNodeLabels
```

Required arguments

partInstanceName A String specifying the name of the part instance to which *nodeLabels* refers.

nodeLabels A sequence of Strings specifying expressions that denote node labels. The expression can be any of the following:

- An Int specifying a single node label; for example, `1`.
- A String specifying a single node label; for example, `'7'`.
- A String specifying a sequence of node labels; for example, `'3:5'` and `'3:15:3'`.

Optional arguments

None.

Return value

A LeafFromNodeLabels object.

Exceptions

None.

Members

The LeafFromNodeLabels object has members with the same names and descriptions as the arguments to the *LeafFromNodeLabels* method.

In addition, the LeafFromNodeLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromNodeSets object

The LeafFromNodeSets object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromNodeSets object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromNodeSets(...)

This method creates a *Leaf* object from a sequence of node sets.

Path

```
LeafFromNodeSets
```

Required arguments

nodeSets A sequence of Strings specifying node sets or a String specifying a single node set.

Optional arguments

None.

Return value

A LeafFromNodeSets object.

Exceptions

None.

Members

The LeafFromNodeSets object has members with the same names and descriptions as the arguments to the *LeafFromNodeSets* method.

In addition, the LeafFromNodeSets object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromNodeVarRange object

The LeafFromNodeVarRange object can be used whenever a Leaf object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromNodeVarRange object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromNodeVarRange(...)

This method creates a *Leaf* object from nodes with values lying in a variable range.

Path

```
LeafFromNodeVarRange
```

Required arguments

None.

Optional arguments

minimumRange A Float specifying the minimum value for the variable range. The default value is −3.40282346639E38.

maximumRange A Float specifying the maximum value for the variable range. The default value is 3.40282346639e+038.

insideRange A Boolean specifying the method used to evaluate the range. If *insideRange*=ON, the range falls inside the specified minimum and maximum values. The default value is ON.

Return value

A LeafFromNodeVarRange object.

Exceptions

None.

Members

The LeafFromNodeVarRange object has members with the same names and descriptions as the arguments to the *LeafFromNodeVarRange* method.

In addition, the LeafFromNodeVarRange object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementLayups object

The LeafFromOdbElementLayups object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementLayups object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementLayups(...)

This method creates a *Leaf* object from a sequence of Strings specifying layup names. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementLayups
```

Required arguments

elementLayups A sequence of Strings specifying element layups.

Optional arguments

None.

Return value

A LeafFromOdbElementLayups object.

Exceptions

None.

Members

The LeafFromOdbElementLayups object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementMaterials object

The LeafFromOdbElementMaterials object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementMaterials object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementMaterials(...)

This method creates a *Leaf* object from a sequence of Strings specifying material names. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementMaterials
```

Required arguments

elementMaterials A sequence of Strings specifying element materials.

Optional arguments

None.

Return value

A LeafFromOdbElementMaterials object.

Exceptions

None.

Members

The LeafFromOdbElementMaterials object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementPick object

The LeafFromOdbElementPick object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementPick object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementPick(...)

This method creates a *Leaf* object from a tuple containing machine readable, compact strings defining the elements picked for each part instance. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementPick
```

Required arguments

elementPick
A sequence of tuples of the form [part name, entity count, machine readable pick strings].

Optional arguments

None.

Return value

A LeafFromOdbElementPick object.

Exceptions

None.

Members

The LeafFromOdbElementPick object has the following member:

leafType
A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementPlies object

The LeafFromOdbElementPlies object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementPlies object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementPlies(...)

This method creates a *Leaf* object from a sequence of Strings specifying ply names. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementPlies
```

Required arguments

elementPlies A sequence of Strings specifying element plies.

Optional arguments

None.

Return value

A LeafFromOdbElementPlies object.

Exceptions

None.

Members

The LeafFromOdbElementPlies object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementSections object

The LeafFromOdbElementSections object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementSections object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementSections(...)

This method creates a *Leaf* object from a sequence of Strings specifying section names. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementSections
```

Required arguments

elementSections A sequence of Strings specifying element Sections.

Optional arguments

None.

Return value

A LeafFromOdbElementSections object.

Exceptions

None.

Members

The LeafFromOdbElementSections object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbElementTypes object

The LeafFromOdbElementTypes object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbElementTypes object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbElementTypes(...)

This method creates a *Leaf* object from a sequence of Strings specifying element names. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbElementTypes
```

Required arguments

elementTypes A sequence of Strings specifying element Types.

Optional arguments

None.

Return value

A LeafFromOdbElementTypes object.

Exceptions

None.

Members

The LeafFromOdbElementTypes object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbNodePick object

The LeafFromOdbNodePick object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbNodePick object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbNodePick(...)

This method creates a *Leaf* object from a tuple containing machine readable, compact strings defining the nodes picked for each part instance. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbNodePick
```

Required arguments

nodePick A sequence of tuples of the form [part name, entity count, machine readable pick strings].

Optional arguments

None.

Return value

A LeafFromOdbNodePick object.

Exceptions

None.

Members

The LeafFromOdbNodePick object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromOdbEdgePick object

The LeafFromOdbEdgePick object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromOdbEdgePick object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromOdbEdgePick(...)

This method creates a *Leaf* object from a tuple containing machine readable, compact strings defining the edges picked for each part instance. Leaf objects specify the items in a display group.

Path

```
LeafFromOdbEdgePick
```

Required arguments

edgePick A sequence of tuples of the form [part name, entity count, machine readable pick strings].

Optional arguments

None.

Return value

A LeafFromOdbEdgePick object.

Exceptions

None.

Members

The LeafFromOdbEdgePick object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromPartElementLabels object

The LeafFromPartElementLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromPartElementLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromPartElementLabels(...)

This method creates a *Leaf* object from a sequence of Strings specifying element labels. Leaf objects specify the items in a display group.

Path

```
LeafFromPartElementLabels
```

Required arguments

part A *Part* object.

elementLabels A sequence of Strings specifying element labels.

Optional arguments

None.

Return value

A LeafFromPartElementLabels object.

Exceptions

None.

Members

The LeafFromPartElementLabels object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromPartInstance object

The LeafFromPartInstance object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromPartInstance object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromPartInstance(...)

This method creates a *Leaf* object from a list of part instance names.

Path

```
LeafFromPartInstance
```

Required arguments

partInstanceName A sequence of Strings specifying the names of the part instances.

Optional arguments

None.

Return value

A LeafFromPartInstance object.

Exceptions

None.

Members

The LeafFromPartInstance object has members with the same names and descriptions as the arguments to the *LeafFromPartInstance* method.

In addition, the LeafFromPartInstance object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromPartNodeLabels object

The LeafFromPartNodeLabels object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromPartNodeLabels object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromPartNodeLabels(...)

This method creates a *Leaf* object from a sequence of Strings specifying node labels. Leaf objects specify the items in a display group.

Path

```
LeafFromPartNodeLabels
```

Required arguments

part	A *Part* object.
nodeLabels	A sequence of Strings specifying node labels.

Optional arguments

None.

Return value

A LeafFromPartNodeLabels object.

Exceptions

None.

Members

The LeafFromPartNodeLabels object has the following member:

leafType	A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromReferencePoint object

The LeafFromReferencePoint object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromReferencePoint object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromReferencePoint(...)

This method creates a *Leaf* object from a sequence of ReferencePoint objects.

Path

```
LeafFromReferencePoint
```

Required arguments

refPtSeq A sequence of Reference Point objects.

Optional arguments

None.

Return value

A LeafFromReferencePoint object.

Exceptions

None.

Members

The LeafFromReferencePoint object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromSets object

The LeafFromSets object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromSets object is derived from the *Leaf* object.

Access

```
import displayGroupMdbToolset
```

LeafFromSets(...)

This method creates a *Leaf* object from a sequence of Set objects.

Path

```
LeafFromSets
```

Required arguments

sets A sequence of Set objects.

Optional arguments

None.

Return value

A LeafFromSets object.

Exceptions

None.

Members

The LeafFromSets object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromSurfaceSets object

The LeafFromSurfaceSets object can be used whenever a *Leaf* object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromSurfaceSets object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromSurfaceSets(...)

This method creates a *Leaf* object from a sequence of surface sets.

Path

```
LeafFromSurfaceSets
```

Required arguments

surfaceSets A sequence of Strings specifying surface sets, or a String specifying a single surface set.

Optional arguments

None.

Return value

A LeafFromSurfaceSets object.

Exceptions

None.

Members

The LeafFromSurfaceSets object has members with the same names and descriptions as the arguments to the *LeafFromSurfaceSets* method.

In addition, the LeafFromSurfaceSets object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

LeafFromSurfaceVarRange object

The LeafFromSurfaceVarRange object can be used whenever a Leaf object is expected as an argument. *Leaf* objects are used to specify the items in a display group. *Leaf* objects are constructed as temporary objects, which are then used as arguments to *DisplayGroup* commands.

The LeafFromSurfaceVarRange object is derived from the *Leaf* object.

Access

```
import displayGroupOdbToolset
```

LeafFromSurfaceVarRange(...)

This method creates a *Leaf* object from surfaces with values lying in a variable range.

Path

```
LeafFromSurfaceVarRange
```

Required arguments

None.

Optional arguments

minimumRange A Float specifying the minimum value for the variable range. The default value is −3.40282346639E38.

maximumRange A Float specifying the maximum value for the variable range. The default value is 3.40282346639e+038.

insideRange A Boolean specifying the method used to evaluate the range. If *insideRange*=ON, the range falls inside the specified minimum and maximum values. The default value is ON.

Return value

A LeafFromSurfaceVarRange object.

Exceptions

None.

Members

The LeafFromSurfaceVarRange object has members with the same names and descriptions as the arguments to the *LeafFromSurfaceVarRange* method.

In addition, the LeafFromSurfaceVarRange object has the following member:

leafType A SymbolicConstant specifying the leaf type. Possible values are EMPTY_LEAF, DEFAULT_MODEL, ALL_ELEMENTS, ALL_NODES, and ALL_SURFACES.

Display Options commands

Display options commands are used to store settings that specify how assemblies and parts are to be displayed and how annotations are to be rendered in a particular viewport.

In this section:

- *AssemblyDisplayOptions object*
- *BCDisplayOptions object*
- *ConstraintDisplayOptions object*
- *EngineeringFeatureDisplayOptions object*
- *PredefinedFieldDisplayOptions object*
- *GeometryDisplayOptions object*
- *GeometricRestrictionDisplayOptions object*
- *GraphicsInfo object*
- *GraphicsOptions object*
- *InteractionDisplayOptions object*
- *Light object*
- *LightOptions object*
- *LoadDisplayOptions object*
- *MeshDisplayOptions object*
- *OptimizationTaskDisplayOptions object*
- *PartDisplayOptions object*
- *StopConditionDisplayOptions object*
- *SymbolDisplayOptions object*
- *ViewportAnnotationOptions object*

AssemblyDisplayOptions object

The AssemblyDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport. The AssemblyDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
import assembly
session.viewports[name].assemblyDisplay
session.viewports[name].layers[name].assemblyDisplay
```

`setValues(...)`

This method modifies the AssemblyDisplayOptions object.

Required arguments

None.

Optional arguments

visibleInstances
A sequence of Strings specifying the names of the part instances that are visible in the viewport. The default value is an empty sequence.

step
A String specifying the step for which objects are to be displayed. Possible values are any valid step name. The default value is "Initial".

renderStyle
A SymbolicConstant specifying how the image in the viewport is rendered. Possible values are WIREFRAME, HIDDEN, SHADED, and FILLED. The default value is WIREFRAME.

mesh
A Boolean specifying whether the mesh is shown. The default value is OFF.

loads
A Boolean specifying whether loads are shown. The default value is OFF.

bcs
A Boolean specifying whether boundary conditions are shown. The default value is OFF.

interactions
A Boolean specifying whether interactions are shown. The default value is OFF.

constraints
A Boolean specifying whether constraints are shown. The default value is OFF.

connectors
A Boolean specifying whether connectors are shown. The default value is OFF.

cnxEndPoints
A Boolean specifying whether the connector end points are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxLocalAxes
A Boolean specifying whether the connector local coordinate system axes are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxTypeLabels
A Boolean specifying whether the connector section type labels are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxTagDisplay
A Boolean specifying whether the tag information is displayed along with the connector section type labels. This member is applicable only if *connectors*=ON and if *cnxTypeLabels*=ON. The default value is OFF.

predefinedFields
A Boolean specifying whether fields and initial conditions are shown. The default value is OFF.

visibleDisplayGroups
A sequence of *DisplayGroup* objects specifying the DisplayGroups visible in the viewport. Currently the sequence can contain a maximum of one *DisplayGroup* object. The default value is an empty sequence.

engineeringFeatures
A Boolean specifying whether to display engineering features. The default value is OFF.

renderBeamProfiles
A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor
A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

optimizationTasks
A Boolean specifying whether optimization tasks are shown. The default value is OFF.

geometricRestrictions
A Boolean specifying whether geometric restrictions are shown. The default value is OFF.

stopConditions
A Boolean specifying whether stop conditions are shown. The default value is OFF.

Return value
`None.`

Exceptions
RangeError.

Members
The AssemblyDisplayOptions object can have the following members:

bcs
A Boolean specifying whether boundary conditions are shown. The default value is OFF.

connectors
A Boolean specifying whether connectors are shown. The default value is OFF.

cnxEndPoints
A Boolean specifying whether the connector end points are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxLocalAxes

A Boolean specifying whether the connector local coordinate system axes are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxTypeLabels

A Boolean specifying whether the connector section type labels are shown. This member is applicable only if *connectors*=ON. The default value is ON.

cnxTagDisplay

A Boolean specifying whether the tag information is displayed along with the connector section type labels. This member is applicable only if *connectors*=ON and if *cnxTypeLabels*=ON. The default value is OFF.

constraints

A Boolean specifying whether constraints are shown. The default value is OFF.

engineeringFeatures

A Boolean specifying whether to display engineering features. The default value is OFF.

geometricRestrictions

A Boolean specifying whether geometric restrictions are shown. The default value is OFF.

renderBeamProfiles

A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor

A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

predefinedFields

A Boolean specifying whether fields and initial conditions are shown. The default value is OFF.

interactions

A Boolean specifying whether interactions are shown. The default value is OFF.

loads

A Boolean specifying whether loads are shown. The default value is OFF.

mesh

A Boolean specifying whether the mesh is shown. The default value is OFF.

optimizationTasks

A Boolean specifying whether optimization tasks are shown. The default value is OFF.

stopConditions

A Boolean specifying whether stop conditions are shown. The default value is OFF.

renderStyle

A SymbolicConstant specifying how the image in the viewport is rendered. Possible values are WIREFRAME, HIDDEN, SHADED, and FILLED. The default value is WIREFRAME.

bcOptions

A *BCDisplayOptions* object.

constraintOptions

A *ConstraintDisplayOptions* object.

displayGroup

A *DisplayGroup* object specifying the current display group and referring to an object in the *displayGroups* member of *Session*.

displayGroupInstances

A repository of *DisplayGroupInstance* objects.

engineeringFeatureOptions

An *EngineeringFeatureDisplayOptions* object.

predefinedFieldOptions

A *PredefinedFieldDisplayOptions* object.

geometricRestrictionOptions

A *GeometricRestrictionDisplayOptions* object.

geometryOptions

A *GeometryDisplayOptions* object.

interactionOptions

An *InteractionDisplayOptions* object.

loadOptions

A *LoadDisplayOptions* object.

meshOptions

A *MeshDisplayOptions* object.

optimizationTaskOptions

An *OptimizationTaskDisplayOptions* object.

stopConditionOptions

A *StopConditionDisplayOptions* object.

symbolOptions

A *SymbolDisplayOptions* object.

visibleInstances

A tuple of Strings specifying the names of the part instances that are visible in the viewport. The default value is an empty sequence.

step

A String specifying the step for which objects are to be displayed. Possible values are any valid step name. The default value is "Initial".

BCDisplayOptions object

The BCDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.bcs=ON
```

The BCDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.bcOptions
session.viewports[name].layers[name].assemblyDisplay.bcOptions
```

`setValues(...)`

This method modifies the BCDisplayOptions object.

Required arguments

None.

Optional arguments

displacement
A Boolean specifying whether displacement symbols are shown. The default value is ON.

velocity
A Boolean specifying whether velocity symbols are shown. The default value is ON.

acceleration
A Boolean specifying whether acceleration symbols are shown. The default value is ON.

symmetry
A Boolean specifying whether symmetry symbols are shown. The default value is ON.

antiSymmetry
A Boolean specifying whether anti- symmetry symbols are shown. The default value is ON.

temperature
A Boolean specifying whether temperature symbols are shown. The default value is ON.

porePressure
A Boolean specifying whether pore pressure symbols are shown. The default value is ON.

fluidCavityPressure
A Boolean specifying whether fluid cavity pressure symbols are shown. The default value is ON.

acousticPressure
A Boolean specifying whether acoustic pressure symbols are shown. The default value is ON.

electricPotential	A Boolean specifying whether electric potential symbols are shown. The default value is ON.
concentration	A Boolean specifying whether concentration mass diffusion symbols are shown. The default value is ON.
encastre	A Boolean specifying whether encastre symbols are shown. The default value is ON.
pinned	A Boolean specifying whether pinned symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

None.

Members

The BCDisplayOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

ConstraintDisplayOptions object

The ConstraintDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.constraints=ON
```

The ConstraintDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.constraintOptions
session.viewports[name].layers[name].assemblyDisplay.constraintOptions
```

`setValues(...)`

This method modifies the ConstraintDisplayOptions object.

Required arguments

None.

Optional arguments

constraintEquation
A Boolean specifying whether constraint equation symbols are shown. The default value is ON.

tieConstraint
A Boolean specifying whether tie constraint symbols are shown. The default value is ON.

rigidBodyConstraint
A Boolean specifying whether rigid body constraint symbols are shown. The default value is ON.

displayBodyConstraint
A Boolean specifying whether display body constraint symbols are shown. The default value is ON.

couplingConstrain
A Boolean specifying whether coupling constraint symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

None.

Members

The ConstraintDisplayOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

EngineeringFeatureDisplayOptions object

The EngineeringFeatureDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.engineeringFeatures=ON
```

The EngineeringFeatureDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.engineeringFeatureOptions
session.viewports[name].layers[name].assemblyDisplay\
.engineeringFeatureOptions
import part
session.viewports[name].layers[name].partDisplay\
.engineeringFeatureOptions
session.viewports[name].partDisplay.engineeringFeatureOptions
```

`setValues(...)`

This method modifies the EngineeringFeatureDisplayOptions object.

Required arguments

None.

Optional arguments

pointMassInertia A Boolean specifying whether point mass inertia symbols are shown. The default value is ON.

nonstructuralMass A Boolean specifying whether nonstructural mass symbols are shown. The default value is ON.

heatCapacitance A Boolean specifying whether heat capacitance symbols are shown. The default value is ON.

contourIntegral A Boolean specifying whether contour integral symbols are shown. The default value is ON.

springToGround A Boolean specifying whether spring-to-ground symbols are shown. The default value is ON.

twoPointSpring A Boolean specifying whether two-point spring symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The EngineeringFeatureDisplayOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

PredefinedFieldDisplayOptions object

The PredefinedFieldDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.predefinedFields=ON
```

The PredefinedFieldDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.predefinedFieldOptions
session.viewports[name].layers[name].assemblyDisplay\
.predefinedFieldOptions
```

`setValues(...)`

This method modifies the PredefinedFieldDisplayOptions object.

Required arguments

None.

Optional arguments

temperatureField A Boolean specifying whether temperature field symbols are shown. The default value is ON.

velocityField A Boolean specifying whether translational velocity symbols are shown. The default value is ON.

generalField A Boolean specifying whether general field symbols are shown. The default value is ON.

stressField A Boolean specifying whether stress field symbols are shown. The default value is ON.

hardeningField A Boolean specifying whether hardening field symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The PredefinedFieldDisplayOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

GeometryDisplayOptions object

The GeometryDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport. The GeometryDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.geometryOptions
session.viewports[name].layers[name].assemblyDisplay.geometryOptions
session.viewports[name].layers[name].partDisplay.geometryOptions
session.viewports[name].partDisplay.geometryOptions
```

`setValues(...)`

This method modifies the GeometryDisplayOptions object.

Required arguments

None.

Optional arguments

geometryEdgesInShaded
A Boolean specifying whether geometry edges are displayed in shaded mode. The default value is ON.

geometryHiddenEdges
A Boolean specifying whether geometry hidden edges are displayed (dotted) in hidden line mode. The default value is OFF.

geometrySilhouetteEdges
A Boolean specifying whether geometry silhouette edges are displayed. The default value is ON.

datumAxes
A Boolean specifying whether datum axes are shown. The default value is ON.

datumCoordSystems
A Boolean specifying whether datum coordinate systems are shown. The default value is ON.

datumPlanes
A Boolean specifying whether datum planes are shown. The default value is ON.

referencePointLabels
A Boolean specifying whether referencePoint labels are shown. The default value is ON.

referencePointSymbols
A Boolean specifying whether referencePoint symbols are shown. The default value is ON.

referenceRepresentation
A Boolean specifying whether geometry that belongs to the reference representation of the Part or Instance is shown. The default value is OFF.

referenceRepTranslucency A Boolean specifying whether to apply translucency to the geometry that belongs to the reference representation of the Part or Instance. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The GeometryDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

GeometricRestrictionDisplayOptions object

The GeometricRestrictionDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.geometricRestrictions=ON
```

The GeometricRestrictionDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.geometricRestrictionOptions
session.viewports[name].layers[name].assemblyDisplay\
.geometricRestrictionOptions
```

`setValues(...)`

This method modifies the GeometricRestrictionDisplayOptions object.

Required arguments

None.

Optional arguments

drillControl

A Boolean specifying whether drill control geometric restriction symbols are shown. The default value is ON.

fixedRegion

A Boolean specifying whether fixed region geometric restriction symbols are shown. The default value is ON.

frozenArea

A Boolean specifying whether frozen area geometric restriction symbols are shown. The default value is ON.

growth

A Boolean specifying whether growth geometric restriction symbols are shown. The default value is ON.

penetrationCheck

A Boolean specifying whether penetration check geometric restriction symbols are shown. The default value is ON.

shapeDemoldControl

A Boolean specifying whether demold control (shape) geometric restriction symbols are shown. The default value is ON.

designDirection

A Boolean specifying whether design direction geometric restriction symbols are shown. The default value is ON.

shapeMemberSize

A Boolean specifying whether member size (shape) geometric restriction symbols are shown. The default value is ON.

shapePlanarSymmetry A Boolean specifying whether planar symmetry (shape) geometric restriction symbols are shown. The default value is ON.

shapePointSymmetry A Boolean specifying whether point symmetry (shape) geometric restriction symbols are shown. The default value is ON.

shapeRotationalSymmetry A Boolean specifying whether rotational symmetry (shape) geometric restriction symbols are shown. The default value is ON.

stampControl A Boolean specifying whether stamp control geometric restriction symbols are shown. The default value is ON.

slideRegionControl A Boolean specifying whether slide region control geometric restriction symbols are shown. The default value is ON.

topologyCyclicSymmetry A Boolean specifying whether cyclic symmetry geometric restriction symbols are shown. The default value is ON.

topologyDemoldControl A Boolean specifying whether demold control (topology) geometric restriction symbols are shown. The default value is ON.

topologyMemberSize A Boolean specifying whether member size (topology) geometric restriction symbols are shown. The default value is ON.

topologyPlanarSymmetry A Boolean specifying whether planar symmetry (topology) geometric restriction symbols are shown. The default value is ON.

topologyPointSymmetry A Boolean specifying whether point symmetry (topology) geometric restriction symbols are shown. The default value is ON.

topologyRotationalSymmetry A Boolean specifying whether rotational symmetry (topology) geometric restriction symbols are shown. The default value is ON.

turnControl A Boolean specifying whether turn control geometric restriction symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The GeometricRestrictionDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

GraphicsInfo object

The GraphicsInfo object stores information about the graphics adapter installed in your system. The GraphicsInfo object has no constructor or methods; Abaqus creates the *GraphicsInfo* member when a session is started.

If you execute Abaqus/CAE on a remote system and display the main window locally, the *glx server* is your local machine and the *glx client* is the remote machine. This definition of client and server follows the X11 naming convention. If you execute and display Abaqus/CAE on the same machine, you will typically see identical values for both the name of the server and the name of the client.

The members are all read-only.

Access

```
session.graphicsInfo
```

Members

The GraphicsInfo object has the following members:

glVersion
A sequence of the type (Int, Int, String) specifying the three components of the OpenGL version. The sequence consists of an Int with the OpenGL major version number, an Int with the OpenGL minor version number, and a String with any additional information.

glxServerVersion
A sequence of the type (Int, Int, String) specifying the three components of the glx version of the server. The sequence consists of an Int with the glx major version number, an Int with the glx minor version number, and a String with any additional information.

glxClientVersion
A sequence of the type (Int, Int, String) specifying the three components of the of glx version of the client. The sequence consists of an Int with the glx major version number, an Int with the glx minor version number, and String with any additional information.

glVendor
A String specifying the graphics adapter vendor. On hardware accelerated systems *glVendor* specifies the vendor that manufactured the adapter. On systems without hardware acceleration *glVendor* specifies the developer of the software graphics library.

glRenderer
A String specifying the name of the rendering device or the name of the software graphics library.

glxServerVendor
A String specifying the glx developer on the server side.

glxClientVendor
A String specifying the glx developer on the client side.

GraphicsOptions object

The GraphicsOptions object stores settings that control how objects are rendered in all viewports. GraphicsOptions objects are accessed in one of two ways:

- The default graphics options. These settings are used as defaults when you start a session and by the **Defaults** button on the **Graphics Options** dialog box.
- The current graphics options.

The GraphicsOptions object has no constructor; Abaqus creates both the *defaultGraphicsOptions* and the *graphicsOptions* members when a session is started.

When you start a session, Abaqus detects the graphics hardware installed on your system and uses the `setValues` method in the environment file (`abaqus_v6.env`) to modify the members of the GraphicsOptions object. If your graphics hardware is not supported by Abaqus/CAE, or if you wish to override the default graphics options, you can modify settings in the environment file. For more information, see *Tuning graphics cards*.

Access

```
session.defaultGraphicsOptions
session.graphicsOptions
```

`setValues(...)`

This method modifies the GraphicsOptions object.

Required arguments

None.

Optional arguments

graphicsDriver

A SymbolicConstant specifying the graphics driver to use. Abaqus/CAE currently uses OpenGL exclusively so the only possible value is OPEN_GL. OPEN_GL takes advantage of graphics adapter hardware acceleration.

doubleBuffering

A Boolean specifying whether double buffering is used. The default value is ON.

Double buffering controls where Abaqus/CAE draws its graphics. When *doubleBuffering*=OFF, everything is drawn directly to the screen and on many systems you can see the progress of the drawing operations. Most users find this distracting, especially in dynamic situations such as view manipulation or animation of results. When *doubleBuffering*=ON, the drawing occurs in a separate graphics buffer that is displayed when all the drawing operations are complete. This results in a much smoother display during view changes or animation. It is recommended that you set double buffering to ON.

displayLists

A Boolean specifying whether a display list will be used to accelerate graphics performance. The default value is ON.

When *displayLists*=ON, drawing operations are recorded in a list that can be quickly replayed. This results in faster drawing on most systems but requires extra memory to record the drawing operations. In the Visualization module, display lists are only

used during view manipulations and then their use is subject to the setting of *viewManipDisplayListThreshold*.

highlightMethodHint

A sequence of SymbolicConstants specifying a hint used to modify the highlight method. Possible values are:

- HARDWARE_OVERLAY, specifying a hint of hardware overlay. The best graphics performance is achieved using hardware overlay, but not all systems and graphics adapters support hardware overlay.
- XOR, specifying a hint of XOR technique. The XOR technique uses a boolean pixel operation to simulate the drawing operations but can produce different colors depending on the color of the underlying pixels.
- SOFTWARE_OVERLAY, specifying a hint of software overlay. The software overlay method simulates the effect of hardware overlay.
- BLEND, specifying a hint of blend method. The blend method combines the color of the underlying pixel with the desired color producing an approximation of the transient graphics.

The default value is (HARDWARE_OVERLAY, XOR, SOFTWARE_OVERLAY, BLEND).

The values of this sequence are applied by Abaqus when you start a session in first to last order. The first successful value becomes the default highlight method. Not all graphics adapters support the HARDWARE_OVERLAY value and you must use the *highlightMethodHint* argument to provide an alternative.

You can use a single value to set the first element of the list, or you can use a tuple with one to four unique values. Abaqus sets any remaining elements of the tuple to unique values based on the default order.

dragMode

A SymbolicConstant specifying which rendering is used during dynamic rotations of the view. Possible values are:

- FAST, specifying a rendering mode where the image is rendered in wireframe.
- AS_IS, specifying a rendering mode where the image is rendered as is.

The default value is AS_IS.

When set to *dragMode*=FAST, a wireframe outline is drawn during view changes by rotation, pan, or zoom. When *dragMode*=AS_IS, everything displayed in the window will be drawn during view changes; however, the display may lag behind the mouse movement when the model is complex especially if you are using an older or slower system. For newer systems with graphics hardware acceleration the AS_IS setting can be accommodated without significant loss of performance.

antiAlias

A Boolean specifying whether lines will be smoothed to reduce the jagged effect of rasterization. The default value is ON.

autoFitAfterRotate

A Boolean specifying whether the model is automatically resized to fit the viewport after each view rotation. The default value is OFF.

polygonOffsetConstant

A Float specifying the offset added when drawing the faces of a polygon. The *polygonOffsetConstant* argument affects the behavior of only the OpenGL driver.

Possible values are $0.0 \le$ *polygonOffsetConstant* ≤ 100.0. The default value is platform dependent and is typically between 0.0 and 2.0.

polygonOffsetSlope

A Float specifying the factor that multiplies the slope of each line before the line is added to the vertexes of a polygon face. The *polygonOffsetSlope* argument affects the behavior of only the OpenGL driver. Possible values are $0.0 \le$ *polygonOffsetSlope* ≤ 100.0. The default value is platform dependent and is typically between 0.0 and 2.0.

printPolygonOffsetConstant

A Float specifying the offset added when drawing the faces of a polygon. *printPolygonOffsetConstant* is similar to *polygonOffsetConstant*; however, *printPolygonOffsetConstant* is used when printing and *polygonOffsetConstant* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different offset values for each driver.

printPolygonOffsetSlope

A Float specifying the factor that multiplies the slope of each line before the line is added to the vertexes of a polygon face. *printPolygonOffsetSlope* is similar to *polygonOffsetSlope*; however, *printPolygonOffsetSlope* is used when printing and *polygonOffsetSlope* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different offset values for each driver.

vertexArrays

A Boolean specifying how the three-dimensional vertices of the model are processed. When *vertexArrays*=OFF, each vertex of the model is processed separately. When *vertexArrays*=ON, the vertices are processed in large blocks resulting in faster display. Not all graphics adapters support this capability correctly. An indicator that the graphics adapters is not processing three-dimensional vertices correctly is the absence of graphics during “rubber banding” operations. For example, when dynamically dragging the radius of a circle in the Sketcher, the circle should be visible. The default value is ON.

vertexArraysInDisplayLists

A Boolean specifying whether the *vertexArrays* setting should temporarily be set to OFF when building a display list. The default value is ON.

Some graphics adapters do not properly support using vertex arrays inside a display list. Setting *vertexArraysInDisplayLists* to OFF has a smaller impact on graphics performance than setting *vertexArrays* or *displayLists* to OFF.

viewManipDisplayListThreshold

An Int specifying how large a display list may be created in order to accelerate view manipulation operations. Increasing this value when viewing large models will increase the delay before a view manipulation operation begins in order to obtain improved graphics performance during the view manipulation. If set high with a large model, the delay can be many seconds. In excessive cases, graphics memory can be exceeded and the result may be an empty display list (no visible model) for the view manipulation. This setting is treated as 0 if *displayLists*=OFF. Possible values are $0 \le$ *viewManipDisplayListThreshold* ≤ 20000. The default value is 40.

directRendering

A Boolean specifying how Abaqus renders X11 graphics operations. When *directRendering*=OFF, the graphics are rendered through the X Server. When *directRendering*=ON, the graphics operations are sent directly to the graphics

adapter producing faster displays. For maximum performance, the initial value is ON. This argument is used only when you first start Abaqus/CAE; you cannot configure *directRendering* during a session.

hardwareAcceleration

A Boolean specifying whether a hardware accelerated OpenGL graphics driver will be used on Windows platforms. The default value is ON.

When *hardwareAcceleration*=OFF, the graphics driver uses a software implementation of OpenGL that is included with the operating system. This results in slower drawing on most systems; however, you may have to use the software implementation of OpenGL if the hardware graphics driver is incompatible with Abaqus/CAE.

hardwareAcceleration is used only when you first start Abaqus/CAE on a Windows platform; you cannot configure *hardwareAcceleration* during a session.*hardwareAcceleration* is not used when you start Abaqus/CAE on an X-Windows platform and display to a Windows platform running Exceed or any other X-Windows server.

hardwareOverlay

A Boolean specifying whether hardware overlay planes will be used if available. The default value is the same value as the *hardwareOverlayAvailable* member.

When *hardwareOverlayAvailable*=OFF, it will not be possible to set *hardwareOverlay* to ON and the HARDWARE_OVERLAY highlight method will not be available. If viewports display a solid color and will not display the model, it will be necessary to inhibit hardware overlay completely by setting the ABAQUS_EMULATE_OVERLAYS environment variable (to any value) before starting Abaqus/CAE.

hardwareOverlay is used only when you first start Abaqus/CAE; you cannot configure *hardwareOverlay* during a session.

textureMapping

A Boolean specifying whether textures will be used to display contour plots. The default value is ON.

Turning off texture mapping is necessary only if viewports will not correctly display a contour plot of your model.

printTextureMapping

A Boolean specifying whether textures will be used to display contour plots. The default value is ON.

Turning off texture mapping for printing is necessary only if printed output does not correctly display a contour plot of your model. *printTextureMapping* is similar to *textureMapping*; however, *printTextureMapping* is used when printing and *textureMapping* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different settings for each driver.

backgroundStyle

A SymbolicConstant specifying the background style to be used for all viewport windows. Possible values are SOLID and GRADIENT. The default value is SOLID.

If *backgroundStyle*=SOLID, the viewport background will appear as a solid color as specified by *backgroundColor*. If *backgroundStyle*=GRADIENT, the viewport background will be drawn as a gradient beginning with the *backgroundColor* at the

top of the viewport and gradually blending to the *backgroundBottomColor* at the bottom of the viewport.

backgroundColor

A String specifying one of the two background colors for all viewport windows. The initial color is black. A list of valid color strings is in the *colors* map in the *Session* object.

backgroundBottomColor

A String specifying one of the two background colors for all viewport windows. This color is used only if *backgroundStyle* =GRADIENT. The initial color is black. A list of valid color strings is in the *colors* map in the *Session* object.

backgroundOverride

A Boolean specifying whether a viewport background style of GRADIENT can be overridden when displaying certain objects, such as sketches or XY plots. When overridden, the background will be the top color of the gradient background.

backfaceCulling

A Boolean specifying whether facets that are determined to be facing away from the viewer will be drawn. The default value is ON. backfaceCulling provides a performance enhancement when displaying solid elements where the front side of the element occludes the back side of the element. Set *backfaceCulling*=OFF if it appears that you are seeing the back side of an element and the front side is missing. You should also set *backfaceCulling*=OFF if you believe the display is not complete.

accelerateOffScreen

A Boolean specifying whether the hardware accelerated graphics driver will be used for off-screen rendering. The default value is ON if graphics hardware acceleration is available and has not been disabled via the hardwareAcceleration option, and the graphics driver supports the underlying technology. When set to OFF, an alternate (slower) technique will be used to create off-screen images. Off-screen rendering is used for Printing, Probe, and backing store (viewport refresh). Setting this value to OFF will force printed images to be rendered without hardware acceleration. This is useful when writing automated tests to produce raster images that you will want to compare across multiple machines that may have different graphics environments.

backingStore

A Boolean specifying whether a backing store will be used to refresh a viewport after a window occluding the viewport is moved or dismissed. The default value is ON.

shadersAvailable

A Boolean specifying if the graphics hardware supports the OpenGL Shading Language (GLSL).

translucencyMode

An Int specifying whether speed or accuracy is more important when drawing translucent objects. Lower values optimize for speed while higher values optimize for accuracy. The actual meaning of each setting will depend on the setting of *shadersAvailable* and the capabilities of the graphics hardware and driver. Possible values are $1 \leq$ *translucencyMode* ≤ 6. The default value is 4.

options

`None` or a GraphicsOptions object specifying the object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in the *options* member. The default value is `None`.

contourRangeTexturePrecision A Float specifying a tolerance used when computing the appropriate scale for transforming result (contour) values to texture values. When set too low the 'out of range' colors may be incorrectly shown for values near the range limits. The default value is 0.5×10^{-5}.

Return value

`None.`

Exceptions

RangeError.

Members

The GraphicsOptions object can have the following members:

backgroundOverride A Boolean specifying whether a viewport background style of GRADIENT can be overridden when displaying certain objects, such as sketches or XY plots. When overridden, the background will be the top color of the gradient background.

backfaceCulling A Boolean specifying whether facets that are determined to be facing away from the viewer will be drawn. The default value is ON. backfaceCulling provides a performance enhancement when displaying solid elements where the front side of the element occludes the back side of the element. Set *backfaceCulling*=OFF if it appears that you are seeing the back side of an element and the front side is missing. You should also set *backfaceCulling*=OFF if you believe the display is not complete.

graphicsDriver A SymbolicConstant specifying the graphics driver to use. Abaqus/CAE currently uses OpenGL exclusively so the only possible value is OPEN_GL. OPEN_GL takes advantage of graphics adapter hardware acceleration.

doubleBuffering A Boolean specifying whether double buffering is used. The default value is ON.

Double buffering controls where Abaqus/CAE draws its graphics. When *doubleBuffering*=OFF, everything is drawn directly to the screen and on many systems you can see the progress of the drawing operations. Most users find this distracting, especially in dynamic situations such as view manipulation or animation of results. When *doubleBuffering*=ON, the drawing occurs in a separate graphics buffer that is displayed when all the drawing operations are complete. This results in a much smoother display during view changes or animation. It is recommended that you set double buffering to ON.

displayLists A Boolean specifying whether a display list will be used to accelerate graphics performance. The default value is ON.

When *displayLists*=ON, drawing operations are recorded in a list that can be quickly replayed. This results in faster drawing on most systems but requires extra memory to record the drawing operations. In the Visualization module, display lists are only used during view manipulations and then their use is subject to the setting of *viewManipDisplayListThreshold*.

dragMode

A SymbolicConstant specifying which rendering is used during dynamic rotations of the view. Possible values are:

- FAST, specifying a rendering mode where the image is rendered in wireframe.
- AS_IS, specifying a rendering mode where the image is rendered as is.

The default value is AS_IS.

When set to *dragMode*=FAST, a wireframe outline is drawn during view changes by rotation, pan, or zoom. When *dragMode*=AS_IS, everything displayed in the window will be drawn during view changes; however, the display may lag behind the mouse movement when the model is complex especially if you are using an older or slower system. For newer systems with graphics hardware acceleration the AS_IS setting can be accommodated without significant loss of performance.

antiAlias

A Boolean specifying whether lines will be smoothed to reduce the jagged effect of rasterization. The default value is ON.

autoFitAfterRotate

A Boolean specifying whether the model is automatically resized to fit the viewport after each view rotation. The default value is OFF.

polygonOffsetConstant

A Float specifying the offset added when drawing the faces of a polygon. The *polygonOffsetConstant* argument affects the behavior of only the OpenGL driver. Possible values are $0.0 \leq$ *polygonOffsetConstant* ≤ 100.0. The default value is platform dependent and is typically between 0.0 and 2.0.

polygonOffsetSlope

A Float specifying the factor that multiplies the slope of each line before the line is added to the vertexes of a polygon face. The *polygonOffsetSlope* argument affects the behavior of only the OpenGL driver. Possible values are $0.0 \leq$ *polygonOffsetSlope* ≤ 100.0. The default value is platform dependent and is typically between 0.0 and 2.0.

printPolygonOffsetConstant

A Float specifying the offset added when drawing the faces of a polygon. *printPolygonOffsetConstant* is similar to *polygonOffsetConstant*; however, *printPolygonOffsetConstant* is used when printing and *polygonOffsetConstant* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different offset values for each driver.

printPolygonOffsetSlope

A Float specifying the factor that multiplies the slope of each line before the line is added to the vertexes of a polygon face. *printPolygonOffsetSlope* is similar to *polygonOffsetSlope*; however, *printPolygonOffsetSlope* is used when printing and *polygonOffsetSlope* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different offset values for each driver.

vertexArrays

A Boolean specifying how the three-dimensional vertices of the model are processed. When *vertexArrays*=OFF, each vertex of the model is processed separately. When *vertexArrays*=ON, the vertices are processed in large blocks resulting in faster display. Not all graphics adapters support this capability correctly. An indicator that the graphics adapters is not processing three-dimensional vertices correctly is

the absence of graphics during “rubber banding” operations. For example, when dynamically dragging the radius of a circle in the Sketcher, the circle should be visible. The default value is ON.

vertexArraysInDisplayLists

A Boolean specifying whether the *vertexArrays* setting should temporarily be set to OFF when building a display list. The default value is ON.

Some graphics adapters do not properly support using vertex arrays inside a display list. Setting *vertexArraysInDisplayLists* to OFF has a smaller impact on graphics performance than setting *vertexArrays* or *displayLists* to OFF.

viewManipDisplayListThreshold

An Int specifying how large a display list may be created in order to accelerate view manipulation operations. Increasing this value when viewing large models will increase the delay before a view manipulation operation begins in order to obtain improved graphics performance during the view manipulation. If set high with a large model, the delay can be many seconds. In excessive cases, graphics memory can be exceeded and the result may be an empty display list (no visible model) for the view manipulation. This setting is treated as 0 if *displayLists*=OFF. Possible values are $0 \leq$ *viewManipDisplayListThreshold* ≤ 20000. The default value is 40.

directRendering

A Boolean specifying how Abaqus renders X11 graphics operations. When *directRendering*=OFF, the graphics are rendered through the X Server. When *directRendering*=ON, the graphics operations are sent directly to the graphics adapter producing faster displays. For maximum performance, the initial value is ON. This argument is used only when you first start Abaqus/CAE; you cannot configure *directRendering* during a session.

hardwareAcceleration

A Boolean specifying whether a hardware accelerated OpenGL graphics driver will be used on Windows platforms. The default value is ON.

When *hardwareAcceleration*=OFF, the graphics driver uses a software implementation of OpenGL that is included with the operating system. This results in slower drawing on most systems; however, you may have to use the software implementation of OpenGL if the hardware graphics driver is incompatible with Abaqus/CAE.

hardwareAcceleration is used only when you first start Abaqus/CAE on a Windows platform; you cannot configure *hardwareAcceleration* during a session.*hardwareAcceleration* is not used when you start Abaqus/CAE on an X-Windows platform and display to a Windows platform running Exceed or any other X-Windows server.

hardwareOverlay

A Boolean specifying whether hardware overlay planes will be used if available. The default value is the same value as the *hardwareOverlayAvailable* member.

When *hardwareOverlayAvailable*=OFF, it will not be possible to set *hardwareOverlay* to ON and the HARDWARE_OVERLAY highlight method will not be available. If viewports display a solid color and will not display the model, it will be necessary to inhibit hardware overlay completely by setting the ABAQUS_EMULATE_OVERLAYS environment variable (to any value) before starting Abaqus/CAE.

hardwareOverlay is used only when you first start Abaqus/CAE; you cannot configure *hardwareOverlay* during a session.

textureMapping

A Boolean specifying whether textures will be used to display contour plots. The default value is ON.

Turning off texture mapping is necessary only if viewports will not correctly display a contour plot of your model.

printTextureMapping

A Boolean specifying whether textures will be used to display contour plots. The default value is ON.

Turning off texture mapping for printing is necessary only if printed output does not correctly display a contour plot of your model. *printTextureMapping* is similar to *textureMapping*; however, *printTextureMapping* is used when printing and *textureMapping* is used for display. Some systems, especially Windows, use different OpenGL drivers for printing and display, and you may have to use different settings for each driver.

backgroundStyle

A SymbolicConstant specifying the background style to be used for all viewport windows. Possible values are SOLID and GRADIENT. The default value is SOLID.

If *backgroundStyle*=SOLID, the viewport background will appear as a solid color as specified by *backgroundColor*. If *backgroundStyle*=GRADIENT, the viewport background will be drawn as a gradient beginning with the *backgroundColor* at the top of the viewport and gradually blending to the *backgroundBottomColor* at the bottom of the viewport.

accelerateOffScreen

A Boolean specifying whether the hardware accelerated graphics driver will be used for off-screen rendering. The default value is ON if graphics hardware acceleration is available and has not been disabled via the hardwareAcceleration option, and the graphics driver supports the underlying technology. When set to OFF, an alternate (slower) technique will be used to create off-screen images. Off-screen rendering is used for Printing, Probe, and backing store (viewport refresh). Setting this value to OFF will force printed images to be rendered without hardware acceleration. This is useful when writing automated tests to produce raster images that you will want to compare across multiple machines that may have different graphics environments.

backingStore

A Boolean specifying whether a backing store will be used to refresh a viewport after a window occluding the viewport is moved or dismissed. The default value is ON.

highlightMethod

A SymbolicConstant specifying the highlight method. For the GraphicsOptions object, possible values of the member are HARDWARE_OVERLAY, XOR, SOFTWARE_OVERLAY, and BLEND.

hardwareOverlayAvailable

A Boolean specifying if the graphics hardware supports hardware overlay.

shadersAvailable

A Boolean specifying if the graphics hardware supports the OpenGL Shading Language (GLSL).

translucencyMode

An Int specifying whether speed or accuracy is more important when drawing translucent objects. Lower values optimize for speed while higher values optimize for accuracy. The actual meaning of each setting will depend on the setting of

shadersAvailable and the capabilities of the graphics hardware and driver. Possible values are $1 \leq$ *translucencyMode* ≤ 6. The default value is 4.

contourRangeTexturePrecision

A Float specifying a tolerance used when computing the appropriate scale for transforming result (contour) values to texture values. When set too low the 'out of range' colors may be incorrectly shown for values near the range limits. The default value is 0.5×10^{-5}.

options

`None` or a GraphicsOptions object specifying the object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in the *options* member. The default value is `None`.

highlightMethodHint

A tuple of SymbolicConstants specifying a hint used to modify the highlight method. Possible values are:

- HARDWARE_OVERLAY, specifying a hint of hardware overlay. The best graphics performance is achieved using hardware overlay, but not all systems and graphics adapters support hardware overlay.
- XOR, specifying a hint of XOR technique. The XOR technique uses a boolean pixel operation to simulate the drawing operations but can produce different colors depending on the color of the underlying pixels.
- SOFTWARE_OVERLAY, specifying a hint of software overlay. The software overlay method simulates the effect of hardware overlay.
- BLEND, specifying a hint of blend method. The blend method combines the color of the underlying pixel with the desired color producing an approximation of the transient graphics.

The default value is (HARDWARE_OVERLAY, XOR, SOFTWARE_OVERLAY, BLEND).

The values of this sequence are applied by Abaqus when you start a session in first to last order. The first successful value becomes the default highlight method. Not all graphics adapters support the HARDWARE_OVERLAY value and you must use the *highlightMethodHint* argument to provide an alternative.

You can use a single value to set the first element of the list, or you can use a tuple with one to four unique values. Abaqus sets any remaining elements of the tuple to unique values based on the default order.

backgroundColor

A String specifying one of the two background colors for all viewport windows. The initial color is black. A list of valid color strings is in the *colors* map in the *Session* object.

backgroundBottomColor

A String specifying one of the two background colors for all viewport windows. This color is used only if *backgroundStyle* =GRADIENT. The initial color is black. A list of valid color strings is in the *colors* map in the *Session* object.

InteractionDisplayOptions object

The InteractionDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.interactions=ON
```

The InteractionDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.interactionOptions
session.viewports[name].layers[name].assemblyDisplay\
.interactionOptions
```

`setValues(...)`

This method modifies the InteractionDisplayOptions object.

Required arguments

None.

Optional arguments

surfaceContact
A Boolean specifying whether surface contact symbols are shown. The default value is ON.

selfContact
A Boolean specifying whether self contact symbols are shown. The default value is ON.

elasticFoundation
A Boolean specifying whether elastic foundation symbols are shown. The default value is ON.

actuatorSensor
A Boolean specifying whether actuator/sensor symbols are shown. The default value is ON.

radiationAmbient
A Boolean specifying whether surface radiation-to-ambient symbols are shown. The default value is ON.

filmCondition
A Boolean specifying whether surface film condition symbols are shown. The default value is ON.

concentratedRadiationToAmbient
A Boolean specifying whether concentrated radiation-to-ambient symbols are shown. The default value is ON.

concentratedFilmCondition
A Boolean specifying whether concentrated film condition symbols are shown. The default value is ON.

Return value
`None.`

Exceptions
RangeError.

Members
The InteractionDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

Light object

The Light object stores settings that control how objects are lit when the *renderStyle* is set to SHADED.

The Light object has no constructor; Abaqus creates them as part of the *defaultLightOptions* and the *lightOptions* objects when a session is started.

Access

```
session.defaultLightOptions.lights[i]
session.viewports[name].lightOptions.lights[i]
```

`setValues(...)`

This method modifies the Light object.

Required arguments

None.

Optional arguments

enabled
A Boolean specifying whether the light is on or off. The default value is OFF.

type
A SymbolicConstant specifying how the effect of the light should be calculated. Possible values are:

- DIRECTIONAL, specifying a constant vector should be used for the direction from the light to a vertex.
- POINT, specifying the vector from the light to each vertex should be calculated.

The default value is DIRECTIONAL.

When set to *type* =DIRECTIONAL, a constant vector is used for the direction from the light to a vertex. When *type* =POINT, the result is more realistic because the actual vector from the light to each vertex is calculated. However, overall performance is decreased.

latitude
A Float specifying the altitude of the light above or below the camera. Possible values are -90.0 $\leq$ *latitude* $\leq$ 90.0. The default value is 0.0.

longitude
A Float specifying the east/west position of the light to the left or right of the camera. Possible values are -90.0 $\leq$ *longitude* $\leq$ 90.0. The default value is 0.0.

diffuseColor
A String specifying the color of the light added to the scene by this light source. The initial value is 70% gray. A list of valid color strings is in the *colors* map in the *session* object.

specularColor
A String specifying the color of the specular highlights created by this light source. The initial value is 36% gray. A list of valid color strings is in the *colors* map in the *session* object.

Return value

None.

Exceptions

RangeError.

Members

The Light object has members with the same names and descriptions as the arguments to the *setValues* method.

LightOptions object

The LightOptions object stores settings that control how objects are lit when the *renderStyle* is set to SHADED. LightOptions objects are accessed in one of two ways:

- The default light options. These settings are used as defaults when you start a session and by the **Defaults** button on the **Light Options** dialog box.
- The light options associated with a particular viewport.

The LightOptions object has no constructor; Abaqus creates the *defaultLightOptions* member when a session is started. When a new viewport is created, the *lightOptions* member is copied from the current viewport.

Access

```
session.defaultLightOptions
session.viewports[name].lightOptions
```

`setValues(...)`

This method modifies the LightOptions object.

Required arguments

None.

Optional arguments

shading

A SymbolicConstant specifying the shading technique that should be used to fill facets for geometric bodies when *renderStyle* =SHADED. The lighting of facets for meshed geometry is not affected by this setting. Possible values are:

- GOURAUD, specifying that lighting values should be computed for each corner of a facet and interpolated to fill the remainder of the facet.
- PHONG, specifying that lighting values should be computed for each pixel of a facet.

The default value is GOURAUD.

When set to *shading* =GOURAUD, lighting values are computed for each corner of a facet and the lighting for the remainder of the facet is interpolated from the corner values. When *shading* =PHONG, lighting values are computed for each pixel of a facet.



Note:

The shading technique can only be set to PHONG when the *shadersAvailable* member of *GraphicsOptions* is True.

Printed output will only contain Phong shading if the output format is raster and the *accelerateOffScreen* member of *GraphicsOptions* is ON.

The *renderStyle* setting is part of *AssemblyDisplayOptions*, *PartDisplayOptions*, and various other options objects not listed here since they are generally used for meshed geometry.

viewpoint

A SymbolicConstant specifying how specular highlights should be calculated. Possible values are:

- INFINITE, specifying that a constant vector should be used for the direction from the camera to a vertex when computing specular highlights.
- LOCAL, specifying that the vector from the camera to each vertex should be calculated when computing specular highlights.

The default value is INFINITE.

When set to *viewpoint* =INFINITE, a constant vector is used for the direction from the camera (viewpoint) to a vertex. When *viewpoint* =LOCAL, the result is more realistic because the actual vector from the camera to each vertex is calculated. However, overall performance is decreased.

ambientColor

A String specifying the light applied evenly to the entire scene independent of any individual light. The initial value is 20% gray. A list of valid color strings is in the *colors* map in the *session* object.

materialShininess

A Float specifying the degree to which specular reflection is focused. The higher the *materialShininess* argument, the more focused the specular highlight. Possible values are $0.0 \leq$ *polygonOffsetConstant* ≤ 128.0. The default value is 105.0.

Return value

`None.`

Exceptions

RangeError.

Members

The LightOptions object has the following members:

shading

A SymbolicConstant specifying the shading technique that should be used to fill facets for geometric bodies when *renderStyle* =SHADED. The lighting of facets for meshed geometry is not affected by this setting. Possible values are:

- GOURAUD, specifying that lighting values should be computed for each corner of a facet and interpolated to fill the remainder of the facet.
- PHONG, specifying that lighting values should be computed for each pixel of a facet.

The default value is GOURAUD.

When set to *shading* =GOURAUD, lighting values are computed for each corner of a facet and the lighting for the remainder of the facet is interpolated from the corner values. When *shading* =PHONG, lighting values are computed for each pixel of a facet.



Note:

The shading technique can only be set to PHONG when the *shadersAvailable* member of *GraphicsOptions* is True.

Printed output will only contain Phong shading if the output format is raster and the *accelerateOffScreen* member of *GraphicsOptions* is ON.

The *renderStyle* setting is part of *AssemblyDisplayOptions*, *PartDisplayOptions*, and various other options objects not listed here since they are generally used for meshed geometry.

viewpoint

A SymbolicConstant specifying how specular highlights should be calculated. Possible values are:

- INFINITE, specifying that a constant vector should be used for the direction from the camera to a vertex when computing specular highlights.
- LOCAL, specifying that the vector from the camera to each vertex should be calculated when computing specular highlights.

The default value is INFINITE.

When set to *viewpoint* =INFINITE, a constant vector is used for the direction from the camera (viewpoint) to a vertex. When *viewpoint* =LOCAL, the result is more realistic because the actual vector from the camera to each vertex is calculated. However, overall performance is decreased.

materialShininess

A Float specifying the degree to which specular reflection is focused. The higher the *materialShininess* argument, the more focused the specular highlight. Possible values are $0.0 \le$ *polygonOffsetConstant* ≤ 128.0. The default value is 105.0.

lights

A *LightArray* object of length 8.

ambientColor

A String specifying the light applied evenly to the entire scene independent of any individual light. The initial value is 20% gray. A list of valid color strings is in the *colors* map in the *session* object.

LoadDisplayOptions object

The LoadDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.loads=ON
```

The LoadDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.loadOptions
session.viewports[name].layers[name].assemblyDisplay.loadOptions
```

`setValues(...)`

This method modifies the LoadDisplayOptions object.

Required arguments

None.

Optional arguments

concentratedForce
A Boolean specifying whether concentrated force symbols are shown. The default value is ON.

moment
A Boolean specifying whether moment symbols are shown. The default value is ON.

pressure
A Boolean specifying whether pressure symbols are shown. The default value is ON.

pipePressure
A Boolean specifying whether pipe pressure symbols are shown. The default value is ON.

bodyForce
A Boolean specifying whether body force symbols are shown. The default value is ON.

lineLoad
A Boolean specifying whether line load symbols are shown. The default value is ON.

gravity
A Boolean specifying whether gravity symbols are shown. The default value is ON.

boltLoad
A Boolean specifying whether bolt load symbols are shown. The default value is ON.

pegLoad
A Boolean specifying whether PEG load symbols are shown. The default value is ON.

connectorForce
A Boolean specifying whether connector force symbols are shown. The default value is ON.

connectorMoment
A Boolean specifying whether connector moment symbols are shown. The default value is ON.

inertiaRelief
A Boolean specifying whether inertia relief symbols are shown. The default value is ON.

rotationalIntertiaLoad
A Boolean specifying whether rotational inertia load symbols are shown. The default value is ON.

coriolisForce
A Boolean specifying whether coriolis force symbols are shown. The default value is ON.

bodyHeatFlux
A Boolean specifying whether body heat flux symbols are shown. The default value is ON.

surfaceHeatFlux
A Boolean specifying whether surface heat flux symbols are shown. The default value is ON.

concentratedHeatFlux
A Boolean specifying whether concentrated heat flux symbols are shown. The default value is ON.

concentratedPoreFluid
A Boolean specifying whether concentrated pore fluid symbols are shown. The default value is ON.

surfacePoreFluid
A Boolean specifying whether surface pore fluid symbols are shown. The default value is ON.

hydroFluidFlow
A Boolean specifying whether hydro fluid flow symbols are shown. The default value is ON.

concentratedCharge
A Boolean specifying whether concentrated charge symbols are shown. The default value is ON.

concentratedCurrent
A Boolean specifying whether concentrated current symbols are shown. The default value is ON.

surfaceCharge
A Boolean specifying whether surface charge symbols are shown. The default value is ON.

surfaceCurrent
A Boolean specifying whether surface current symbols are shown. The default value is ON.

bodyCharge
A Boolean specifying whether body charge symbols are shown. The default value is ON.

bodyCurrent A Boolean specifying whether body current symbols are shown. The default value is ON.

inwardVolAccel A Boolean specifying whether inward volume acceleration symbols are shown. The default value is ON.

bodyConcentrationFlux A Boolean specifying whether body concentration flux symbols are shown. The default value is ON.

surfaceConcentrationFlux A Boolean specifying whether surface concentration flux symbols are shown. The default value is ON.

concentratedConcentrationFlux A Boolean specifying whether concentrated concentration flux symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The LoadDisplayOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

MeshDisplayOptions object

The MeshDisplayOptions object stores settings that specify how the assembly is displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.mesh=ON
```

The MeshDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.meshOptions
session.viewports[name].layers[name].assemblyDisplay.meshOptions
session.viewports[name].layers[name].partDisplay.meshOptions
session.viewports[name].partDisplay.meshOptions
```

`setValues(...)`

This method modifies the MeshDisplayOptions object.

Required arguments

None.

Optional arguments

nodeLabels
A Boolean specifying whether node labels are shown. The default value is OFF.

elementLabels
A Boolean specifying whether element labels are shown. The default value is OFF.

meshVisibleEdges
A SymbolicConstant specifying how the mesh's edges are drawn. Possible values are:

- ALL, specifying that all edges are shown.
- EXTERIOR, specifying that only exterior edges are shown.
- FEATURE, specifying that edges are shown based on *featureAngle*.
- FREE, specifying that only free edges are shown.
- NONE, specifying that no display edges are shown.

The default value is EXTERIOR.

featureAngle
A Float specifying the angle in degrees to be used in calculating a feature edge plot. Possible values are $0 \leq featureAngle \leq 90$. The default value is 20.0.

meshEdgesInShaded
A Boolean specifying whether mesh edges are displayed in shaded mode. The default value is ON.

meshTechnique
A Boolean specifying whether the regions of the assembly will be color coded based on the meshing technique assigned to the regions. This argument is ignored in partDisplay. The default value is OFF.

seeds
A Boolean specifying whether seeds are shown. This argument is ignored in partDisplay. The default value is OFF.

Return value
`None.`

Exceptions
RangeError.

Members
The MeshDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

OptimizationTaskDisplayOptions object

The OptimizationTaskDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.optimizationTasks=ON
```

The OptimizationTaskDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.optimizationTaskOptions
session.viewports[name].layers[name].assemblyDisplay\
.optimizationTaskOptions
```

`setValues(...)`

This method modifies the OptimizationTaskDisplayOptions object.

Required arguments

None.

Optional arguments

topologyTask A Boolean specifying whether topology task symbols are shown. The default value is ON.

shapeTask A Boolean specifying whether shape task symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The OptimizationTaskDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

PartDisplayOptions object

The PartDisplayOptions object stores settings that specify how parts are to be displayed in a particular viewport. The PartDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].layers[name].partDisplay
import part
session.viewports[name].partDisplay
```

setValues(...)

This method modifies the PartDisplayOptions object.

Required arguments

None.

Optional arguments

renderStyle
A SymbolicConstant specifying how the image in the viewport is rendered. Possible values are WIREFRAME, HIDDEN, and SHADED. The default value is WIREFRAME.

visibleDisplayGroups
A sequence of *DisplayGroup* objects specifying the DisplayGroups visible in the viewport. Currently the sequence can contain a maximum of one *DisplayGroup* object. The default value is an empty sequence.

engineeringFeatures
A Boolean specifying whether engineering features are shown. The default value is OFF.

renderBeamProfiles
A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor
A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

Return value

`None.`

Exceptions

RangeError.

Members

The PartDisplayOptions object can have the following members:

engineeringFeatures

A Boolean specifying whether engineering features are shown. The default value is OFF.

renderBeamProfiles

A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor

A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

mesh

A Boolean specifying whether the mesh should be displayed.

renderStyle

A SymbolicConstant specifying how the image in the viewport is rendered. Possible values are WIREFRAME, HIDDEN, and SHADED. The default value is WIREFRAME.

displayGroup

A *DisplayGroup* object specifying the current display group and referring to an object in the *displayGroups* member of *Session*.

displayGroupInstances

A repository of *DisplayGroupInstance* objects.

engineeringFeatureOptions

An *EngineeringFeatureDisplayOptions* object.

geometryOptions

A *GeometryDisplayOptions* object.

meshOptions

A *MeshDisplayOptions* object.

StopConditionDisplayOptions object

The StopConditionDisplayOptions object stores settings that specify how assemblies are to be displayed in a particular viewport when

```
session.viewports[name].assemblyDisplay.stopConditions=ON
```

The StopConditionDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.stopConditionOptions
session.viewports[name].layers[name].assemblyDisplay\
.stopConditionOptions
```

`setValues(...)`

This method modifies the StopConditionDisplayOptions object.

Required arguments

None.

Optional arguments

localStopCondition	A Boolean specifying whether local stop condition symbols are shown. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The StopConditionDisplayOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

SymbolDisplayOptions object

The SymbolDisplayOptions object stores settings that specify how the assembly is displayed in a particular viewport. The SymbolDisplayOptions object has no constructor. When you create a new viewport, the settings are copied from the current viewport.

Access

```
session.viewports[name].assemblyDisplay.symbolOptions
session.viewports[name].layers[name].assemblyDisplay.symbolOptions
```

setValues(...)

This method modifies the SymbolDisplayOptions object.

Required arguments

None.

Optional arguments

otherSymbolSize

An Int specifying the size of the scalar attribute symbols. Possible values are 1 ≤*scalarSymbolSize*≤ 30. The default value is 12.

arrowSymbolSize

An Int specifying the size of the vector and tensor attribute symbols. Possible values are 1 ≤*vectorSymbolSize*≤ 30. The default value is 12.

faceSymbolDensity

An Int specifying the relative density of the attribute symbols drawn on geometric faces. Possible values are 1 ≤*faceSymbolDensity*≤ 10. The default value is 5.

edgeSymbolDensity

An Int specifying the relative density of the attribute symbols drawn on geometric edges. Possible values are 1 ≤*edgeSymbolDensity*≤ 10. The default value is 5.

meshSymbolFraction

A Float specifying the fraction of the attribute symbols drawn on orphan mesh regions. Possible values are 0.0 ≤*meshSymbolFraction*≤ 1.0. The default value is 1.0.

showFields

A Boolean specifying whether symbols should be scaled based on analytical field value. The default value is ON.

Return value

None.

Exceptions

None.

Members

The SymbolDisplayOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

ViewportAnnotationOptions object

The ViewportAnnotationOptions object stores settings that control how annotations are rendered in a particular viewport. ViewportAnnotationOptions objects are accessed in one of two ways:

- The default viewport annotations. These settings are used as defaults when other *viewportAnnotationOptions* members are created and can be set to customize user preferences.
- The viewport annotations associated with a particular viewport.

The ViewportAnnotationOptions object has no constructor; Abaqus creates the *defaultViewportAnnotationOptions* member when a session is started. When a new viewport is created, the settings are copied from the current viewport.

Access

```
session.defaultViewportAnnotationOptions
session.viewports[name].viewportAnnotationOptions
```

`setValues(...)`

This method modifies the ViewportAnnotationOptions object.

Required arguments

None.

Optional arguments

triad

A Boolean specifying whether the view orientation triad is shown. The default value is ON.

triadPosition

A pair of Ints specifying the position of the view orientation triad as a percentage of the viewport size. Possible values are $(0, 0) \le$ *triadPosition* $\le (100, 100)$. The default value is (6, 12).

triadColor

A String specifying the color of the view orientation triad. Possible values are any valid color. The default value is "White".

triadLabels

A SymbolicConstant specifying how the view orientation triad is labeled. Possible values are:

- NUMBERS, specifying the label axes 1, 2, 3.
- LETTERS, specifying the label axes X, Y, Z.

The default value is NUMBERS.

triadFont

A String specifying the font of the view orientation triad labels. Possible values are any valid font. The default value is "-*-verdana-bold-r-normal--120-*".

triadSize

An Int specifying the length of each triad axis as a percentage of the viewport size. Possible values are $1 \le$ *legendDecimalPlaces* ≤ 50. The default value is 4.

legend
A Boolean specifying whether the legend is shown. The default value is ON.

legendMinMax
A Boolean specifying whether the minimum and maximum values for *X–Y* and contour plots are shown. The default value is OFF.

legendBox
A Boolean specifying whether the box around the legend is shown. The default value is ON.

legendDecimalPlaces
An Int specifying the number of decimal places to display in the legend. Possible values are $0 \leq$ *legendDecimalPlaces* ≤ 9. The default value is 3.

legendPosition
A pair of Ints specifying the position of the legend as a percentage of the viewport size. Possible values are (0, 0)$\leq$ *legendPosition* $\leq$ (100, 100). The default value is (2, 98).

legendFont
A String specifying the font of the legend labels. Possible values are any valid font. The default value is "-*-verdana-medium-r-*-*-*-120-*-*-*-*-iso8859-1".

legendTextColor
A String specifying the color of the legend. Possible values are any valid color. The default value is "White".

legendBackgroundStyle
A SymbolicConstant specifying the legend background style. Possible values are MATCH, TRANSPARENT, and OTHER. The default value is TRANSPARENT.

The default color when OTHER is specified is the background color (black).

legendBackgroundColor
A String specifying the color of the legend background. Possible values are any valid color. The initial value matches the viewport background.

title
A Boolean specifying whether the title block is shown. The default value is ON.

titleBox
A Boolean specifying whether the box around the title block is shown. The default value is OFF.

titlePosition
A pair of Ints specifying the position of the title block as a percentage of the viewport size. Possible values are (0, 0) $\leq$ *titlePosition* $\leq$ (100, 100). The default value is (13, 20).

titleFont
A String specifying the font of the title. Possible values are any valid font. The default value is "-*-verdana-medium-r-*-*-*-120-*-*-*-*-iso8859-1".

titleTextColor
A String specifying the color of the title. Possible values are any valid color. The default value is "White".

titleBackgroundStyle
A SymbolicConstant specifying the title block background style. Possible values are MATCH, TRANSPARENT, and OTHER. The default value is TRANSPARENT.

The default color when OTHER is specified is the background color (black).

titleBackgroundColor
A String specifying the color of the title block background. Possible values are any valid color. The initial value matches the viewport background.

state
A Boolean specifying whether the state block is shown. The default value is ON.

stateBox
A Boolean specifying whether the box around the state block is shown. The default value is OFF.

statePosition
A pair of Ints specifying the position of the state block as a percentage of the viewport size. Possible values are (0, 0) $\leq$ *statePosition* $\leq$ (100, 100). The default value is (13, 12).

stateFont
A String specifying the font of the state label. Possible values are any valid font. The default value is "-*-verdana-medium-r-*-*-*-120-*-*-*-*-iso8859-1".

stateTextColor
A String specifying the color of the state block label. Possible values are any valid color. The default value is "White".

stateBackgroundStyle
A SymbolicConstant specifying the state block background style. Possible values are MATCH, TRANSPARENT, and OTHER. The default value is TRANSPARENT.

The default color when OTHER is specified is the background color (black).

stateBackgroundColor
A String specifying the color of the state block background. Possible values are any valid color. The initial value matches the viewport background.

compass
A Boolean specifying whether the 3D Compass is shown. The default value is ON.

compassScale
A Float specifying the relative size of the 3D Compass in the viewport. Possible values are 0.5 $\leq$ *compassScale* $\leq$ 2.0. The default value is 0.8.

compassPrivilegedPlane
A SymbolicConstant specifying the plane that will be used for the base of the 3D Compass. Possible values are XYPLANE, XZPLANE, and YZPLANE. The default value is XZPLANE.

Return value
`None.`

Exceptions
RangeError.

Members
The ViewportAnnotationOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

Edit mesh commands

Edit mesh commands are used to edit an orphan mesh part or part instance or an Abaqus native mesh.

In this section:

- *Assembly object*
- *Part object*
- *MeshEditOptions object*

Assembly object

The following commands operate on Assembly objects. For more information about the Assembly object, see *Assembly object*.

Access

```
import meshEdit
```

collapseMeshEdge(...)

This method collapses an edge of a quadrilateral or triangular element of a part instance.

Required arguments

edge A single *MeshEdge* object specifying the element edge to collapse.

collapseMethod A SymbolicConstant specifying the method used to collapse the edge. Possible values are FORWARD, REVERSE, and AVERAGE.

Optional arguments

None.

Return value

None.

Exceptions

None.

combineElement(...)

This method combines two triangular elements of a part instance.

Required arguments

elements A sequence of triangular *MeshElement* objects specifying the elements to combine.

Optional arguments

None.

Return value

None.

Exceptions

None.

deleteElement(...)

This method deletes the given elements from a part instance. The elements must have been generated using the bottom-up meshing technique.

Required arguments

elements A sequence of *MeshElement* objects or a Set object containing elements.

Optional arguments

deleteUnreferencedNodes A Boolean specifying whether to delete all those associated nodes that become unreferenced after the given elements are deleted. The default value is OFF.

Return value

`None.`

Exceptions

None.

projectNode(...)

This method projects the given nodes of a part instance onto a mesh entity, geometric entity, or a datum object.

Required arguments

nodes A sequence of *MeshNode* objects to be projected.

projectionReference An object specifying the target for the node projection operation. The *projectionReference* can be any one of the following objects: *MeshNode*, *MeshEdge*, *MeshFace*, *Vertex*, *Edge*, *Face*, *DatumPoint*, *DatumAxis*, or *DatumPlane*.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

editNode(...)

This method changes the coordinates of the given nodes on a part instance.

Required arguments

nodes
A sequence of MeshNode objects or a Set object containing nodes.

Optional arguments

coordinate1
A Float specifying the value of the first coordinate. If *coordinate1* and *offset1* are unspecified, the existing value does not change.

coordinate2
A Float specifying the value of the second coordinate. If *coordinate2* and *offset2* are unspecified, the existing value does not change.

coordinate3
A Float specifying the value of the third coordinate. If *coordinate3* and *offset3* are unspecified, the existing value does not change.

coordinates
A sequence of three-dimensional coordinate tuples specifying the coordinates for each of the given nodes. When specified, the number of coordinate tuples must match the number of given nodes, and be ordered to correspond to the given nodes in *ascending order* according to index. Furthermore, *coordinate1*, *coordinate2*, *coordinate3*, *offset1*, *offset2*, or *offset3* may not be specified.

offset1
A Float specifying an offset to apply to the value of the first coordinate of the specified nodes.

offset2
A Float specifying an offset to apply to the value of the second coordinate of the specified nodes.

offset3
A Float specifying an offset to apply to the value of the third coordinate of the specified nodes.

localCsys
A DatumCsys object specifying the local coordinate system. If unspecified, the global coordinate system will be used.

projectToGeometry
A Boolean specifying whether to project nodes back to their original geometry. For example, if a node is on a face, this method first positions the node at the new location and then projects it back to the original face. The default value is ON.

Return value

`None.`

Exceptions

A coordinate and an offset may not both be specified for the same coordinate component.

`mergeNodes(...)`

Merge the nodes of a part instance. The nodes must have been generated using the bottom-up meshing technique.

Required arguments

nodes A sequence of *MeshNode* objects specifying the nodes to merge.

Optional arguments

tolerance A Float specifying the maximum distance between nodes that will be merged to a single node. The location of the new node is the average position of the merged nodes. The default value is 10^{-6}.

removeDuplicateElements A Boolean specifying whether elements with the same connectivity after the merge will merged into a single element. The default value is True.

Return value

`None.`

Exceptions

None.

`mergeNodes(...)`

Merge two nodes of a part instance. At least one of the two nodes must have been generated using the bottom-up meshing technique.

Required arguments

node1 A *MeshNode* object specifying the first node to merge.

node2 A *MeshNode* object specifying the second node to merge.

Optional arguments

removeDuplicateElements A Boolean specifying whether elements with the same connectivity after the merge will merged into a single element. The default value is True.

Return value

`None.`

Exceptions

None.

`splitElement(...)`

This method splits quadrilateral elements into triangular elements.

Required arguments

elements A sequence of quadrilateral *MeshElement* objects specifying the elements to split. Each quadrilateral element is split into two triangular elements by the shorter diagonal.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`splitMeshEdge(...)`

This method splits an edge of a quadrilateral or triangular element of a part instance.

Required arguments

edge A single *MeshEdge* object specifying the element edge to split.

Optional arguments

parameter A Float specifying the normalized distance along the *edge* at which to split. Possible values are 0.0 <*parameter*< 1.0. The default value is 0.5.

Return value

`None.`

Exceptions

None.

`swapMeshEdge(...)`

This method swaps the diagonal of two adjacent triangular elements of a part instance.

Required arguments

edge A single *MeshEdge* object specifying the element edge to swap.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`generateMeshByOffset(...)`

This method generates a solid or shell mesh from an orphan mesh surface by generating layers of elements that propagate out normal to the surface boundary.

Required arguments

region A Region object specifying the domain to be offset.

meshType A Symbolic Constant specifying the type of mesh to be generated. Possible values are SOLID or SHELL.

totalThickness A Float specifying the total thickness of the solid layers. This argument applies only when *meshType*=SOLID.

distanceBetweenLayers A Float specifying the distance between shell layers. This argument applies only when *meshType*=SHELL.

numLayers An Int specifying the number of element layers to be generated.

Optional arguments

offsetDirection A Symbolic Constant specifying the direction of the offset. This argument is required only when the given region relates to a shell mesh. Possible values are OUTWARD, INWARD, and BOTH. The default value is OUTWARD.

initialOffset A Float specifying the magnitude of the initial offset. The default value is zero.

shareNodes Boolean specifying whether the first layer of nodes should be shared with nodes on the base surface. The default value is False.

deleteBaseElements A Boolean specifying whether to delete the shell elements after the offset layers are generated. The default value is False. This argument applies only when *meshType*=SHELL.

constantThicknessCorners A Boolean specifying whether to use element-based thickness or nodal-based thickness. The default value is False.

extendElementSets A Boolean specifying whether existing element sets that include base elements will be extended to include corresponding offset elements. The default value is False.

Return value

`None.`

Exceptions

None.

redoMeshEdit()

This method executes the edit mesh or the bottom-up meshing operation most recently undone by the `undoMeshEdit` method on an assembly. A redo action must be currently available for the assembly. This implies that the user must have executed the `undoMeshEdit` method on the assembly and that the user has not subsequently executed any further edit mesh commands on the assembly. It also implies that the user provided a sufficient cache allowance to store the undo operation.

Arguments

None.

Return value

`None.`

Exceptions

None.

undoMeshEdit()

This method undoes the most recent edit mesh or the bottom-up meshing operation on an assembly and restores the mesh on the affected part instance to its previous state. An edit mesh undo action must be available for the assembly. This implies that prior to executing an edit mesh command on the assembly, the user enabled edit mesh undo with a sufficient cache allowance to store the edit mesh operation.

Arguments

None.

Return value

`None.`

Exceptions

None.

Part object

The following commands operate on Part objects. For more information about the Part object, see *Part object*.

Access

```
import meshEdit
```

adjustMidsideNode(...)

This method is used to adjust the midside node of second-order elements of an orphan mesh part.

Required arguments

cornerNodes
A sequence of Node objects specifying the nodes towards which connected midside nodes will be biased.

parameter
A Float specifying the normalized distance along the edge of the midside nodes. Possible values are 0.0 $\leq$*parameter*$\leq$ 1.0, where 0.0 specifies the position of the corner node. The default value is 0.5.

Optional arguments

None.

Return value

None.

Exceptions

None.

cleanMesh(...)

This method is used to collapse short element edges and delete collapsed elements, or grow short element edges, on an orphan mesh part composed of linear elements.

Required arguments

mergeTolerance
A Float specifying the edge length tolerance. During the operation, edges shorter than the given tolerance will be collapsed, or grown to the specified length.

Optional arguments

growEdges
A Boolean specifying whether short element edges will be grown to the specified tolerance. Default is False, meaning short edges will be collapsed.

elements
The elements to consider as the domain for the operation. By default all elements on the part are considered. The elements may be given as a *MeshElementArray*, a list of *MeshElement* objects, a *Set*, or a list of *Set* objects.

refEdge
A *MeshEdge* specifying a reference edge to indicate a topological direction in a structured mesh that will limit which edges within the element domain are considered. That is, only edges that are found to be topologically parallel to the given reference edge will be considered by the operation. By default all edges of the element domain are considered, unless *thicknessDir* is specified, in which case the operation will attempt to determine the topological edges from the thickness direction.

thicknessDir
A tuple of two or three Floats indicating a vector along which element edge lengths will be measured.

moveLayers
A Boolean indicating whether element edges will be all grown in the direction of the thickness vector specified by *thicknessDir*. This argument is ignored unless *growEdges* is True and *thicknessDir* is provided. When this argument is True the growth of any given element edge will no longer be constrained by short edges on neighboring elements, but elements could move from their original positions in cases where there are multiple adjacent layers of thin elements. The default value is False.

Return value
`None.`

Exceptions
None.

`collapseMeshEdge(...)`

This method collapses an edge of a quadrilateral or triangular element of an orphan mesh part or part instance.

Required arguments
edge
A single *MeshEdge* object specifying the element edge to collapse.

collapseMethod
A SymbolicConstant specifying the method used to collapse the edge. Possible values are FORWARD, REVERSE, and AVERAGE.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

`combineElement(...)`

This method combines two triangular elements of an orphan mesh part or an Abaqus native mesh.

Required arguments

elements A sequence of triangular *MeshElement* objects specifying the elements to combine.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`convertSolidMeshToShell()`

This method removes all solid elements from an orphan mesh part and creates triangular or quadrilateral shell elements along their outer faces.

Arguments

None.

Return value

`None.`

Exceptions

None.

`deleteElement(...)`

This method deletes the given elements from an orphan mesh part or an Abaqus native mesh. If the elements belong to an Abaqus native mesh then the elements must have been generated using the bottom-up meshing technique.

Required arguments

elements A sequence of *MeshElement* objects or a Set object containing elements.

Optional arguments

deleteUnreferencedNodes A Boolean specifying whether to delete all those associated nodes that become unreferenced after the given elements are deleted. The default value is OFF.

Return value
`None.`

Exceptions
None.

deleteNode(...)

This method deletes the given nodes from an orphan mesh part.

Required arguments

nodes A sequence of MeshNode objects or a Set object containing nodes.

Optional arguments

deleteUnreferencedNodes A Boolean specifying whether to delete all those associated nodes that become unreferenced after the given nodes and the connected elements are deleted. The default value is OFF.

Return value
`None.`

Exceptions
None.

editNode(...)

This method changes the coordinates of the given nodes on an orphan mesh part or on an Abaqus native mesh.

Required arguments

nodes A sequence of MeshNode objects or a Set object containing nodes.

Optional arguments

coordinate1 A Float specifying the value of the first coordinate. If *coordinate1* and *offset1* are unspecified, the existing value does not change.

coordinate2 A Float specifying the value of the second coordinate. If *coordinate2* and *offset2* are unspecified, the existing value does not change.

coordinate3 A Float specifying the value of the third coordinate. If *coordinate3* and *offset3* are unspecified, the existing value does not change.

coordinates A sequence of three-dimensional coordinate tuples specifying the coordinates for each of the given nodes. When specified, the number of coordinate tuples must match the number

of given nodes, and be ordered to correspond to the given nodes in *ascending order* according to index. Furthermore, *coordinate1*, *coordinate2*, *coordinate3*, *offset1*, *offset2*, or *offset3* may not be specified.

offset1
A Float specifying an offset to apply to the value of the first coordinate of the specified nodes.

offset2
A Float specifying an offset to apply to the value of the second coordinate of the specified nodes.

offset3
A Float specifying an offset to apply to the value of the third coordinate of the specified nodes.

localCsys
A DatumCsys object specifying the local coordinate system. If unspecified, the global coordinate system will be used.

projectToGeometry
A Boolean specifying whether to project nodes back to their original geometry. For example, if a node is on a face, this method first positions the node at the new location and then projects it back to the original face. The default value is ON.

Return value
`None.`

Exceptions
A coordinate and an offset may not both be specified for the same coordinate component.

`projectNode(...)`

This method projects the given nodes onto a mesh entity, geometric entity, or a datum object. The nodes may belong to an orphan mesh part or to an Abaqus native mesh.

Required arguments

nodes
A sequence of *MeshNode* objects to be projected.

projectionReference
An object specifying the target for the node projection operation. The *projectionReference* can be any one of the following objects: *MeshNode*, *MeshEdge*, *MeshFace*, *Vertex*, *Edge*, *Face*, *DatumPoint*, *DatumAxis*, or *DatumPlane*.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

generateMesh(...)

This method generates a new mesh on an orphan mesh part based on the original mesh.

Required arguments

None.

Optional arguments

elemShape A SymbolicConstant specifying the element shape to be used for meshing. Possible values are:

TRI

Refine a planar triangular mesh and replace it with a new one. If no element sizes are attached, the new mesh will be governed by the sizes of the boundary edges in the old mesh.

TET

Create a tetrahedral mesh from a closed shell of triangular elements.

Return value

`None.`

Exceptions

None.

generateMeshByOffset(...)

This method generates a solid or shell mesh from an orphan mesh surface by generating layers of elements that propagate out normal to the surface boundary.

Required arguments

region A Region object specifying the domain to be offset.

meshType A Symbolic Constant specifying the type of mesh to be generated. Possible values are SOLID or SHELL.

totalThickness A Float specifying the total thickness of the solid layers. This argument applies only when *meshType*=SOLID.

distanceBetweenLayers A Float specifying the distance between shell layers. This argument applies only when *meshType*=SHELL.

numLayers An Int specifying the number of element layers to be generated.

Optional arguments

offsetDirection A Symbolic Constant specifying the direction of the offset. This argument is required only when the given region relates to a shell mesh. Possible values are OUTWARD, INWARD, and BOTH. The default value is OUTWARD.

initialOffset A Float specifying the magnitude of the initial offset. The default value is zero.

shareNodes Boolean specifying whether the first layer of nodes should be shared with nodes on the base surface. The default value is False.

deleteBaseElements A Boolean specifying whether to delete the shell elements after the offset layers are generated. The default value is False. This argument applies only when *meshType*=SHELL.

constantThicknessCorners A Boolean specifying whether to use element-based thickness or nodal-based thickness. The default value is False.

extendElementSets A Boolean specifying whether existing element sets that include base elements will be extended to include corresponding offset elements. The default value is False.

Return value

`None.`

Exceptions

None.

`mergeNodes(...)`

Merge the nodes of an orphan mesh part, or nodes that were generated using the bottom-up meshing technique.

Required arguments

nodes A sequence of Node objects specifying the nodes to merge.

Optional arguments

tolerance A Float specifying the maximum distance between nodes that will be merged to a single node. The location of the new node is the average position of the merged nodes. The default value is 10^{-6}.

removeDuplicateElements A Boolean specifying whether elements with the same connectivity after the merge will merged into a single element. The default value is True.

keepHighLabels A Boolean specifying which node labels will remain after nodes are merged. If True then the highest node labels are kept; when False the lowest node labels are kept. The default value is False. This argument applies only to merging of orphan mesh nodes.

Return value
`None.`

Exceptions
None.

mergeElement(...)

Merge a selection of elements arranged in layers on an orphan mesh part into a single layer.

Required arguments

edge A *MeshEdge* of one of the specified elements that serves as a reference edge to indicate the topological direction for merging elements. All specified elements must be reachable by topological navigation from this element edge, and the topological direction must be unambiguous.

elements A *MeshElementArray*, a list of *MeshElement* objects, a *Set*, or a list of *Set* objects containing the elements to be included in the merge operation.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

mergeNodes(...)

Merge two nodes of an orphan mesh part or an Abaqus native mesh. If the nodes belong to an Abaqus native mesh then at least one of the two nodes must have been generated using the bottom-up meshing technique.

Required arguments

node1 A *MeshNode* object specifying the first node to merge.

node2 A *MeshNode* object specifying the second node to merge.

Optional arguments

removeDuplicateElements A Boolean specifying whether elements with the same connectivity after the merge will merged into a single element. The default value is True.

keepHighLabels A Boolean specifying which node label will remain after nodes are merged. If True then the higher node label is kept; when False the lower node label is kept. The default value is False. This argument applies only to merging of orphan mesh nodes.

Return value
`None.`

Exceptions
None.

`orientElements(...)`

This method orients the stack direction of elements in a continuum shell or gasket mesh.

Required arguments

pickedElements	A sequence of *MeshElement* objects specifying the elements to orient.
referenceRegion	A *MeshFace* object specifying a reference top face that indicates the desired orientation.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

`removeElementSize()`

This method removes the global element size from an orphan mesh part.

Arguments
None.

Return value
`None.`

Exceptions
None.

`renumberElement(...)`

This method assigns new labels to orphan mesh elements.

Required arguments
None.

Optional arguments

Either *startLabel* and *increment*, or *offset*, or *labels* must be specified.

elements
A *MeshElementArray* or a tuple or list of *MeshElement* objects, or a *Set* containing elements to be renumbered. If unspecified, all elements in the part will be renumbered.

startLabel
A positive Int specifying the new label for the first element in *elements*.

increment
A positive Int specifying the increment used for computation of new labels for all consecutive elements in *elements*.

offset
An Int by which existing labels of the specified elements will be offset.

labels
A list of labels for the specified elements. The length of this list must match the number of specified elements.

Return value

`None.`

Exceptions

Renumbering is attempted on a native part:
Error: Renumbering can be applied to orphan mesh parts only

Renumbering data is specified incorrectly:
Error: Either startLabel and increment or offset must be specified

Renumbering will result in invalid labels:
Error: Specified data will result in invalid labels

Renumbering will result in conflicting labels:
Error: Specified data will result in conflicting labels

`renumberNode(...)`

This method assigns new labels to orphan mesh nodes.

Required arguments

None.

Optional arguments

Either *startLabel* and *increment*, or *offset*, or *labels* must be specified.

nodes
A *MeshNodeArray* or a tuple or list of *MeshNode* objects, or a *Set* containing nodes to be renumbered. If unspecified, all nodes in the part will be renumbered.

startLabel
A positive Int specifying the new label for the first node in *nodes*.

increment
A positive Int specifying the increment used for computation of new labels for all consecutive nodes in *nodes*.

offset An Int by which existing labels of the specified nodes will be offset.

labels A list of labels for the specified nodes. The length of this list must match the number of specified nodes.

Return value
`None.`

Exceptions

Renumbering is attempted on a native part: Error: Renumbering can be applied to orphan mesh parts only

Renumbering data is specified incorrectly: Error: Either startLabel and increment or offset must be specified

Renumbering will result in invalid labels: Error: Specified data will result in invalid labels

Renumbering will result in conflicting labels: Error: Specified data will result in conflicting labels

`setElementSize(...)`

This method sets the global element size for an orphan mesh part.

Required arguments

size A Float specifying the desired element size.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

`splitElement(...)`

This method splits quadrilateral elements of an orphan mesh part or a Abaqus native mesh into triangular elements.

Required arguments

elements A sequence of quadrilateral *MeshElement* objects specifying the elements to split. Each quadrilateral element is split into two triangular elements by the shorter diagonal.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

splitMeshEdge(...)

This method splits an edge of a quadrilateral or triangular element of an orphan mesh part or an Abaqus native mesh.

Required arguments

edge A single *MeshEdge* object specifying the element edge to split.

Optional arguments

parameter A Float specifying the normalized distance along *edge* at which to split. Possible values are 0.0 <*parameter*< 1.0. The default value is 0.5.

Return value

`None.`

Exceptions

None.

subdivideElement(...)

Subdivide a selection of elements on an orphan mesh part in one or more directions.

Required arguments

None.

Optional arguments

elements A *MeshElementArray*, a list of *MeshElement* objects, a *Set*, or a list of *Set* objects containing the elements to be subdivided. By default all the elements of the part are subdivided.

divisionNumber An Int specifying the number of resulting elements for each input element in each direction of the subdivision. If *face* or *edge* are not specified, elements will be subdivided according to this number in all possible directions. Must be greater than one. Default is 2.

face A *MeshFace* object that serves as a reference for indicating two topological directions for the subdivision operation. Must be a face of one of the specified elements, and all specified elements

must be reachable by topological navigation from this element face. May not be combined with *edge*.

edge

A *MeshEdge* object that serves as a reference for indicating a single topological direction for the subdivision operation. Must be an edge of one of the specified elements, and all specified elements must be reachable by topological navigation from this element edge. May not be combined with *face*.

Return value
`None.`

Exceptions
None.

`swapMeshEdge(...)`

This method swaps the diagonal of two adjacent triangular elements of an orphan mesh part or an Abaqus native mesh.

Required arguments

edge

A single *MeshEdge* object specifying the element edge to swap.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

`wrapMesh(...)`

This method wraps a planar orphan mesh part about the *Z*-axis.

Required arguments

radius

A Float specifying the radius of the cylinder about which the part is to be wrapped. The wrapping procedure will relocate a node at point (x, y) on the planar mesh to (x, θ, z), where r is the specified radius, $\theta = \frac{x}{r}$, and z=y.

Optional arguments
None.

Return value
`None.`

Exceptions

None.

redoMeshEdit()

This method executes the edit mesh or the bottom-up meshing operation most recently undone by the `undoMeshEdit` method on an part. A redo action must be currently available for the part. This implies that the user must have executed the `undoMeshEdit` method on the part and that the user has not subsequently executed any further edit mesh commands on the assembly. It also implies that the user provided a sufficient cache allowance to store the undo operation.

Arguments

None.

Return value

`None.`

Exceptions

None.

undoMeshEdit()

This method undoes the most recent edit mesh or the bottom-up meshing operation on a part and restores the mesh to its previous state. An edit mesh undo action must be available for the part. This implies that prior to executing an edit mesh command on the part, the user enabled edit mesh undo with a sufficient cache allowance to store the edit mesh operation.

Arguments

None.

Return value

`None.`

Exceptions

None.

MeshEditOptions object

The MeshEditOptions object stores settings that specify the behavior when editing meshes on parts or part instances.

The MeshEditOptions object has no constructor. Abaqus creates the *MeshEditOptions* member when a session is started.

Access

```
mdb.meshEditOptions
```

`setValues(...)`

This method modifies the MeshEditOptions object.

Required arguments

None.

Optional arguments

maxUndoCacheElements
A Float specifying the maximum allowable mesh edit undo cache size in millions of elements. If this value is set to at least the number of elements on a given part or part instance, at least one level of undo/redo capability is assured for subsequent mesh edit operations on that part or part instance. The default value is 0.0.

enableUndo
A Boolean specifying whether undo/redo of mesh edit operations will be enabled. If *enableUndo* =OFF any existing cache for undo/redo operations will be cleared for all parts and assemblies in all models. The default value is OFF.

_suspendUndo
A Boolean specifying the suspension of undo/redo for mesh edit operations. When undo/redo is suspended, undo/redo will not be available after subsequent mesh edit operations on a given part or part instances. Any pre-existing cache for mesh edit operations on other parts or assemblies in any model will be unaffected. The default value is OFF.

If you change the value of *enableUndo* to True, Abaqus sets *_suspendUndo* to False.

Return value

```
None.
```

Exceptions

None.

Members

The MeshEditOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

Engineering Feature commands

A specific type of engineering feature object is designed for each type of engineering feature.

In this section:

- *EngineeringFeature object*
- *AssembledFastener object*
- *ContourIntegral object*
- *Crack object*
- *DebondVCCT object*
- *DiscreteFastener object*
- *Fastener object*
- *HeatCapacitance object*
- *Inertia object*
- *NonstructuralMass object*
- *PointFastener object*
- *PointMassInertia object*
- *SpringDashpot object*
- *SpringDashpotToGround object*
- *TwoPointSpringDashpot object*
- *XFEMCrack object*
- *Imperfection object*
- *FileImperfection object*
- *InputImperfection object*
- *DataImperfection object*

EngineeringFeature object

The EngineeringFeature object is a container for various specific engineering feature repositories.
The EngineeringFeature object has no explicit constructor or methods.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures
import assembly
mdb.models[name].rootAssembly.engineeringFeatures
```

assignSeam(...)

This method creates a seam crack along an edge or a face.

Required arguments

regions A sequence of *Region* objects specifying the domain of the seam crack. The *Region* objects must be faces or edges.

Optional arguments

None.

Return value

None.

Exceptions

None.

deleteSeam(...)

This method deletes a seam crack.

Required arguments

regions A sequence of *Region* objects specifying the domain of the seam crack. The *Region* objects must be faces or edges.

Optional arguments

None.

Return value

None.

Exceptions

None.

Members

The EngineeringFeature object can have the following members:

inertias A repository of *Inertia* objects.

cracks A repository of *Crack* objects.

fasteners A repository of *Fastener* objects.

springDashpots A repository of *SpringDashpot* objects.

imperfections A repository of *Imperfection* objects.

AssembledFastener object

The AssembledFastener object defines an assembled fastener.

The AssembledFastener object is derived from the *Fastener* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.fasteners[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.fasteners[name]
```

AssembledFastener(...)

This method creates an AssembledFastener object. Although the constructor is available both for parts and for the assembly, AssembledFastener objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.AssembledFastener
mdb.models[name].rootAssembly.engineeringFeatures.AssembledFastener
```

Required arguments

name
A String specifying the repository key.

region
A *Region* object specifying the region of attachment points to which assembled fasteners are applied.

templateModel
A String specifying the name of the template model.

controlSet
A *Region* object specifying the template model control point set. The set must contain a single node or vertex.

templateSurfaces
A sequence of Strings specifying the names of the template model surfaces that are referenced by tie or coupling constraints.

assignedSurfaces
A sequence of Strings specifying the names of the main model surfaces that will be substituted for the template model constraint surfaces.

propertyPrefix
A String specifying the name of the property prefix string. This string will be prepended to every property name as it is copied to the main model from the template model.

Optional arguments

orientMethod
A SymbolicConstant specifying the method used to orient the virtual instances of the template model at each attachment point. Possible values are NORMALS and CSYS. The default value is NORMALS.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system. If *localCsys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

This argument applies only when *orientMethod*=CSYS.

scriptName

A String specifying the name of the property generation script. The default value is an empty string.

Return value

An AssembledFastener object.

Exceptions

None.

`setValues(...)`

This method modifies the AssembledFastener object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AssembledFastener* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The AssembledFastener object has members with the same names and descriptions as the arguments to the *AssembledFastener* method.

In addition, the AssembledFastener object has the following member:

suppressed

A Boolean specifying whether the fastener is suppressed or not. The default value is OFF.

ContourIntegral object

The ContourIntegral object defines contour integral objects on an region. Currently only assembly regions are supported.

The ContourIntegral object is derived from the *Crack* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.cracks[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.cracks[name]
```

ContourIntegral(...)

This method creates a ContourIntegral object. Although the constructor is available both for parts and for the assembly, ContourIntegral objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.ContourIntegral
mdb.models[name].rootAssembly.engineeringFeatures.ContourIntegral
```

Required arguments

name
A String specifying the repository key.

crackFront
A *RegionArray* object specifying the crack-front region to which the contour integral is applied. If the crack-front consists of a single region, a *Region* object may be specified instead of a sequence with a single item in it.

crackTip
A *RegionArray* object specifying the crack-tip region to which the contour integral is applied. If the crack-tip consists of a single region, a *Region* object may be specified instead of a sequence with a single item in it.

extensionDirectionMethod
A SymbolicConstant specifying how the virtual crack extension direction vectors are defined. Possible values are CRACK_NORMAL and Q_VECTORS.

Optional arguments

symmetric
A Boolean specifying whether the crack is defined on a half model (about a symmetry plane) or whether it is defined on the whole model. The default value is OFF.

listOfRegions
A Boolean specifying whether the regions specified by *crackFront* and *crackTip* are specified using a single region or tuples of region objects. The default value is OFF.

crackFrontName
A String specifying the name of the crack-front region generated from the tuple of regions specifying the crack-front region. This argument is valid only when *listOfRegions* is ON. The default value is *name*+"Front."

crackTipName A String specifying the name of the crack-tip region generated from the tuple of regions specifying the crack-tip region. This parameter is valid only when *listOfRegions*=ON. The default value is *name*+"Tip."

crackNormal A sequence of sequences of Floats specifying the two points of the vector that describes the crack normal direction. Each point is defined by a tuple of two or three coordinates indicating its position. This argument is required only when *extensionDirectionMethod*=CRACK_NORMAL. The default value is an empty sequence.

qVectors A sequence of sequences of sequences of Floats specifying the vectors that indicate the set of crack extension directions. Each vector is described by a tuple of two points, and each point is described by a tuple of two or three coordinates indicating its position. This argument is required only when *extensionDirectionMethod*=Q_VECTORS. The default value is an empty sequence.

midNodePosition A Float specifying the position of the midside node along the edges of the second-order elements that radiate from the crack tip. Possible values are 0.0 <*midNodeParameter*< 1.0. The default value is 0.5.

collapsedElementAtTip A SymbolicConstant specifying the crack-tip singularity. Possible values are NONE, SINGLE_NODE, and DUPLICATE_NODES. The default value is NONE.

Return value

A ContourIntegral object.

Exceptions

None.

`setValues(...)`

This method modifies the ContourIntegral object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContourIntegral* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ContourIntegral object has members with the same names and descriptions as the arguments to the *ContourIntegral* method.

In addition, the ContourIntegral object has the following member:

suppressed
A Boolean specifying whether the crack is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **CONTOUR INTEGRAL*

Crack object

The Crack object is the abstract base type for *ContourIntegral* and future crack objects.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.cracks[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.cracks[name]
```

`resume()`

This method resumes the crack that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the crack.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Crack object has the following members:

name

A String specifying the repository key.

suppressed

A Boolean specifying whether the crack is suppressed or not. The default value is OFF.

DebondVCCT object

The DebondVCCT object defines the parameters needed to activate crack propagation using VCCT.

The DebondVCCT object is derived from the *Crack* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.cracks[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.cracks[name]
```

DebondVCCT(...)

This method creates a DebondVCCT object. Although the constructor is available both for parts and for the assembly, DebondVCCT objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.DebondVCCT
mdb.models[name].rootAssembly.engineeringFeatures.DebondVCCT
```

Required arguments

name	A String specifying the repository key.
initiationStep	A String specifying the name of the step in which the DebondVCCT object is created.
surfToSurfInteraction	A String specifying the name of the *SurfaceToSurfaceContactStd* object that defines the surface to surface interaction for the crack surfaces.

Optional arguments

debondingForceAmplitude	A SymbolicConstant specifying whether the debond force between the two surfaces at the crack tip is to be released immediately or gradually during the following increment after debonding. Possible values are STEP and RAMP. The default value is STEP.
printToDATFrequency	An Int specifying the frequency at which output will be printed to DAT file. The default value is 1.

Return value

A DebondVCCT object.

Exceptions

None.

`setValues(...)`

This method modifies the DebondVCCT object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DebondVCCT* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DebondVCCT object has members with the same names and descriptions as the arguments to the *DebondVCCT* method.

In addition, the DebondVCCT object has the following member:

suppressed A Boolean specifying whether the crack is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **DEBOND*

DiscreteFastener object

The DiscreteFastener object defines a discrete fastener.

The DiscreteFastener object is derived from the *Fastener* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.fasteners[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.fasteners[name]
```

DiscreteFastener(...)

This method creates a DiscreteFastener object. Although the constructor is available both for parts and for the assembly, DiscreteFastener objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.DiscreteFastener
mdb.models[name].rootAssembly.engineeringFeatures.DiscreteFastener
```

Required arguments

name A String specifying the repository key.

region A *Region* object specifying the region to which the fastener is applied.

influenceRadius The SymbolicConstant WHOLE_SURFACE or a Float specifying the coupling influence radius.

Optional arguments

ur1 A Boolean specifying whether to constrain rotational displacement component about the 1-direction. The default value is ON.

ur2 A Boolean specifying whether to constrain rotational displacement component about the 2-direction. The default value is ON.

ur3 A Boolean specifying whether to constrain rotational displacement component about the 3-direction. The default value is ON.

coupling A SymbolicConstant specifying the coupling method used to couple the displacement and rotation of each attachment point to the average motion of the surface nodes within the radius of influence from the fastening point. Possible values are CONTINUUM and STRUCTURAL. The default value is CONTINUUM.

rotatonalCoupling A SymbolicConstant specifying the rotational coupling method used. Possible values are ROTATIONAL_CONTINUUM and ROTATIONAL_STRUCTURAL. The default value is ROTATIONAL_STRUCTURAL.

weightingMethod A SymbolicConstant specifying the weighting scheme to be used to weight the contribution of the displacements of the surface nodes within the radius of influence to the motion of the fastening point. UNIFORM, LINEAR, QUADRATIC, and CUBIC indicate uniform, linear decreasing, quadratic polynomial decreasing, and cubic polynomial monotonic decreasing weight distributions. Possible values are UNIFORM, LINEAR, QUADRATIC, and CUBIC. The default value is UNIFORM.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of fastener couplings. If *localCsys*=`None`, couplings are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

Return value

A DiscreteFastener object.

Exceptions

None.

`setValues(...)`

This method modifies the DiscreteFastener object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DiscreteFastener* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DiscreteFastener object has members with the same names and descriptions as the arguments to the *DiscreteFastener* method.

In addition, the DiscreteFastener object has the following member:

suppressed A Boolean specifying whether the fastener is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **COUPLING*, **DISTRIBUTING*

Fastener object

The Fastener object is the abstract base type for *PointFastener*, *DiscreteFastener*, and *AssembledFastener*.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.fasteners[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.fasteners[name]
```

`resume()`

This method resumes the fastener that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the fastener.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Fastener object has the following members:

name	A String specifying the repository key.
suppressed	A Boolean specifying whether the fastener is suppressed or not. The default value is OFF.

HeatCapacitance object

The HeatCapacitance object defines point heat capacitance on a part or an assembly region.

The HeatCapacitance object is derived from the *Inertia* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.inertias[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.inertias[name]
```

HeatCapacitance(...)

This method creates a HeatCapacitance object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.HeatCapacitance
mdb.models[name].rootAssembly.engineeringFeatures.HeatCapacitance
```

Required arguments

name
A String specifying the repository key.

region
A *Region* object specifying the region to which the heat capacitance is applied.

table
A sequence of sequences of Floats specifying heat capacitance properties. The items in the table data are described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Heat capacitance magnitude, ρcV (density × specific heat × volume).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A HeatCapacitance object.

Exceptions

None.

setValues(...)

This method modifies the HeatCapacitance object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HeatCapacitance* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The HeatCapacitance object has members with the same names and descriptions as the arguments to the *HeatCapacitance* method.

In addition, the HeatCapacitance object has the following member:

suppressed
A Boolean specifying whether the inertia is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **HEATCAP*

Inertia object

The Inertia object is the abstract base type for *HeatCapacitance*, *NonstructuralMass*, and *PointMassInertia*.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.inertias[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.inertias[name]
```

`resume()`

This method resumes the inertia that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the inertia.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Inertia object has the following members:

name

A String specifying the repository key.

suppressed

A Boolean specifying whether the inertia is suppressed or not. The default value is OFF.

NonstructuralMass object

The NonstructuralMass object defines the mass contribution from nonstructural features into the model.

The NonstructuralMass object is derived from the *Inertia* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.inertias[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.inertias[name]
```

NonstructuralMass(...)

This method creates a NonstructuralMass object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.NonstructuralMass
mdb.models[name].rootAssembly.engineeringFeatures.NonstructuralMass
```

Required arguments

name
A String specifying the repository key.

region
A *Region* object specifying the region to which the mass is applied.

units
A SymbolicConstant specifying the units used to specify the nonstructural mass. Possible values are TOTAL_MASS, MASS_PER_VOLUME, MASS_PER_AREA, and MASS_PER_LENGTH.

magnitude
A Float specifying the mass magnitude.

Optional arguments

distribution
A SymbolicConstant specifying the distribution of the nonstructural mass. Possible values are MASS_PROPORTIONAL and VOLUME_PROPORTIONAL. The default value is MASS_PROPORTIONAL.

The *distribution* argument applies only when *units*=TOTAL_MASS.

Return value

A NonstructuralMass object.

Exceptions

None.

setValues(...)

This method modifies the NonstructuralMass object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *NonstructuralMass* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The NonstructuralMass object has members with the same names and descriptions as the arguments to the *NonstructuralMass* method.

In addition, the NonstructuralMass object has the following member:

suppressed A Boolean specifying whether the inertia is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **NONSTRUCTURAL MASS*

PointFastener object

The PointFastener object defines a point fastener.

The PointFastener object is derived from the *Fastener* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.fasteners[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.fasteners[name]
```

PointFastener(...)

This method creates a PointFastener object. Although the constructor is available both for parts and for the assembly, PointFastener objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.PointFastener
mdb.models[name].rootAssembly.engineeringFeatures.PointFastener
```

Required arguments

name
A String specifying the repository key.

region
A *Region* object specifying the region to which fasteners are applied.

physicalRadius
A Float specifying the physical fastener radius.

Optional arguments

directionVector
A *VertexArray* object of length 2 specifying the direction of projection. Instead of through a *Vertex*, each point might be specified through a tuple of coordinates. The default value is None.

targetSurfaces
A *RegionArray* object specifying surfaces to be fastened. The default value is MODEL.

ur1
A Boolean specifying whether to constrain the rotational displacement component about the 1-direction. The default value is ON.

ur2
A Boolean specifying whether to constrain the rotational displacement component about the 2-direction. The default value is ON.

ur3
A Boolean specifying whether to constrain the rotational displacement component about the 3-direction. The default value is ON.

attachmentMethod

A SymbolicConstant specifying the method used to locate points for attaching fasteners. Possible values are FACETOFACE, EDGETOFACE, FACETOEDGE, and EDGETOEDGE. The default value is FACETOFACE.

influenceRadius

The SymbolicConstant DEFAULT or a Float specifying the maximum distance from the projection point on a connected surface within which the nodes on that surface must lie to contribute to the motion of the projection point. If the value is DEFAULT, a radius is computed from the fastener diameter and the surface facet lengths. The default value is DEFAULT.

searchRadius

The SymbolicConstant DEFAULT or a Float specifying the distance from the positioning points within which the connected points must lie. The default value is DEFAULT.

maximumLayers

The SymbolicConstant ALL or an Int specifying the maximum number of layers for each fastener. If the value is ALL, the maximum possible number of layers within the searchRadius will be used for each fastener. The default value is ALL.

coupling

A SymbolicConstant specifying the coupling method used to couple the displacement and rotation of each attachment point to the average motion of the surface nodes within the radius of influence from the fastener projection point. Possible values are CONTINUUM and STRUCTURAL. The default value is CONTINUUM.

weightingMethod

A SymbolicConstant specifying the weighting scheme to be used to weight the contribution of the displacements of the surface nodes within the radius of influence to the motion of the fastener projection point. UNIFORM, LINEAR, QUADRATIC, and CUBIC indicate uniform, linear decreasing, quadratic polynomial decreasing, and cubic polynomial monotonic decreasing weight distributions. Possible values are UNIFORM, LINEAR, QUADRATIC, and CUBIC. The default value is UNIFORM.

adjustOrientation

A Boolean specifying whether to adjust localCsys such that the local z-axis for each fastener is normal to the surface that is closest to the reference node for that fastener. The default value is ON.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system. If *localCsys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

connectionType

A SymbolicConstant specifying the fastener connection type. Possible values are CONNECTOR and BEAM_MPC. The default value is CONNECTOR.

sectionName

A String specifying the connector section assigned to generated connectors. The default value is an empty string.

connectorOrientationLocalCsys1

`None` or a *DatumCsys* object specifying the local coordinate system of the first connector point in generated connectors. If *connectorOrientationLocalCsys1*=`None`, the degrees of freedom are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

axis1
A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied for the first point in generated connectors. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle1
A Float specifying the angle of the additional rotation for the first point in generated connectors. The default value is 0.0.

orient2SameAs1
A Boolean specifying whether or not the second connector point in generated connectors is to use the same local coordinate system, axis, and angle as the first point. The default value is ON.

connectorOrientationLocalCsys2
`None` or a *DatumCsys* object specifying the local coordinate system of the second connector point in generated connectors. If *connectorOrientationLocalCsys2*=`None`, the degrees of freedom are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

axis2
A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied for the second point in generated connectors. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle2
A Float specifying the angle of the additional rotation for the second point in generated connectors. The default value is 0.0.

unsorted
A Boolean specifying whether the analysis product should leave targetSurfaces in the given unsorted order, or sort them by proximity to determine the connectivity of fastening points. The default value is OFF.

Return value

A PointFastener object.

Exceptions

None.

`setValues(...)`

This method modifies the PointFastener object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PointFastener* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The PointFastener object has members with the same names and descriptions as the arguments to the *PointFastener* method.

In addition, the PointFastener object has the following member:

suppressed
A Boolean specifying whether the fastener is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **FASTENER*

PointMassInertia object

The PointMassInertia object defines point masses and point rotary inertia on a part or an assembly region.

The PointMassInertia object is derived from the *Inertia* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.inertias[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.inertias[name]
```

PointMassInertia(...)

This method creates a PointMassInertia object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.PointMassInertia
mdb.models[name].rootAssembly.engineeringFeatures.PointMassInertia
```

Required arguments

name
A String specifying the repository key.

region
A *Region* object specifying the region to which the mass or rotary inertia is applied.

Optional arguments

mass
A Float specifying the mass magnitude for isotropic mass. This parameter cannot be specified when anisotropic mass terms are specified. The default value is 0.0.

mass1
A Float specifying the mass in the 1-direction for anisotropic mass. This parameter cannot be specified when isotropic mass is also specified. The default value is 0.0.

mass2
A Float specifying the mass in the 2-direction for anisotropic mass. This parameter cannot be specified when isotropic mass is also specified. The default value is 0.0.

mass3
A Float specifying the mass in the 3-direction for anisotropic mass. This parameter cannot be specified when isotropic mass is also specified. The default value is 0.0.

i11
A Float specifying the rotary inertia about the local 1-axis, I_{11}. The default value is 0.0.

i22
A Float specifying the rotary inertia about the local 2-axis, I_{22}. The default value is 0.0.

i33
A Float specifying the rotary inertia about the local 3-axis, I_{33}. The default value is 0.0.

i12
A Float specifying the product of inertia, I_{12}. The default value is 0.0.

i13

A Float specifying the product of inertia, I_{13}. The default value is 0.0.

i23

A Float specifying the product of inertia, I_{23}. The default value is 0.0.

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system for the anisotropic mass terms (when specified), and the rotary inertia (when specified). If *localCsys*=`None`, the anisotropic mass and rotary inertia data are defined in the global coordinate system. The default value is `None`.

alpha

A Float specifying the alpha damping magnitude. The default value is 0.0.

This argument applies only to Abaqus/Standard analyses.

composite

A Float specifying the composite damping magnitude. The default value is 0.0.

This argument applies only to Abaqus/Standard analyses.

Return value

A PointMassInertia object.

Exceptions

None.

`setValues(...)`

This method modifies the PointMassInertia object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PointMassInertia* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The PointMassInertia object has members with the same names and descriptions as the arguments to the *PointMassInertia* method.

In addition, the PointMassInertia object has the following member:

suppressed

A Boolean specifying whether the inertia is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **MASS*, **ROTARY INERTIA*

SpringDashpot object

The SpringDashpot object is the abstract base type for the *SpringDashpotToGround* and *TwoPointSpringDashpot* objects.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.springDashpots[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.springDashpots[name]
```

`resume()`

This method resumes the spring/dashpot that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the spring/dashpot.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SpringDashpot object has the following members:

name A String specifying the repository key.

suppressed A Boolean specifying whether the spring/dashpot is suppressed or not. The default value is OFF.

SpringDashpotToGround object

The SpringDashpotToGround object defines springs and/or dashpots between points and ground on a part or an assembly region.

The SpringDashpotToGround object is derived from the *SpringDashpot* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.springDashpots[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.springDashpots[name]
```

SpringDashpotToGround(...)

This method creates a SpringDashpotToGround object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.SpringDashpotToGround
mdb.models[name].rootAssembly.engineeringFeatures\
.SpringDashpotToGround
```

Required arguments

name A String specifying the repository key.

region A *Region* object specifying the region to which the springs and/or dashpots are applied.

dof An Int specifying the degree of freedom associated with the spring and dashpot behaviors.

Optional arguments

orientation `None` or a *DatumCsys* object specifying the local directions for the spring and/or dashpot. If *orientation*=`None`, the spring and/or dashpot data are defined in the global coordinate system. The default value is `None`.

springBehavior A Boolean specifying whether to apply spring behavior to the selected points. The default value is OFF.

At least one of the arguments *springBehavior*=ON or *dashpotBehavior*=ON must be specified.

dashpotBehavior A Boolean specifying whether to apply dashpot behavior to the selected points. The default value is OFF.

At least one of the arguments *springBehavior*=ON or *dashpotBehavior*=ON must be specified.

springStiffness A Float specifying the force per relative displacement for the spring. The default value is 0.0.

dashpotCoefficient A Float specifying the force per relative velocity for the dashpot. The default value is 0.0.

Return value

A SpringDashpotToGround object.

Exceptions

None.

`setValues(...)`

This method modifies the SpringDashpotToGround object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SpringDashpotToGround* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SpringDashpotToGround object has members with the same names and descriptions as the arguments to the *SpringDashpotToGround* method.

In addition, the SpringDashpotToGround object has the following member:

suppressed A Boolean specifying whether the spring/dashpot is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **ELEMENT*, TYPE=SPRING1; **ELEMENT*, TYPE=DASHPOT1; **SPRING*; **DASHPOT*

TwoPointSpringDashpot object

The TwoPointSpringDashpot object defines springs and/or dashpots between two points on a part or an assembly.

The TwoPointSpringDashpot object is derived from the *SpringDashpot* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.springDashpots[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.springDashpots[name]
```

TwoPointSpringDashpot(...)

This method creates a TwoPointSpringDashpot object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.TwoPointSpringDashpot
mdb.models[name].rootAssembly.engineeringFeatures\
.TwoPointSpringDashpot
```

Required arguments

name
A String specifying the repository key.

regionPairs
A sequence of pairs of *Region* objects specifying the points between which the springs and/or dashpots are applied.

axis
A SymbolicConstant specifying whether the axis of the springs and/or dashpots follows the rotation of the nodes or is in a specified direction. Possible values are NODAL_LINE and FIXED_DOF.

Optional arguments

dof1
An Int specifying the degree of freedom with which the springs and/or dashpots are associated at their first points. The *dof1* argument applies only when *axis*=FIXED_DOFS. The default value is 0.

dof2
An Int specifying the degree of freedom with which the springs and/or dashpots are associated at their second points. The *dof2* argument applies only when *axis*=FIXED_DOFS. The default value is 0.

orientation
`None` or a *DatumCsys* object specifying the local directions for the spring and/or dashpot. If *orientation*=`None`, the spring and/or dashpot data are defined in the global coordinate system. The default value is `None`.

The *orientation* argument applies only when *axis*=FIXED_DOFS.

springBehavior A Boolean specifying whether to apply spring behavior to the selected point pairs. The default value is OFF.

At least one of the arguments *springBehavior*=ON or *dashpotBehavior*=ON must be specified.

dashpotBehavior A Boolean specifying whether to apply dashpot behavior to the selected point pairs. The default value is OFF.

At least one of the arguments *springBehavior*=ON or *dashpotBehavior*=ON must be specified.

springStiffness A Float specifying the force per relative displacement for the springs. The default value is 0.0.

dashpotCoefficient A Float specifying the force per relative velocity for the dashpots. The default value is 0.0.

Return value

A TwoPointSpringDashpot object.

Exceptions

None.

`setValues(...)`

This method modifies the TwoPointSpringDashpot object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TwoPointSpringDashpot* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TwoPointSpringDashpot object has members with the same names and descriptions as the arguments to the *TwoPointSpringDashpot* method.

In addition, the TwoPointSpringDashpot object has the following member:

suppressed A Boolean specifying whether the spring/dashpot is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **ELEMENT*, TYPE=SPRINGA; **ELEMENT*, TYPE=SPRING2; **ELEMENT*, TYPE=DASHPOTA; **ELEMENT*, TYPE=DASHPOT2; **SPRING*; **DASHPOT*

XFEMCrack object

The XFEMCrack object defines the parameters needed to model crack initiation or crack growth using XFEM technology. Currently only assembly regions are supported.

The XFEMCrack object is derived from the *Crack* object.

Access

```
import part
mdb.models[name].parts[name].engineeringFeatures.cracks[name]
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.cracks[name]
```

XFEMCrack(...)

This method creates a XFEMCrack object. Although the constructor is available both for parts and for the assembly, XFEMCrack objects are currently supported only under the assembly.

Path

```
mdb.models[name].parts[name].engineeringFeatures.XFEMCrack
mdb.models[name].rootAssembly.engineeringFeatures.XFEMCrack
```

Required arguments

name
A String specifying the repository key.

crackDomain
A *Region* object specifying the region that contains the crack or is likely to contain the crack.

Optional arguments

allowCrackGrowth
A Boolean specifying whether the crack is allowed to propagate (grow). The default value is ON.

crackLocation
A *Region* object specifying the initial crack location. This parameter is required when *allowCrackGrowth*=OFF.

singularityCalcRadius
`None` or a Float specifying the radius from the crack tips within which the elements are used for crack singularity calculations. This argument applies only when *allowCrackGrowth*=OFF. The default value is `None`.

interactionProperty
A String specifying the name of the *ContactProperty* object that defines the contact properties for the crack surfaces. The default value is an empty string.

elemId
A sequence of Ints specifying the labels of the elements that are intersected by the initial crack location. This argument is used only by the input file reader.

nodeId
A sequence of Ints specifying the position of a node in the corresponding element connectivity. This argument is used only by the input file reader.

hasCrackFront
A sequence of Ints specifying the values indicating the inclusion/exclusion of the *crackFrontDist* values. A zero value indicates that *crackFrontDist* is not specified for the ith pair *elemId* and *nodeId*. This argument is used only by the input file reader.

crackPlaneDist
A sequence of Floats specifying the values of the first signed distance function. This argument is used by the input file reader.

crackFrontDist
A sequence of Floats specifying the values of the second signed distance function. This argument is used only by the input file reader.

autoDetectValue
An integer specifying the number of element layers around the crack location, to which the crack domain is shrunk.

Return value

A XFEMCrack object.

Exceptions

None.

`setValues(...)`

This method modifies the XFEMCrack object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *XFEMCrack* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The XFEMCrack object has members with the same names and descriptions as the arguments to the *XFEMCrack* method.

In addition, the XFEMCrack object has the following member:

suppressed
A Boolean specifying whether the crack is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **ENRICHMENT*
- **INITIAL CONDITIONS*, TYPE=ENRICHMENT

Imperfection object

The Imperfection object is the abstract base type for *FileImperfection*, *InputImperfection*, and *DataImperfection*.

Access

```
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.imperfections[name]
```

`resume()`

This method resumes the imperfection that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the imperfection.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Imperfection object has the following members:

name
A String specifying the repository key.

suppressed
A Boolean specifying whether the imperfection is suppressed or not. The default value is OFF.

FileImperfection object

The FileImperfection object defines geometric imperfection on an assembly region.

The FileImperfection object is derived from the *Imperfection* object.

Access

```
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.imperfections[name]
```

FileImperfection(...)

This method creates a FileImperfection object.

Path

```
mdb.models[name].rootAssembly.engineeringFeatures.FileImperfection
```

Required arguments

name	A String specifying the repository key.
file	A String specifying the results file from a previous analysis from which the file imperfection is applied.
step	An Int specifying the step number (in the analysis whose file is being used as input to this option) from which the modal or displacement data are to be read.
linearSuperpositions	A sequence of sequences of Integers and Floats specifying linearSuperpositions. The items in the table data are described below.

Optional arguments

region	A *Region* object specifying the region to which the file imperfection is applied. By default, the imperfection will be applied to all nodes in the model.
increment	An Int specifying the increment number (in the analysis whose file is being used as input to this option) from which the displacement data are to be read. By default, the data will be read from the last increment available for the specified step.

Table data

The table data specify the following:

- Mode Number.
- Scaling factor for this mode.

Return value

A FileImperfection object.

Exceptions

None.

setValues(...)

This method modifies the FileImperfection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FileImperfection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FileImperfection object has members with the same names and descriptions as the arguments to the *FileImperfection* method.

In addition, the FileImperfection object has the following member:

suppressed
A Boolean specifying whether the imperfection is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **IMPERFECTION*

InputImperfection object

The InputImperfection object defines geometric imperfection on an assembly region.

The InputImperfection object is derived from the *Imperfection* object.

Access

```
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.imperfections[name]
```

InputImperfection(...)

This method creates an InputImperfection object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.InputImperfection
```

Required arguments

name A String specifying the repository key.

file A String specifying the name of the alternate input file containing the imperfection data.

Optional arguments

system A SymbolicConstant specifying the coordinate system. The imperfection values in the alternate input file being used as input to this option would be treated as perturbation values of respective coordinates. Possible values areR, C and S. The default value is R.

Return value

A InputImperfection object.

Exceptions

None.

setValues(...)

This method modifies the InputImperfection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *InputImperfection* method, except for the *name* argument.

Return value

```
None.
```

Exceptions

None.

Members

The InputImperfection object has members with the same names and descriptions as the arguments to the *InputImperfection* method.

In addition, the InputImperfection object has the following member:

suppressed
A Boolean specifying whether the imperfection is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **IMPERFECTION*

DataImperfection object

The DataImperfection object defines geometric imperfection on an assembly region.

The DataImperfection object is derived from the *Imperfection* object.

Access

```
import assembly
mdb.models[name].rootAssembly.engineeringFeatures.imperfections[name]
```

DataImperfection(...)

This method creates a DataImperfection object.

Path

```
mdb.models[name].parts[name].engineeringFeatures.DataImperfection
```

Required arguments

name
A String specifying the repository key.

imperfectionTable
A sequence of sequences of Ints and Floats specifying the imperfection components at a given node. The items in the table data are described below.

Optional arguments

system
A SymbolicConstant specifying the coordinate system. The imperfection values in the data lines would be treated as perturbation values of the respective coordinates. Possible values are R, C, and S. The default value is R.

Table data

The imperfectionTable data specify the following:

- Node number.
- Component of imperfection in the first coordinate direction.
- Component of imperfection in the second coordinate direction.
- Component of imperfection in the third coordinate direction.

Return value

A DataImperfection object.

Exceptions

None.

`setValues(...)`

This method modifies the DataImperfection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DataImperfection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DataImperfection object has members with the same names and descriptions as the arguments to the *DataImperfection* method.

In addition, the DataImperfection object has the following member:

suppressed
A Boolean specifying whether the imperfection is suppressed or not. The default value is OFF.

Corresponding analysis keywords

- **IMPERFCTION*

Feature commands

Features in Abaqus/CAE include Parts, Datums, Partitions, and Assembly operations.
The commands that create Feature objects on only the Part object are described in *Part commands*. The commands that create Feature objects in only the rootAssembly object are described in *Assembly commands*. The commands that create Feature objects on both the Part and the rootAssembly objects are described here.

In this section:

- *Feature object*
- *FeatureOptions object*

Feature object

Abaqus/CAE is a feature-based modeling system, and features are stored in the Feature object. The user defines the parameters of the feature, and Abaqus/CAE modifies the model based on the value of the parameters. This evaluation of the parameters is called regeneration of the feature. Feature objects contain both the parameters and the resulting model modification.

Access

```
import part
mdb.models[name].parts[name].features[name]
mdb.models[name].parts[name].featuresById[i]
import assembly
mdb.models[name].rootAssembly.features[name]
mdb.models[name].rootAssembly.featuresById[i]
```

AttachmentPoints(...)

This method creates an attachment points Feature. Attachment points may be created using datum points, vertices, reference points, attachment points, interesting points, orphan mesh nodes or coordinates. Optionally, the attachment points can be projected on geometric faces or element faces.

Path

mdb.models[*name*].parts[*name*].AttachmentPoints

mdb.models[*name*].rootAssembly.AttachmentPoints

Required arguments

name A String specifying a unique Feature name.

points A tuple of points. Each point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

Optional arguments

projectionMethod A SymbolicConstant specifying the projection method. Possible values are PROJECT_BY_PROXIMITY and PROJECT_BY_DIRECTION. The default value is PROJECT_BY_PROXIMITY.

projectOnFaces A sequence of *Face* objects specifying the geometry faces onto which the points are to be projected.

projectOnElementFaces A sequence of *MeshFace* objects specifying the orphan mesh element faces onto which the points are to be projected.

projectionDirStartPt A point specifying the start point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

projectionDirEndPt A point specifying the end point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

setName A String specifying a unique set name.

Return value

A Feature object.

Exceptions

None.

AttachmentPointsAlongDirection(...)

This method creates a Feature object by creating attachment points along a direction or between two points. A Datum point, a Vertex, a Reference point, an Attachment point, an Interesting point, or an orphan mesh Node can be specified as the start or end point. The direction can be specified using a straight edge or a datum axis.

Path

mdb.models[*name*].parts[*name*].AttachmentPointsAlongDirection

mdb.models[*name*].rootAssembly.AttachmentPointsAlongDirection

Required arguments

name A String specifying a unique Feature name.

startPoint A point specifying the start point of the direction along which to create points. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

pointCreationMethod A SymbolicConstant specifying the point creation method. Possible values are AUTO_FIT, NUM_PTS_ALONG_DIR, and NUM_PTS_BETWEEN_PTS.

Optional arguments

endPoint A point specifying the end point if creating points between two points. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

direction The direction can be specified by a straight edge or a datum axis.

spacing A float specifying the spacing to be used between two points.

numPtsAlongDir An integer specifying the number of points to be created along the specified direction.

numPtsBetweenPts
An integer specifying the number of points to be created between the start and end points.

createPtAtStartPt
A Boolean specifying whether to create an attachment point at the start point. The default value is True.

createPtAtEndPt
A Boolean specifying whether to create an attachment point at the end point. The default value is True.

projectionMethod
A SymbolicConstant specifying the projection method. Possible values are PROJECT_BY_PROXIMITY and PROJECT_BY_DIRECTION. The default value is PROJECT_BY_PROXIMITY.

projectOnFaces
A sequence of *Face* objects specifying the geometry faces onto which the points are to be projected.

projectOnElementFaces
A sequence of *MeshFace* objects specifying the orphan mesh element faces onto which the points are to be projected.

projectionDirStartPt
A point specifying the start point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

projectionDirEndPt
A point specifying the end point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

flipDirection
A Boolean specifying if the direction along which the attachment points are created should be reversed. This argument is valid only when *pointCreationMethod*=NUM_PTS_ALONG_DIR.

setName
A String specifying a unique set name.

Return value

A Feature object.

Exceptions

None.

AttachmentPointsOffsetFromEdges(...)

This method creates a Feature object by creating attachment points along or offset from one or more connected edges.

Path

mdb.models[*name*].parts[*name*].AttachmentPointsOffsetFromEdges

mdb.models[*name*].rootAssembly.AttachmentPointsOffsetFromEdges

Required arguments

name
A String specifying a unique Feature name.

edges
A sequence of connected *Edge* objects specifying the geometry edges from which to offset the points.

Optional arguments

startPoint
A Vertex of the selected edges that specifies the point from which to create points. This point can be one of the two end vertices of the connected edges. In case of edges forming a closed loop and having multiple vertices, this point can be any one of the vertices on the edges.

flipDirection
This parameter is required to indicate the direction in which to create the points. This parameter is required only in case of edges forming a closed loop.

pointCreationMethod
A SymbolicConstant specifying the point creation method. Possible values are BY_NUMBER or BY_SPACING.

numberOfPoints
An integer specifying the number of points to be created along the selected edges.

spacingBetweenPoints
A float specifying the spacing to be used between two points while creating the points between the start and end points of the edges.

offsetFromStartPoint
A float specifying the distance by which to offset the first point from the start vertex of the edge chain. The default value is 0.0.

offsetFromEndPoint
A float specifying the distance by which to offset the last point from the end vertex of the edge chain. This parameter should be specified only if the point creation method is BY_NUMBER. The default value is 0.0.

spacingMethod
A SymbolicConstant specifying the spacing method. Possible values are AUTO_FIT_PTS or SPECIFY_NUM_PTS. The default value is AUTO_FIT_PTS.

patterningMethod
A SymbolicConstant specifying the method to pattern of points. Possible values are PATTERN_ORTHOGONALLY or PATTERN_ALONG_DIRECTION.

referenceFace
A geometry *Face* object adjacent to one of the edges from which to offset the points to create a pattern of points when the PATTERN_ORTHOGONALLY method is chosen for patterning. The face is used to identify the patterning direction. If the number of rows is one and the initial offset is zero, the reference face may not be specified.

startPointForPatternDirection
A point specifying the start point of the direction along which to create a pattern of points when the PATTERN_ALONG_DIRECTION method is chosen for patterning. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

endPointForPatternDirection A point specifying the end point of the direction along which to create a pattern of points when the PATTERN_ALONG_DIRECTION method is chosen for patterning. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

offsetFromEdges A float specifying the distance by which to offset the first row of points from the edges.

numberOfRows An integer specifying the number of rows of points to be created for the pattern. The default value is 1.

spacingBetweenRows A float specifying the spacing to be used between two rows while creating a pattern of points.

projectionMethod A SymbolicConstant specifying the projection method. Possible values are PROJECT_BY_PROXIMITY and PROJECT_BY_DIRECTION. The default value is PROJECT_BY_PROXIMITY.

projectOnFaces A sequence of *Face* objects specifying the geometry faces onto which the points are to be projected.

projectOnElementFaces A sequence of *MeshFace* objects specifying the orphan mesh element faces onto which the points are to be projected.

projectionDirStartPt A point specifying the start point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

projectionDirEndPt A point specifying the end point of the projection direction. The point can be a Vertex, Datum point, Reference point, Attachment point, orphan mesh Node, Interesting point object, or a tuple of Floats representing the coordinates of a point.

setName A String specifying a unique set name.

Return value

A Feature object.

Exceptions

None.

DatumAxisByCylFace(...)

This method creates a Feature object and a *DatumAxis* object along the axis of a cylinder or cone.

Path

mdb.models[*name*].rootAssembly.DatumAxisByCylFace

mdb.models[*name*].parts[*name*].DatumAxisByCylFace

Required arguments

face A cylindrical or conical Face object.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByNormalToPlane(...)

This method creates a Feature object and a *DatumAxis* object normal to the specified plane and passing through the specified point.

Path

mdb.models[*name*].rootAssembly.DatumAxisByNormalToPlane

mdb.models[*name*].parts[*name*].DatumAxisByNormalToPlane

Required arguments

plane A planar Face, an ElementFace, or a *Datum* object representing a datum plane.

point A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByParToEdge(...)

This method creates a Feature object and a *DatumAxis* object parallel to the specified edge and passing through the specified point.

Path

mdb.models[*name*].rootAssembly.DatumAxisByParToEdge

mdb.models[*name*].parts[*name*].DatumAxisByParToEdge

Required arguments

edge A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis.

point A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByPrincipalAxis(...)

This method creates a Feature object and a *DatumAxis* object along one of the three principal axes.

Path

mdb.models[*name*].rootAssembly.DatumAxisByPrincipalAxis

mdb.models[*name*].parts[*name*].DatumAxisByPrincipalAxis

Required arguments

principalAxis A SymbolicConstant specifying the principal axis. Possible values are XAXIS, YAXIS, and ZAXIS.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByRotation(...)

This method creates a Feature object and a *DatumAxis* object in a three-dimensional model by rotating a line about the specified axis through the specified angle.

Path

mdb.models[*name*].rootAssembly.DatumAxisByRotation

mdb.models[*name*].parts[*name*].DatumAxisByRotation

Required arguments

line A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object specifying the line to rotate.

axis A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object specifying the axis about which to rotate the line.

angle A Float specifying the angle in degrees to rotate the line.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByRotation(...)

This method creates a Feature object and a *DatumAxis* object in a two-dimensional model by rotating a line about the specified point through the specified angle.

Path

mdb.models[*name*].rootAssembly.DatumAxisByRotation

mdb.models[*name*].parts[*name*].DatumAxisByRotation

Required arguments

line A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object specifying the line to rotate.

point A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying the point about which to rotate the line.

angle A Float specifying the angle in degrees to rotate the line.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByThreePoint(...)

This method creates a Feature object and a *DatumAxis* object normal to the circle described by three points and through its center.

Path

mdb.models[*name*].rootAssembly.DatumAxisByThreePoint

mdb.models[*name*].parts[*name*].DatumAxisByThreePoint

Required arguments

point1 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying the first point on the circle.

point2 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying the second point on the circle.

point3 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying the third point on the circle.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByThruEdge(...)

This method creates a Feature object and a *DatumAxis* object along the specified edge.

Path

mdb.models[*name*].rootAssembly.DatumAxisByThruEdge

mdb.models[*name*].parts[*name*].DatumAxisByThruEdge

Required arguments

edge A straight Edge or an ElementEdge object.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByTwoPlane(...)

This method creates a Feature object and a *DatumAxis* object at the intersection of two planes.

Path

mdb.models[*name*].rootAssembly.DatumAxisByTwoPlane

mdb.models[*name*].parts[*name*].DatumAxisByTwoPlane

Required arguments

plane1 A planar Face, an ElementFace, or a *Datum* object representing a datum plane.

plane2 A planar Face, an ElementFace, or a *Datum* object representing a datum plane.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumAxisByTwoPoint(...)

This method creates a Feature object and a *DatumAxis* object along the line joining two points.

Path

mdb.models[*name*].rootAssembly.DatumAxisByTwoPoint

mdb.models[*name*].parts[*name*].DatumAxisByTwoPoint

Required arguments

point1 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

point2 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumCsysByDefault(...)

This method creates a Feature object and a *DatumCsys* object from the specified default coordinate system at the origin.

Path

mdb.models[*name*].rootAssembly.DatumCsysByDefault

mdb.models[*name*].parts[*name*].DatumCsysByDefault

Required arguments

coordSysType A SymbolicConstant specifying the default coordinate system to be used. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

Optional arguments

name A String specifying the name of the *DatumCsys*.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumCsysByOffset(...)

This method creates a Feature object and a *DatumCsys* object by offsetting the origin of an existing datum coordinate system to a specified point.

Path

mdb.models[name].rootAssembly.DatumCsysByOffset

mdb.models[name].parts[name].DatumCsysByOffset

Required arguments

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

datumCoordSys A *Datum* object representing a datum coordinate system from which to offset.

vector A sequence of three Floats specifying the *X*-, *Y*-, and *Z*-offsets from *datumCoordSys*. The arguments *vector* and *point* are mutually exclusive, and one of them must be specified.

point A *Vertex*, *InterestingPoint*, *DatumPoint* object or a sequence of three Floats specifying the *X*-, *Y*-, and *Z*-coordinates of a point in space. The point represents the origin of the new datum coordinate system. The arguments *vector* and *point* are mutually exclusive, and one of them must be specified.

Optional arguments

name A String specifying the name of the *DatumCsys*.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumCsysByThreePoints(...)

This method creates a Feature object and a *DatumCsys* object from three points.

Path

```
mdb.models[name].parts[name].DatumCsysByThreePoints
mdb.models[name].rootAssembly.DatumCsysByThreePoints
```

Required arguments

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

origin A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying the origin of the coordinate system.

point1 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying a point on the *X*-axis or the r-axis. The *point1* and *line1* arguments are mutually exclusive. One of them must be specified.

point2 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point specifying a point in the *X*–*Y* plane or the r–θ plane. The *point2* and *line2* arguments are mutually exclusive. One of them must be specified.

line1 An Edge, an Element Edge, or a *Datum* object representing a datum axis specifying the *X*-axis or the r-axis. The *point1* and *line1* arguments are mutually exclusive. One of them must be specified.

line2 An Edge, an Element Edge, or a *Datum* object representing a datum axis specifying a vector in the *X*–*Y* plane or the r–θ plane. The *point2* and *line2* arguments are mutually exclusive. One of them must be specified.

Optional arguments

name A String specifying the name of the *DatumCsys*.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumCsysByTwoLines(...)

This method creates a Feature object and a *DatumCsys* object from two orthogonal lines. The origin of the new datum coordinate system is placed at the intersection of the two lines.

Path

mdb.models[*name*].rootAssembly.DatumCsysByTwoLines

mdb.models[*name*].parts[*name*].DatumCsysByTwoLines

Required arguments

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

line1 A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis specifying the *X*-axis or the r-axis.

line2
A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis specifying a line in the *X*–*Y* plane or in the r–θ plane.

Optional arguments

name
A String specifying the name of the *DatumCsys*.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByPrincipalPlane(...)

This method creates a Feature object and a *DatumPlane* object through the origin along one of the three principal planes.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByPrincipalPlane

mdb.models[*name*].parts[*name*].DatumPlaneByPrincipalPlane

Required arguments

principalPlane
A SymbolicConstant specifying the principal plane. Possible values are XYPLANE, YZPLANE, and XZPLANE.

offset
A Float specifying the offset from the plane.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByOffset(...)

This method creates a Feature object and a *DatumPlane* object offset by a specified distance from an existing plane.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByOffset

mdb.models[*name*].parts[*name*].DatumPlaneByOffset

Required arguments

plane	A planar Face, an ElementFace, or a *Datum* object representing a datum plane.
flip	A SymbolicConstant specifying whether the normal should be flipped. Possible values are SIDE1 and SIDE2.
offset	A Float specifying the offset from the plane.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByOffset(...)

This method creates a Feature object and a *DatumPlane* object offset from an existing plane and passing through the specified point.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByOffset

mdb.models[*name*].parts[*name*].DatumPlaneByOffset

Required arguments

plane	A planar Face, an ElementFace, or a *Datum* object representing a datum plane.
point	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByRotation(...)

This method creates a Feature object and a *DatumPlane* object by rotating a plane about the specified axis through the specified angle.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByRotation

mdb.models[*name*].parts[*name*].DatumPlaneByRotation

Required arguments

plane	A planar Face, an ElementFace, or a *Datum* object representing a datum plane.
axis	A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis.
angle	A Float specifying the angle in degrees to rotate the plane.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByThreePoints(...)

This method creates a Feature object and a *DatumPlane* object defined by passing through three points.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByThreePoints

mdb.models[*name*].parts[*name*].DatumPlaneByThreePoints

Required arguments

point1	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.
point2	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.
point3	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByLinePoint(...)

This method creates a Feature object and a *DatumPlane* object that pass through the specified line and through the specified point that does not lie on the line.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByLinePoint

mdb.models[*name*].parts[*name*].DatumPlaneByLinePoint

Required arguments

line	A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis.
point	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByPointNormal(...)

This method creates a Feature object and a *DatumPlane* object normal to the specified line and running through the specified point.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByPointNormal

mdb.models[*name*].parts[*name*].DatumPlaneByPointNormal

Required arguments

point	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.
normal	A straight Edge, an ElementEdge, or a *Datum* object representing a datum axis.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPlaneByTwoPoint(...)

This method creates a Feature object and a *DatumPlane* object midway between two points and normal to the line connecting the points.

Path

mdb.models[*name*].rootAssembly.DatumPlaneByTwoPoint

mdb.models[*name*].parts[*name*].DatumPlaneByTwoPoint

Required arguments

point1 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

point2 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPointByCoordinate(...)

This method creates a Feature object and a *DatumPoint* object at the point defined by the specified coordinates.

Path

mdb.models[*name*].rootAssembly.DatumPointByCoordinate

mdb.models[*name*].parts[*name*].DatumPointByCoordinate

Required arguments

coords A sequence of three Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

DatumPointByOffset(...)

This method creates a Feature object and a *DatumPoint* object offset from an existing point by a vector.

Path

mdb.models[*name*].rootAssembly.DatumPointByOffset

mdb.models[*name*].parts[*name*].DatumPointByOffset

Required arguments

point A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

vector A sequence of three Floats specifying the *X*-, *Y*-, and *Z*-offsets from *point*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

DatumPointByMidPoint(...)

This method creates a Feature object and a *DatumPoint* object midway between two points.

Path

mdb.models[*name*].rootAssembly.DatumPointByMidPoint

mdb.models[*name*].parts[*name*].DatumPointByMidPoint

Required arguments

point1 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

point2 A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

DatumPointByOnFace(...)

This method creates a Feature object and a *DatumPoint* object on the specified face, offset from two edges.

Path

mdb.models[*name*].rootAssembly.DatumPointByOnFace

mdb.models[*name*].parts[*name*].DatumPointByOnFace

Required arguments

face A planar Face or a *Datum* object representing a datum plane.

edge1 A straight Edge or a *Datum* object representing a datum axis.

offset1 A Float specifying the offset from *edge1*.

edge2 A straight Edge or a *Datum* object representing a datum axis.

offset2 A Float specifying the offset from *edge2*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPointByEdgeParam(...)

This method creates a Feature object and a *DatumPoint* object along an edge at a selected distance from one end of the edge.

Path

mdb.models[*name*].rootAssembly.DatumPointByEdgeParam

mdb.models[*name*].parts[*name*].DatumPointByEdgeParam

Required arguments

edge	An Edge object.
parameter	A Float specifying the distance along *edge* to the *DatumPoint* object. Possible values are 0 <*parameter*< 1.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException and RangeError.

DatumPointByProjOnEdge(...)

This method creates a Feature object and a *DatumPoint* object along an edge by projecting an existing point along the normal to the edge.

Path

mdb.models[*name*].rootAssembly.DatumPointByProjOnEdge

mdb.models[*name*].parts[*name*].DatumPointByProjOnEdge

Required arguments

point	A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.
edge	An Edge, an ElementEdge or a *Datum* object representing a datum axis.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

DatumPointByProjOnFace(...)

This method creates a Feature object and a *DatumPoint* object on a specified face by projecting an existing point onto the face.

Path

mdb.models[*name*].rootAssembly.DatumPointByProjOnFace

mdb.models[*name*].parts[*name*].DatumPointByProjOnFace

Required arguments

point A Vertex, an InterestingPoint, a MeshNode, or a *Datum* object representing a datum point.

face A Face object or a *Datum* object representing a datum plane.



Note:

Any other types of planes are not supported.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

MakeSketchTransform(...)

This method creates a *Transform* object. A Transform object is a 4x3 matrix of Floats that represents the transformation from sketch coordinates to part coordinates.

Path

mdb.models[*name*].parts[*name*].MakeSketchTransform

mdb.models[*name*].rootAssembly.MakeSketchTransform

Required arguments

sketchPlane A *Datum* plane object or a planar *Face* object specifying the sketch plane.

Optional arguments

origin A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates that will be used as the origin of the sketch. The default value is computed as the centroid of the face.

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

sketchPlaneSide A SymbolicConstant specifying on which side of the *sketchPlane* the sketch is positioned. Possible values are SIDE1 and SIDE2. The default value is SIDE1.

sketchUpEdge An *Edge* or *DatumAxis* object specifying the orientation of the sketch. If unspecified, the sketch is assumed to be oriented with the *Y*-direction pointing up.

Return value

A Transform object. A Transform is an object with one method that returns the transform matrix.

Exceptions

If the sketchUpEdge is parallel to the sketchPlane: Up direction is parallel to plane normal

PartitionCellByDatumPlane(...)

This method partitions one or more cells using the given datum plane.

Path

mdb.models[*name*].parts[*name*].PartitionCellByDatumPlane

mdb.models[*name*].rootAssembly.PartitionCellByDatumPlane

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

datumPlane A *DatumPlane* object.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByExtendFace(...)

This method partitions one or more cells by extending the underlying geometry of a given face to partition the target cells.

Path

mdb.models[*name*].parts[*name*].PartitionCellByExtendFace

mdb.models[*name*].rootAssembly.PartitionCellByExtendFace

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

extendFace A planar, cylindrical, conical, or spherical Face object.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByExtrudeEdge(...)

This method partitions one or more cells by extruding selected edges in the given direction.

Path

mdb.models[*name*].parts[*name*].PartitionCellByExtrudeEdge

mdb.models[*name*].rootAssembly.PartitionCellByExtrudeEdge

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

edges The *Edge* objects to be extruded. The edges must be in the same plane. The edges must form a continuous chain, without branches. The edges must belong to the same *PartInstance* object.

line A straight *Edge* or *DatumAxis* object specifying the extrude direction. *line* must be perpendicular to the plane formed by *edges*.

sense A SymbolicConstant specifying the direction of the extrusion. Possible values are FORWARD and REVERSE. If *sense*=FORWARD, the extrusion is in the direction of *line*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByPatchNCorners(...)

This method partitions a cell using an N-sided cutting patch defined by the given corner points.

Path

mdb.models[*name*].parts[*name*].PartitionCellByPatchNCorners

mdb.models[*name*].rootAssembly.PartitionCellByPatchNCorners

Required arguments

cell
A *Cell* object specifying the cell to partition.

cornerPoints
A sequence of Vertex, InterestingPoint, or DatumPoint objects. $3 \leq$ len(*cornerPoints*) ≤ 5. The corner points must not coincide.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByPatchNEdges(...)

This method partitions a cell using an N-sided cutting patch defined by the given edges.

Path

mdb.models[*name*].parts[*name*].PartitionCellByPatchNEdges

mdb.models[*name*].rootAssembly.PartitionCellByPatchNEdges

Required arguments

cell
A *Cell* specifying the cell to partition.

edges A sequence of *Edge* objects bounding the patch. The edges must form a closed loop. The *Edge* objects must belong to the same *PartInstance* object as *cell*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByPlaneNormalToEdge(...)

This method partitions one or more cells using a plane normal to an edge at the given edge point.

Path

mdb.models[*name*].parts[*name*].PartitionCellByPlaneNormalToEdge

mdb.models[*name*].rootAssembly.PartitionCellByPlaneNormalToEdge

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

edge An *Edge* object specifying the normal to the plane.

point A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on *edge*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByPlanePointNormal(...)

This method partitions one or more cells using a plane defined by a point and a normal direction.

Path

mdb.models[*name*].parts[*name*].PartitionCellByPlanePointNormal

mdb.models[*name*].rootAssembly.PartitionCellByPlanePointNormal

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

point A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on the plane.

normal A straight *Edge* or *DatumAxis* object specifying the normal to the plane.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellByPlaneThreePoints(...)

This method partitions one or more cells using a plane defined by three points.

Path

mdb.models[*name*].parts[*name*].PartitionCellByPlaneThreePoints

mdb.models[*name*].rootAssembly.PartitionCellByPlaneThreePoints

Required arguments

cells A sequence of *Cell* objects specifying the cells to partition.

point1 A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on the plane.

point2 A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on the plane.

point3 A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on the plane.



Note:

point1, *point2*, and *point3* must not be colinear and must not coincide.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionCellBySweepEdge(...)

This method partitions one or more cells by sweeping selected edges along the given sweep path.

Path

mdb.models[*name*].parts[*name*].PartitionCellBySweepEdge

mdb.models[*name*].rootAssembly.PartitionCellBySweepEdge

Required arguments

cells
A sequence of *Cell* objects specifying the cells to partition.

edges
A sequence of *Edge* objects to be swept. The edges must be in the same plane. The edges must form a continuous chain without branches. The Edge objects must all belong to the same *PartInstance* object.

sweepPath
An *Edge* object specifying the sweep path. The start of *sweepPath* must be in the plane and perpendicular to the plane formed by *edges*. The sweep path must be planar.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionEdgeByDatumPlane(...)

This method partitions an edge where it intersects with a datum plane.

Path

mdb.models[*name*].parts[*name*].PartitionEdgeByDatumPlane

mdb.models[*name*].rootAssembly.PartitionEdgeByDatumPlane

Required arguments

edges
A sequence of Edge objects specifying the edges to partition.

datumPlane A *DatumPlane* object specifying the location of the partition.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionEdgeByParam(...)

This method partitions one or more edges at the given normalized edge parameter.

Path

mdb.models[*name*].parts[*name*].PartitionEdgeByParam

mdb.models[*name*].rootAssembly.PartitionEdgeByParam

Required arguments

edges A sequence of Edge objects specifying the edges to partition.

parameter A Float specifying the normalized distance along *edge* at which to partition. Possible values are 0.0 <*parameter*< 1.0.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionEdgeByPoint(...)

This method partitions an edge at the given point.

Path

mdb.models[*name*].parts[*name*].PartitionEdgeByPoint

mdb.models[*name*].rootAssembly.PartitionEdgeByPoint

Required arguments

edge An Edge object specifying the edge to partition.

point An *InterestingPoint* or *DatumPoint* object specifying a point on *edge*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceByAuto(...)

This method automatically partitions a target face into simple regions that can be meshed using a structured meshing technique.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByAuto

mdb.models[*name*].rootAssembly.PartitionFaceByAuto

Required arguments

face A *Face* object specifying the face to partition.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

PartitionFaceByCurvedPathEdgeParams(...)

This method partitions a face normal to two edges, using a curved path between the two given edge points defined by the normalized edge parameters.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByCurvedPathEdgeParams

mdb.models[*name*].rootAssembly.PartitionFaceByCurvedPathEdgeParams

Required arguments

face	A *Face* object specifying the face to partition.
edge1	An *Edge* object specifying the start of the partition. The edge must belong to *face*.
parameter1	A Float specifying the distance along *edge1* at which to partition. Possible values are 0.0 ≤*distance1*≤ 1.0.
edge2	An *Edge* object specifying the end of the partition. The edge must belong to *face*.
parameter2	A Float specifying the distance along *edge2* at which to partition. Possible values are 0.0 ≤*distance2*≤ 1.0.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceByCurvedPathEdgePoints(...)

This method partitions a face normal to two edges, using a curved path between the two given edge points.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByCurvedPathEdgePoints

mdb.models[*name*].rootAssembly.PartitionFaceByCurvedPathEdgePoints

Required arguments

face	A *Face* object specifying the face to partition.
edge1	An *Edge* object specifying the start of the partition. The edge must belong to *face*.
point1	A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on *edge1*.
edge2	An *Edge* object specifying the end of the partition. The edge must belong to *face*.
point2	A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying a point on *edge2*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceByDatumPlane(...)

This method partitions one or more faces using the given datum plane.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByDatumPlane

mdb.models[*name*].rootAssembly.PartitionFaceByDatumPlane

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

datumPlane A *DatumPlane* object specifying the location of the partition.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

PartitionFaceByExtendFace(...)

This method partitions one or more faces by extending the underlying geometry of another given face to partition the target faces.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByExtendFace

mdb.models[*name*].rootAssembly.PartitionFaceByExtendFace

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

extendFace A *Face* object that is to be extended to create the partition. The face to extend can be a planar, cylindrical, conical, or spherical face.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

PartitionFaceByIntersectFace(...)

This method partitions one or more faces using the given cutting faces to partition the target faces.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByIntersectFace

mdb.models[*name*].rootAssembly.PartitionFaceByIntersectFace

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

cuttingFaces A sequence of Face objects that specify the cutting faces.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

PartitionFaceByProjectingEdges(...)

This method partitions one or more faces by projecting the given edges on the target faces.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByProjectingEdges

mdb.models[*name*].rootAssembly.PartitionFaceByProjectingEdges

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

edges A sequence of *Edge* objects specifying the edges that will be projected onto the target faces.

Optional arguments

extendEdges A boolean specifying whether to extend the given edges at their free ends in the tangent direction before partitioning the target faces. The default value is False.

Return value

A Feature object.

Exceptions

None.

PartitionFaceByShortestPath(...)

This method partitions one or more faces using a minimum distance path between the two given points.

Path

mdb.models[*name*].parts[*name*].PartitionFaceByShortestPath

mdb.models[*name*].rootAssembly.PartitionFaceByShortestPath

Required arguments

faces A sequence of *Face* objects specifying the face to partition.

point1 A *Vertex*, *InterestingPoint*, or *DatumPoint* object.

point2 A *Vertex*, *InterestingPoint*, or *DatumPoint* object.



Note:

point1 and *point2* must not coincide, and they must both lie on the underlying surface geometry of at least one of the target faces.

Optional arguments

None.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceBySketch(...)

This method partitions one or more planar faces by sketching on them.

Path

mdb.models[*name*].parts[*name*].PartitionFaceBySketch

mdb.models[*name*].rootAssembly.PartitionFaceBySketch

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

sketch A *ConstrainedSketch* object specifying the partition.

Optional arguments

sketchUpEdge An *Edge* or *DatumAxis* object specifying the orientation of *sketch*. This edge or datum axis must not be orthogonal to the plane defined by *faces*. If unspecified, *sketch* is assumed to be oriented in with the *Y* direction pointing up.

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceBySketchDistance(...)

This method partitions one or more faces by sketching on a sketch plane and then projecting the sketch toward the target faces through the given distance.

Path

mdb.models[*name*].parts[*name*].PartitionFaceBySketchDistance

mdb.models[*name*].rootAssembly.PartitionFaceBySketchDistance

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

sketchPlane A planar Face or DatumPlane object.

sketchPlaneSide A SymbolicConstant specifying the side of the plane to be used for sketching. Possible values are SIDE1 and SIDE2.

sketchUpEdge An Edge object specifying the orientation of *sketch*. This edge must not be orthogonal to *sketchPlane*.

sketch A *ConstrainedSketch* object specifying the partition.

distance A Float specifying the projection distance. Possible values are *distance*> 0.0.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceBySketchRefPoint(...)

This method partitions one or more faces by sketching on a sketch plane and then projecting the sketch toward the target faces through a distance governed by the reference point.

Path

mdb.models[*name*].parts[*name*].PartitionFaceBySketchRefPoint

mdb.models[*name*].rootAssembly.PartitionFaceBySketchRefPoint

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

sketchPlane A planar Face or DatumPlane object.

sketchUpEdge An *Edge* object or a DatumAxis object specifying the orientation of *sketch*. This edge or datum axis must not be orthogonal to *sketchPlane*.

sketch A *ConstrainedSketch* object specifying the partition.

point A *Vertex*, *InterestingPoint*, or *DatumPoint* object specifying the distance to project *sketch*. The point must not lie on *sketchPlane*.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

AbaqusException.

PartitionFaceBySketchThruAll(...)

This method partitions one or more faces by sketching on a sketch plane and then projecting toward the target faces through an infinite distance.

Path

mdb.models[*name*].parts[*name*].PartitionFaceBySketchThruAll

mdb.models[*name*].rootAssembly.PartitionFaceBySketchThruAll

Required arguments

faces A sequence of *Face* objects specifying the faces to partition.

sketchPlane A planar Face or DatumPlane object.

sketchPlaneSide A SymbolicConstant specifying the extrude direction of the sketch. Possible values are SIDE1 and SIDE2.

sketchUpEdge An *Edge* or a DatumAxis object specifying the orientation of *sketch*. This edge or datum axis must not be orthogonal to *sketchPlane*.

sketch A *ConstrainedSketch* object specifying the partition.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

AbaqusException.

ReferencePoint(...)

This method creates a Feature object and a *ReferencePoint* object at the specified location.

Path

mdb.models[*name*].rootAssembly.ReferencePoint

mdb.models[*name*].parts[name].ReferencePoint

Required arguments

point A *Vertex*, *InterestingPoint*, a *MeshNode*, or a *Datum* object specifying a reference point. *point* can also be a sequence of three Floats representing the *X*-, *Y*-, and *Z*-coordinates of the point.

Optional arguments

instanceName Used internally by the input file writer.

Return value

A Feature Object.

Exceptions

None.

RemoveWireEdges(...)

This method removes wire edges.

Path

mdb.models[*name*].parts[*name*].RemoveWireEdges

mdb.models[*name*].rootAssembly.RemoveWireEdges

Required arguments

wireEdgeList A sequence of Edge objects specifying the edges to remove. Any specified edge that is not a wire edge will not be removed.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

WirePolyLine(...)

This method creates an additional Feature object by creating a series of wires joining points in pairs. When such a feature is created at the Part level, then each point can be either a datum point, a vertex, a reference point, an interesting point, an orphan mesh node, or the coordinates of a point. When such a feature is created at the Assembly level, then each point can only be a vertex, a reference point, or an orphan mesh node.

Path

mdb.models[*name*].parts[*name*].WirePolyLine

mdb.models[*name*].rootAssembly.WirePolyLine

Required arguments

points A tuple of point pairs, each pair being itself represented by a tuple. For part level features each point can be a Vertex, Datum point, Reference point, orphan mesh Node, or InterestingPoint object specifying the points through which the polyline wire will pass. Each point can also be a tuple of Floats representing the coordinates of a point. For assembly level features each point can only be a Vertex, Reference point, or orphan mesh Node specifying the points through which the polyline wire will pass (coordinates cannot be specified). In any of the pairs, the first or second point can be NONE. In that case, the point pair will create a zero-length wire, which is required for certain types of connectors. You must specify at least one pair.

Optional arguments

mergeType A SymbolicConstant specifying the merge behavior of the wire with existing geometry. If *mergeType* is MERGE, Abaqus merges the wire into solid regions of the part if the wire passes through them. If *mergeType* is IMPRINT, Abaqus imprints the wire on existing geometry as edges. If *mergeType* is SEPARATE, Abaqus neither merges nor imprints the spline wire with existing geometry. It creates the wire separately. The default value is IMPRINT.

meshable A Boolean specifying whether the wire should be available for selection for meshing operations. If *meshable*=OFF, the wire can be used for connector section assignment. The default value is ON.

Return value

A Feature object.

Exceptions

None.

`isSuppressed()`

This method queries the suppressed state of the feature.

Arguments

None.

Return value

A Boolean value of True if the feature is suppressed and False if it is not suppressed.

Exceptions

None.

restore()

This method restores the parameters of a feature to the value they had when the `backup` method was invoked on the part or assembly. Use the `restore` method after the `backup` method.

Arguments

None.

Return value

```
None.
```

Exceptions

None.

resume()

This method resumes suppressed features. Resuming a feature fully restores it to the part or assembly. You can resume the last feature you suppressed, all suppressed features, or just selected features. When you resume a child feature, Abaqus/CAE also resumes the parent features automatically.

Arguments

None.

Return value

```
None.
```

Exceptions

None.

setValues(...)

This method modifies the Feature object.

Required arguments

None.

Optional arguments

parameter A Float specifying the normalized distance along *edge* at which to partition. Possible values are 0.0 <*parameter*< 1.0. You use this argument to modify a partition created with the created with the PartitionEdgeByParam method.

parameter1 A Float specifying the distance along *edge1* at which to partition. Possible values are 0.0 ≤*parameter1*≤ 1.0. You use this argument to modify a partition object created with the PartitionFaceByCurvedPathEdgeParam method.

parameter2 A Float specifying the distance along *edge2* at which to partition. Possible values are 0.0 ≤*parameter2*≤ 1.0. You use this argument to modify a partition object created with the PartitionFaceByCurvedPathEdgeParam method.

sketch A *ConstrainedSketch* object specifying the partition. You use this argument to modify a partition object created with a sketch; for example, using the PartitionFaceBySketch method.

distance A Float specifying the projection *distance*. Possible values are *distance*> 0.0. You use this argument to modify a partition object created with the PartitionFaceBySketchDistance method.

Return value

`None.`

Exceptions

AbaqusException.

`suppress()`

This method suppresses features. Suppressing a feature is equivalent to temporarily removing the feature from the part or assembly. Suppressed features remain suppressed when you regenerate a part or assembly. You cannot suppress the base feature. In addition, if you suppress a parent feature, all of its child features are also suppressed automatically. Suppressed features can be restored with the `resume` command.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Feature object has the following members:

name A String specifying the repository key.

id

An Int specifying the ID of the feature.

FeatureOptions object

The FeatureOptions object stores the options that control the behavior of feature regeneration for all features in a model.

Access

```
import part
mdb.models[name].featureOptions
```

setValues(...)

This method modifies the FeatureOptions object for the specified model.

Required arguments

None.

Optional arguments

checkSelfIntersection
A Boolean specifying whether Abaqus/CAE should perform self-intersection checks while regenerating features. The default value is ON.

autoCaching
A Boolean specifying whether geometric states should be automatically cached. The default value is ON.

maxCachedStates
An Int specifying the maximum number of caches to be stored with each part or with the assembly. The default value is 5.

Return value

None.

Exceptions

None.

Members

The FeatureOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

Field commands

A Field object stores the non-propagating data of a field as well as a number of instances of the corresponding FieldState object.
The FieldState object stores the propagating data of the field in a single step. A specific type of Field object and a specific type of FieldState object are designed for each type of predefined field. Instances of the FieldState object are created and deleted internally by its corresponding Field object.

In this section:

- *Field object*
- *AnalyticalField object*
- *DataTable object*
- *DiscreteField object*
- *ExpressionField object*
- *MappedField object*
- *OdbMeshRegionData object*

Field object

The Field object is the abstract base type for other Field objects. The Field object has no explicit constructor. The methods and members of the Field object are common to all objects derived from the Field.

Access

```
import fields
```

Members

The Field object can have the following members:

name	A String specifying the repository key.
description	A String specifying the description of the field. The default value is an empty string.

AnalyticalField object

The AnalyticalField object is the abstract base type for other AnalyticalField objects. The AnalyticalField object has no explicit constructor. The methods and members of the AnalyticalField object are common to all objects derived from the AnalyticalField.

The AnalyticalField object is derived from the *Field* object.

Access

```
import fields
mdb.models[name].analyticalFields[name]
```

Members

The AnalyticalField object can have the following members:

name
A String specifying the repository key.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the field. If *localCsys*=`None`, the field is defined in the global coordinate system. The default value is `None`.

description
A String specifying the description of the field. The default value is an empty string.

DataTable object

A DataTable is an object used to define the domain and data for a *DiscreteField*.

Access

```
import field
mdb.models[name].discreteFields[name].data[i]
```

Members

The DataTable object has the following members:

dataWidth An Int specifying the width of the data. Valid widths are 1, 6, 21, corresponding to scalar data, orientations and 4D tensors.

name A String specifying the index.

instanceName A String specifying the instance name.

domain A tuple of Ints specifying the domain node, element or integration point identifiers.

table A tuple of Floats specifying the data within the domain.

DiscreteField object

The DiscreteField object defines a varying field whose values correspond to distinct points within a domain.

The DiscreteField object is derived from the *Field* object.

Access

```
import fields
mdb.models[name].discreteFields[name]
```

DiscreteField(...)

This method creates a DiscreteField object.

Path

```
mdb.models[name].DiscreteField
```

Required arguments

name A String specifying the repository key.

defaultValues A sequence of Floats specifying a sequence of floats specifying the default values.

fieldType A SymbolicConstant or an Int specifying the type of data represented by this discrete field. Possible values are SCALAR, ORIENTATION, and PRESCRIBEDCONDITION_DOF.

Optional arguments

location A SymbolicConstant or an Int specifying the location of the domain data. Possible values are NODES and ELEMENTS. The default value is NODES.

dataWidth An Int specifying the width of the supplied data. The default value is 1.

data A *DataTableArray* object.

description A String specifying the description of the field. The default value is an empty string.

orientationType A SymbolicConstant specifying the type of the system being described by a discrete field used for an orientation. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL. The default value is CARTESIAN.

partLevelOrientation A Boolean specifying whether or not the orientations are described in terms of part level coordinates. The default value is OFF.

Return value

A DiscreteField object.

Exceptions

AbaqusException.

DiscreteFieldByVolumeFraction(...)

This method creates a DiscreteField object that represents the volume fraction of each element of an Eulerian Instance that is occupied by a reference instance.

Path

mdb.models[*name*].rootAssembly.DiscreteFieldByVolumeFraction

Required arguments

name	A String specifying the repository key.
eulerianInstance	A *PartInstance* object specifying the elements for which volume fraction values will be computed.
referenceInstance	A *PartInstance* object specifying the region that either contains material or is empty of material.

Optional arguments

accuracy	A Symbolic Constant specifying the level of accuracy that will be used in computing volume fractions. Possible values are LOW, MEDIUM, or HIGH. The default value is MEDIUM.
materialLocation	A Symbolic Constant indicating whether the material is inside or outside the *referenceInstance*. Possible values are INSIDE or OUTSIDE. The default value is INSIDE.
description	A String specifying the description of the field. The default value is an empty string.
scaleFactor	A float specifying the fraction of the volume that is occupied by the *referenceInstance*. Valid values are between 0 and 1.

Return value

A DiscreteField object.

Exceptions

AbaqusException.

DiscreteFieldFromAnalytic(...)

This method creates a DiscreteField object from a *AnalyticalField* object.

Path

```
mdb.models[name].DiscreteFieldFromAnalytic
```

Required arguments

name	A String specifying the repository key.
location	A SymbolicConstant or an Int specifying the location of the domain data. Possible values are NODES and ELEMENTS. The default value is NODES.
analyticFieldName	A String specifying the name of the *AnalyticalField* containing the source data.
region	A *Region* object for the field.

Optional arguments

None.

Return value

A DiscreteField object.

Exceptions

AbaqusException.

`setValues(...)`

This method modifies the DiscreteField object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DiscreteField* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DiscreteField object has members with the same names and descriptions as the arguments to the *DiscreteField* method.

ExpressionField object

The ExpressionField object defines a spatially varying field whose value is calculated from a user-supplied mathematical expression.

The ExpressionField object is derived from the *AnalyticalField* object.

Access

```
import fields
mdb.models[name].analyticalFields[name]
```

ExpressionField(...)

This method creates an ExpressionField object.

Path

```
mdb.models[name].ExpressionField
```

Required arguments

name
A String specifying the repository key.

expression
A String specifying the Python expression to evaluate in space. Variables are X, Y, and Z; R, Th, and Z; or R, Th, and P based on the selected coordinate system.

Optional arguments

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the field. If *localCsys*=`None`, the field is defined in the global coordinate system. The default value is `None`.

description
A String specifying the description of the field. The default value is an empty string.

Return value

An ExpressionField object.

Exceptions

TextException.

setValues(...)

This method modifies the ExpressionField object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ExpressionField* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ExpressionField object has members with the same names and descriptions as the arguments to the *ExpressionField* method.

MappedField object

The MappedField object defines a spatially varying field whose value is calculated from an external source data.

The MappedField object is derived from the *AnalyticalField* object.

Access

```
import fields
mdb.models[name].analyticalFields[name]
```

MappedField(...)

This method creates an MappedField object.

Path

```
mdb.models[name].MappedField
```

Required arguments

name
A String specifying the repository key.

Optional arguments

regionType
A SymbolicConstant specifying the data source region type. It can be either an ODB mesh or a cloud of points. Possible values are MESH and POINT. The default value is POINT.

partLevelData
A Boolean specifying whether or not the point cloud source data are described in terms of part level coordinates. If part level coordinates is employed, the local coordinate system defined in *localCsys* will be ignored. The default value is OFF.

pointDataFormat
A SymbolicConstant specifying point cloud source data format. Possible values are GRID and XYZ. The default value is XYZ.

gridPointPlane
A SymbolicConstant specifying the plane on which the point cloud source data of grid format are described. Possible values are XYPLANE, YZPLANE, and XZPLANE. The default value is XYPLANE.

defaultUnMappedValue
A Float specifying the parameter (field) value reported when a value cannot be calculated from the data source. The default value is 0.0.

mappingAlgorithm
A SymbolicConstant specifying the mapping algorithm for target surface, or on mesh target model when the parameter value are located at nodes, for example nodal temperatures. Possible values are SURFACE and VOLUMETRIC. The default value is SURFACE.

searchTolType
A SymbolicConstant specifying searching tolerance type in terms of absolute value or a fraction of the average of all element characteristic length in target model region. Possible values are ABSOLUTE and RELATIVE. The default value is RELATIVE.

boundarySearchTol
A Float specifying the search distance tolerance value on the exterior boundary of target model region. Source points within this distance will be included in computing the parameter value of target region. This tolerance applies to both surface and volumetric mapping. The default value is 0.01.

neighborhoodSearchTol
A Float specifying the search distance tolerance value used for distance weighting algorithm. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 1000000.0.

negativeNormalSearchTol
A Float specifying the search distance tolerance value in the negative normal of target surface region. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 0.15.

positiveNormalSearchTol
A Float specifying the search distance tolerance value in the positive normal of target surface region. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 0.05.

scaleCoordinates
A Boolean specifying whether or not to scale the user-supplied coordinate values from the point cloud or indicated ODB. The default value is OFF.

gridPointData
A sequence of sequences of Floats specifying the point cloud source data of grid format. The default value is an empty sequence.

xyzPointData
A sequence of sequences of Floats specifying the point cloud source data of XYZ format. Each data item is defining the XYZ coordinates of a point and its field value. The default value is an empty sequence.

coordinateScalingFactors
A sequence of Floats specifying the scaling factors for the global 1, 2 and 3 directions. The default value is (1.0, 1.0, 1.0).

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the field. If *localCsys*=`None`, the field is defined in the global coordinate system. The default value is `None`.

description
A String specifying the description of the field. The default value is an empty string.

Return value

A MappedField object.

Exceptions

AbaqusException.

`setValues(...)`

This method modifies the MappedField object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MappedField* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The MappedField object has the following members:

name

A String specifying the repository key.

regionType

A SymbolicConstant specifying the data source region type. It can be either an ODB mesh or a cloud of points. Possible values are MESH and POINT. The default value is POINT.

partLevelData

A Boolean specifying whether or not the point cloud source data are described in terms of part level coordinates. If part level coordinates is employed, the local coordinate system defined in *localCsys* will be ignored. The default value is OFF.

pointDataFormat

A SymbolicConstant specifying point cloud source data format. Possible values are GRID and XYZ. The default value is XYZ.

gridPointPlane

A SymbolicConstant specifying the plane on which the point cloud source data of grid format are described. Possible values are XYPLANE, YZPLANE, and XZPLANE. The default value is XYPLANE.

defaultUnMappedValue

A Float specifying default parameter (field) value of target model region while its value cannot be calculated from the data source. The default value is 0.0.

mappingAlgorithm

A SymbolicConstant specifying the mapping algorithm for target surface, or on mesh target model when the parameter value are located at nodes, for example nodal temperatures. Possible values are SURFACE and VOLUMETRIC. The default value is SURFACE.

searchTolType

A SymbolicConstant specifying searching tolerance type in terms of absolute value or a fraction of the average of all element characteristic length in target model region. Possible values are ABSOLUTE and RELATIVE. The default value is RELATIVE.

boundarySearchTol

A Float specifying the search distance tolerance value on the exterior boundary of target model region. Source points within this distance will be included in computing the

parameter value of target region. This tolerance applies to both surface and volumetric mapping. The default value is 0.01.

neighborhoodSearchTol

A Float specifying the search distance tolerance value used for distance weighting algorithm. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 1000000.0.

negativeNormalSearchTol

A Float specifying the search distance tolerance value in the negative normal of target surface region. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 0.15.

positiveNormalSearchTol

A Float specifying the search distance tolerance value in the positive normal of target surface region. Source points within this distance will be included in computing the parameter value of target region. This tolerance only applies to surface mapping. The default value is 0.05.

scaleCoordinates

A Boolean specifying whether or not to scale the user-supplied coordinate values from the point cloud or indicated ODB. The default value is OFF.

gridPointData

A tuple of tuples of Floats specifying the point cloud source data of grid format. The default value is an empty sequence.

xyzPointData

A tuple of tuples of Floats specifying the point cloud source data of XYZ format. Each data item is defining the XYZ coordinates of a point and its field value. The default value is an empty sequence.

odbMeshRegionData

An *OdbMeshRegionData* object specifying the external source data from ODB mesh region.

coordinateScalingFactors

A tuple of Floats specifying the scaling factors for the global 1, 2 and 3 directions. The default value is (1.0, 1.0, 1.0).

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the field. If *localCsys*=`None`, the field is defined in the global coordinate system. The default value is `None`.

description

A String specifying the description of the field. The default value is an empty string.

OdbMeshRegionData object

The OdbMeshRegionData object defines the external source data of *MappedField* from an ODB file.

Access

```
import
field
mdb.models[name].analyticalFields[name].odbMeshRegionData
```

OdbMeshRegionData(...)

This method creates an OdbMeshRegionData object.

Path

```
mdb.models[name].analyticalFields[name].OdbMeshRegionData
```

Required arguments

odbFileName

A String specifying the name of the output database file (including the .odb extension) to be read into as the source data. This String can also be the full path to the output database file if it is located in another directory.

variableLabel

A String specifying the field output variable.

Optional arguments

stepIndex

An Int specifying the step index. Possible values are $0 \le stepIndex \le (numSteps - 1)$. The default value is 0.

frameIndex

An Int specifying the frame in the specified step. Valid values are $0 \le frameIndex \le (numFramesInStep - 1)$. The default value is 0.

outputPosition

A SymbolicConstant specifying the position where the data is written in the output database. Data can be obtained only from the position at which it was written to the output database during the analysis. This position should be aligned with the field output variable. Possible values are:

- UNDEFINED_POSITION
- NODAL
- INTEGRATION_POINT
- ELEMENT_FACE
- ELEMENT_NODAL
- ELEMENT_CENTROID
- WHOLE_ELEMENT
- WHOLE_REGION
- WHOLE_PART_INSTANCE

- WHOLE_MODEL
- GENERAL_PARTICLE

The default value is UNDEFINED_POSITION.

dataType

A SymbolicConstant specifying the data type of the field output variable which should be aligned with the variable. Currently only SCALAR is supported. Possible values are:

- ENUMERATION
- BOOLEAN
- INTEGER
- SCALAR
- VECTOR
- QUATERNION_2D
- QUATERNION_3D
- TENSOR
- TENSOR_3D_FULL
- TENSOR_3D_PLANAR
- TENSOR_3D_SURFACE
- TENSOR_2D_PLANAR
- TENSOR_2D_SURFACE

The default value is SCALAR.

storageType

A SymbolicConstant specifying the storage type of the field output variable which should be aligned with the variable. Possible values are FLOAT, DOUBLE, INTEGER, and BOOLEAN. The default value is FLOAT.

quantityToPlot

A SymbolicConstant specifying the quantity to plot. Currently only FIELD_OUTPUT is supported. Possible values are FIELD_OUTPUT and DISCONTINUITIES. The default value is FIELD_OUTPUT.

averageElementOutput

A Boolean specifying whether to average the element output. The default value is OFF.

useRegionBoundaries

A Boolean specifying whether to use region boundaries when averaging. The default value is OFF.

regionBoundaries

A SymbolicConstant specifying the type of averaging region boundaries. Currently only NONE and ODB_REGIONS are supported. Possible values are NONE, ODB_REGIONS, ELEMENT_SET, and DISPLAY_GROUPS. The default value is NONE.

includeFeatureBoundaries

A Boolean specifying whether to include additional averaging boundaries for shells and membranes based on feature edges. The default value is ON.

featureAngle

A Float specifying the feature angle to be used when *includeFeatureBoundaries*=ON. The default value is 20.0.

averageOnlyDisplayed

A Boolean specifying whether to average only values on displayed elements. The default value is OFF.

averagingThreshold

A Float specifying the nodal averaging threshold percentage. 0 ≤*averagingThreshold*≤100. The default value is 75.0.

computeOrder

A SymbolicConstant specifying the order or the computations to be performed on the interested field output variable. Possible values are EXTRAPOLATE_AVERAGE_COMPUTE, EXTRAPOLATE_COMPUTE_AVERAGE, EXTRAPOLATE_COMPUTE, EXTRAPOLATE_COMPUTE_DISCONTINUITIES, and RAW_DATA. The default value is EXTRAPOLATE_COMPUTE_AVERAGE.

numericForm

A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_MAG_AT_ANGLE. The default value is REAL.

complexAngle

A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm=COMPLEX_MAG_AT_ANGLE*=COMPLEX_MAG_AT_ANGLE. The default value is 0.0.

sectionPoint

A Dictionary with String keys and String values. Each key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
                    sectionPoint={'shell < MAT > < 7 section
 points >':'SPOS,
                     (fraction = 1.0)', 'shell < MAT > < 5
section points >':
                     'SPOS, (fraction = 1.0)', }
```

refinementType

A SymbolicConstant specifying the type of the FieldOutput object. Possible values for the SymbolicConstant are NO_REFINEMENT, INVARIANT and COMPONENT. Default argument is NO_REFINEMENT. *refinementType* is mandetory if *variableLabel* has an INVARIANT or a COMPONENT.

refinementLabel

A String specifying the Label of FieldOutput object. This is required only if the *refinementType* is INVARIANT or COMPONENT.

displayOutputPosition

A SymbolicConstant specifying the position from which to obtain the data. Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, ELEMENT_CENTROID, WHOLE_ELEMENT, WHOLE_REGION, WHOLE_PART_INSTANCE, WHOLE_MODEL, and GENERAL_PARTICLE.

Return value

An OdbMeshRegionData object.

Exceptions

TextException.

setValues(...)

This method modifies the OdbMeshRegionData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *OdbMeshRegionData* method.

Return value

`None.`

Exceptions

None.

Members

The OdbMeshRegionData object has the following members:

stepIndex

An Int specifying the step index. Possible values are $0 \le stepIndex \le (numSteps - 1)$. The default value is 0.

frameIndex

An Int specifying the frame in the specified step. Valid values are $0 \le frameIndex \le (numFramesInStep - 1)$. The default value is 0.

outputPosition

A SymbolicConstant specifying the position from which to obtain data. Data can be obtained only from the position at which they were written to the output database during the analysis. This position should be aligned with the field output variable. Possible values are:

- UNDEFINED_POSITION
- NODAL
- INTEGRATION_POINT
- ELEMENT_FACE
- ELEMENT_NODAL
- ELEMENT_CENTROID
- WHOLE_ELEMENT
- WHOLE_REGION
- WHOLE_PART_INSTANCE
- WHOLE_MODEL
- GENERAL_PARTICLE

The default value is UNDEFINED_POSITION.

dataType

A SymbolicConstant specifying the data type of the field output variable which should be aligned with the variable. Currently only SCALAR is supported. Possible values are:

- ENUMERATION
- BOOLEAN
- INTEGER
- SCALAR
- VECTOR
- QUATERNION_2D
- QUATERNION_3D
- TENSOR
- TENSOR_3D_FULL
- TENSOR_3D_PLANAR
- TENSOR_3D_SURFACE
- TENSOR_2D_PLANAR
- TENSOR_2D_SURFACE

The default value is SCALAR.

storageType

A SymbolicConstant specifying the storage type of the field output variable which should be aligned with the variable. Possible values are FLOAT, DOUBLE, INTEGER, and BOOLEAN. The default value is FLOAT.

quantityToPlot

A SymbolicConstant specifying the quantity to plot. Currently only FIELD_OUTPUT is supported. Possible values are FIELD_OUTPUT and DISCONTINUITIES. The default value is FIELD_OUTPUT.

averageElementOutput

A Boolean specifying whether to average the element output. The default value is OFF.

useRegionBoundaries

A Boolean specifying whether to use region boundaries when averaging. The default value is OFF.

regionBoundaries

A SymbolicConstant specifying the type of averaging region boundaries. Currently only NONE and ODB_REGIONS are supported. Possible values are NONE, ODB_REGIONS, ELEMENT_SET, and DISPLAY_GROUPS. The default value is NONE.

includeFeatureBoundaries

A Boolean specifying whether to include additional averaging boundaries for shells and membranes based on feature edges. The default value is ON.

featureAngle

A Float specifying the feature angle to be used when *includeFeatureBoundaries*=ON. The default value is 20.0.

averageOnlyDisplayed

A Boolean specifying whether to average only values on displayed elements. The default value is OFF.

averagingThreshold

A Float specifying the nodal averaging threshold percentage. 0 ≤*averagingThreshold*≤100. The default value is 75.0.

computeOrder

A SymbolicConstant specifying the order or the computations to be performed on the interested field output variable. Possible values are EXTRAPOLATE_AVERAGE_COMPUTE, EXTRAPOLATE_COMPUTE_AVERAGE, EXTRAPOLATE_COMPUTE, EXTRAPOLATE_COMPUTE_DISCONTINUITIES, and RAW_DATA. The default value is EXTRAPOLATE_COMPUTE_AVERAGE.

numericForm

A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_MAG_AT_ANGLE. The default value is REAL.

complexAngle

A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm=COMPLEX_MAG_AT_ANGLE*=COMPLEX_MAG_AT_ANGLE. The default value is 0.0.

odbFileName

A String specifying the name of the output database file (including the .odb extension) to be read into as the source data. This String can also be the full path to the output database file if it is located in another directory.

variableLabel

A String specifying the field output variable.

displayOutputPosition

A SymbolicConstant specifying the position where the output is displayed in the viewport. Possible values are:

- UNDEFINED_POSITION
- NODAL
- INTEGRATION_POINT
- ELEMENT_FACE
- ELEMENT_NODAL
- ELEMENT_CENTROID
- WHOLE_ELEMENT
- WHOLE_REGION
- WHOLE_PART_INSTANCE
- WHOLE_MODEL
- GENERAL_PARTICLE

The default value is UNDEFINED_POSITION.

Field Report commands

Field report commands are used to write a field output report and free body computational results to an ASCII file.

In this section:

- *FieldReportOptions object*
- *FreeBodyReportOptions object*
- *OdbFieldVarList object*
- *OdbModelFieldVarList object*
- *writeFieldReport*
- *writeFreeBodyReport*

FieldReportOptions object

The FieldReportOptions object stores settings used by the `writeFieldReport` method when you write a *FieldOutput* object to an ASCII file. The FieldReportOptions object has no constructor. Abaqus creates the *fieldReportOptions* member when you import the Visualization module.

Access

```
import visualization
session.defaultFieldReportOptions
session.fieldReportOptions
```

`setValues(...)`

This method modifies the FieldReportOptions object.

Required arguments

None.

Optional arguments

numColumns
An Int specifying the number of columns to display for the tabular report. The default value is 80.

numberFormat
A *NumberFormat* object specifying the format type, number of digits and precision used to print the numeric output.

printXYData
A Boolean specifying whether to include *X*–*Y* data values in the tabular report. The default value is ON.

printTotal
A Boolean specifying whether to include column totals in the tabular report. The default value is ON.

printMinMax
A Boolean specifying whether to include column summary minimum and maximum values in the tabular report. The default value is ON.

pageWidth
A SymbolicConstant specifying how the width of the tabular report is to be determined. Possible values are NO_LIMIT and SPECIFY. The default value is NO_LIMIT.

separateTables
A Boolean specifying whether to present a separate tabular report for each output variable. The default value is OFF.

sort
A SymbolicConstant specifying the order in which values are to be sorted within a tabular report. Possible values are ASCENDING and DESCENDING. The default value is ASCENDING.

printLocalCSYS
A Boolean specifying whether to include the local coordinate system values in the tabular report. The default value is OFF.

Return value

A FieldReportOptions object.

Exceptions

None.

Members

The FieldReportOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

FreeBodyReportOptions object

The FreeBodyReportOptions object stores settings used by the `writeFreeBodyReport` method when you write free body computational results to an ASCII file. The FreeBodyReportOptions object has no constructor. Abaqus creates the *freeBodyReportOptions* member when you import the Visualization module.

Access

```
import visualization
session.defaultFreeBodyReportOptions
session.freeBodyReportOptions
```

`setValues(...)`

This method modifies the FreeBodyReportOptions object.

Required arguments

None.

Optional arguments

numDigits	An Int specifying the number of decimal places. The default value is 3.
forceThreshold	A Float specifying the threshold value for force. The default value is 10^{-6}.
momentThreshold	A Float specifying the threshold value for moment. The default value is 10^{-6}.
numberFormat	A SymbolicConstant specifying the number format. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.
reportFormat	A SymbolicConstant specifying the report format. Possible values are NORMAL_ANNOTATED and COMMA_SEPARATED_VALUES. The default value is NORMAL_ANNOTATED.
csysType	A SymbolicConstant specifying the coordinate system type. Possible values are GLOBAL and LOCAL. The default value is GLOBAL.

Return value

A FreeBodyReportOptions object.

Exceptions

None.

Members

The FreeBodyReportOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

OdbFieldVarList object

The read-only OdbFieldVarList object is a sequence listing all variables available for the current step and frame. Each item in the sequence is itself a sequence fully describing the given variable.

Access

```
import visualization
session.viewports[name].layers[name].odbDisplay.fieldVariables
session.viewports[name].odbDisplay.fieldVariables
```

Members

The OdbFieldVarList object has no members.

OdbModelFieldVarList object

The read-only OdbModelFieldVarList object lists all variables available for the model in the current *OdbDisplay* object.

Access

```
import visualization
session.viewports[name].layers[name].odbDisplay.modelVariableList
session.viewports[name].odbDisplay.modelVariableList
```

Members

The OdbModelFieldVarList object has no members.

writeFieldReport

This command writes a field output report to a file.

writeFieldReport

This method writes a FieldOutput object to a user-defined ASCII file.

Path

session.writeFieldReport

Required arguments

filename A String specifying the name of the file to which field output will be written.

append A Boolean specifying whether to append the field output to an existing file. The default value is ON.

sortItem A String specifying the item by which to sort the tabular values.

odb An Odb object from which to obtain field output values.

step An Int (or stepIndex) specifying the step from which to obtain field output values. Possible values are $0 \leq step \leq (numSteps - 1)$.

frame An Int (or frameIndex) specifying the frame from which to obtain field output values. Possible values are $0 \leq frame \leq (numFramesInStep - 1)$.

outputPosition A SymbolicConstant specifying the position from which to obtain data. Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, ELEMENT_CENTROID, WHOLE_ELEMENT, WHOLE_REGION, WHOLE_PART_INSTANCE, WHOLE_MODEL, and GENERAL_PARTICLE.

displayGroup A DisplayGroup object specifying the subset of the model for which to obtain data.

variable A sequence of variable description sequences specifying one or more field output variables for which to obtain data. Each variable description sequence contains the following elements:

- *element0*: A String specifying the name of the variable.
- *element1*: A SymbolicConstant specifying the output position at which to report data. Possible values are ELEMENT_CENTROID, ELEMENT_FACE, ELEMENT_NODAL, GENERAL_PARTICLE, INTEGRATION_POINT, NODAL, WHOLE_ELEMENT, WHOLE_MODEL, WHOLE_PART_INSTANCE, and WHOLE_REGION.
- *element2*: A Sequence of tuples each consisting of a SymbolicConstant specifying the refinement (COMPONENT or INVARIANT), followed by a String specifying the name of a component or invariant for which to obtain values.

  If this element is omitted, data are written for all components and invariants (if applicable). This element is required if *element3* (the following element in the tuple) is included.

- *element3 (if applicable)*: A Dictionary with a String key and a String value specifying a single section point at which to report data. The key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
{'shell < MAT > < 7 section points >':'SPOS,
    (fraction = 1.0)'}
```

  If this element is omitted, data are written for all section points (if applicable).

Optional arguments

numericForm

A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_MAG_AT_ANGLE. The initial value is COMPLEX_MAGNITUDE.

complexAngle

A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_MAG_AT_ANGLE. The initial value is 0.

stepFrame

A SymbolicConstant indicating whether to obtain the values from the specified frame or from all active frames. Possible values are SPECIFY and ALL. The default value is SPECIFY.

Return value

`None.`

Exceptions

None.

writeFreeBodyReport

This command writes a free body output report to a file.

writeFreeBodyReport

This method writes a FreeBody object to a user-defined ASCII file.

Path

session.writeFreeBodyReport

Required arguments

fileName A String specifying the name of the file to which the free body output will be written.

append A Boolean specifying whether to append the free body output to an existing file. The default value is ON.

Optional arguments

step An Int identifying the step from which to obtain values. The default value is the current step.

frame An Int identifying the frame from which to obtain values. The default value is the current frame.

stepFrame A SymbolicConstant indicating whether to obtain the values from the specified frame or from all active frames. Possible values are SPECIFY and ALL. The default value is SPECIFY.

odb An Odb object specifying the output database from which data will be read.

Filter commands

Filter commands are used to create real-time filters of output request data.

In this section:

- *Filter object*
- *ButterworthFilter object*
- *Chebyshev1Filter object*
- *Chebyshev2Filter object*
- *OperatorFilter object*

Filter object

The Filter object is the abstract base type for other Filter objects. The Filter object has no explicit constructor. The methods and members of the Filter object are common to all objects derived from the Filter.

Access

```
import filter
mdb.models[name].filters[name]
import odbFilter
session.odbs[name].filters[name]
```

Members

The Filter object has the following members:

name
A String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency
A Float specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

order
An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation
A SymbolicConstant specifying the filter operator. Possible values are NONE, MIN, MAX, and ABS. The default value is NONE.

halt
A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is OFF.

limit
`None` or a Float specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is `None`.

invariant
A SymbolicConstant specifying the invariant to which filtering is applied. Possible values are NONE, FIRST, and SECOND. The default value is NONE.

ButterworthFilter object

The ButterworthFilter object defines a Butterworth type filter.

The ButterworthFilter object is derived from the *Filter* object.

Access

```
import filter
mdb.models[name].filters[name]
import odbFilter
session.odbs[name].filters[name]
```

ButterworthFilter(...)

This method creates a ButterworthFilter object.

Path

```
mdb.models[name].ButterworthFilter
session.odbs[name].ButterworthFilter
```

Required arguments

name
A String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency
A Float specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

order
An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation
A SymbolicConstant specifying the filter operator. Possible values are NONE, MIN, MAX, and ABS. The default value is NONE.

halt
A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is OFF.

limit
`None` or a Float specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is `None`.

invariant
A SymbolicConstant specifying the invariant to which filtering is applied. Possible values are NONE, FIRST, and SECOND. The default value is NONE.

Return value

A ButterworthFilter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the ButterworthFilter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ButterworthFilter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ButterworthFilter object has members with the same names and descriptions as the arguments to the *ButterworthFilter* method.

Corresponding analysis keywords

- **FILTER*

Chebyshev1Filter object

The Chebyshev1Filter object defines a Chebyshev type 1 filter.

The Chebyshev1Filter object is derived from the *Filter* object.

Access

```
import filter
mdb.models[name].filters[name]
import odbFilter
session.odbs[name].filters[name]
```

Chebyshev1Filter(...)

This method creates a Chebyshev1Filter object.

Path

```
mdb.models[name].Chebyshev1Filter
session.odbs[name].Chebyshev1Filter
```

Required arguments

name
A String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency
A Float specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

rippleFactor
A Float specifying the amount of allowable ripple in the filter. Possible values are non-negative numbers. The default value is 0.225.

order
An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation
A SymbolicConstant specifying the filter operator. Possible values are NONE, MIN, MAX, and ABS. The default value is NONE.

halt
A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is OFF.

limit
`None` or a Float specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is `None`.

invariant
A SymbolicConstant specifying the invariant to which filtering is applied. Possible values are NONE, FIRST, and SECOND. The default value is NONE.

Return value

A Chebyshev1Filter object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the Chebyshev1Filter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Chebyshev1Filter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The Chebyshev1Filter object has members with the same names and descriptions as the arguments to the *Chebyshev1Filter* method.

Corresponding analysis keywords

- **FILTER*

Chebyshev2Filter object

The Chebyshev2Filter object defines a Chebyshev type 2 filter.

The Chebyshev2Filter object is derived from the *Filter* object.

Access

```
import filter
mdb.models[name].filters[name]
import odbFilter
session.odbs[name].filters[name]
```

Chebyshev2Filter(...)

This method creates a Chebyshev2Filter object.

Path

```
mdb.models[name].Chebyshev2Filter
session.odbs[name].Chebyshev2Filter
```

Required arguments

name A String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency A Float specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

rippleFactor A Float specifying the amount of allowable ripple in the filter. Possible values are non-negative numbers less than 1. The default value is 0.025.

order An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation A SymbolicConstant specifying the filter operator. Possible values are NONE, MIN, MAX, and ABS. The default value is NONE.

halt A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is OFF.

limit `None` or a Float specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is `None`.

invariant A SymbolicConstant specifying the invariant to which filtering is applied. Possible values are NONE, FIRST, and SECOND. The default value is NONE.

Return value

A Chebyshev2Filter object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the Chebyshev2Filter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Chebyshev2Filter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The Chebyshev2Filter object has members with the same names and descriptions as the arguments to the *Chebyshev2Filter* method.

Corresponding analysis keywords

- **FILTER*

OperatorFilter object

The OperatorFilter object defines an operator filter.

The OperatorFilter object is derived from the *Filter* object.

Access

```
import filter
mdb.models[name].filters[name]
import odbFilter
session.odbs[name].filters[name]
```

OperatorFilter(...)

This method creates an OperatorFilter object.

Path

```
mdb.models[name].OperatorFilter
session.odbs[name].OperatorFilter
```

Required arguments

name
A String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency
A Float specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

order
An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation
A SymbolicConstant specifying the filter operator. Possible values are NONE, MIN, MAX, and ABS. The default value is NONE.

halt
A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is OFF.

limit
`None` or a Float specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is `None`.

invariant
A SymbolicConstant specifying the invariant to which filtering is applied. Possible values are NONE, FIRST, and SECOND. The default value is NONE.

Return value

An OperatorFilter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the OperatorFilter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *OperatorFilter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The OperatorFilter object has members with the same names and descriptions as the arguments to the *OperatorFilter* method.

Corresponding analysis keywords

- **FILTER*

Input File Parser commands

The `inpParser` module provides functions that allow you to parse an Abaqus input file.

In this section:

- *InputFile object*
- *Keyword object*
- *AbaqusNDarray object*

InputFile object

The InputFile object is used to store the definitions in an Abaqus input file. InputFile objects can be created using the methods described in this section.

Access

```
import inpParser
```

InputFile(...)

This method creates an InputFile object by reading an Abaqus input file.

Path

inpParser.InputFile

Required arguments

file A String specifying the path to the input file.

Optional arguments

directory A String specifying the path to the directory containing the input file.

Return value

An InputFile object.

Exceptions

None.

parse(...)

This method parses the input file associated with the InputFile object.

Optional arguments

organize A Boolean specifying whether keywords should be organized into suboptions. The default is False.

verbose A Boolean specifying whether verbose output is to be printed. If *verbose* is True, information about fatal errors is printed. If no fatal errors occur, there is no output. The default is False.

bulk A Boolean specifying whether the input file includes bulk data that should be parsed. The default is True.

usePyArray A Boolean specifying that parse method can return an AbaqusNDarray object for a keyword data value. In cases where large amounts of numerical data (i.e., large node arrays) are expected, it is recommended that you use the option `usePyArray=True`. The default is False.

Return value

A KeywordSequence object.

Exceptions

If you parse an input file more than once, a ValueError is raised for each subsequent parsing.

Members

The InputFile object has the following members:

file A String specifying the source file name of the Abaqus input file.

directory A String specifying the directory where the input file is located.

includes A sequence of Strings specifying any additional input files included in the specified input file.

missingIncludes A sequence of Strings for input files included in the specified input file that could not be located.

Keyword object

The Keyword object is used to store a keyword definition from an Abaqus input file.

Keyword objects are returned via the `InputFile.parse()` method.

Access

```
import inpParser
```

Members

The InputFile object has the following members:

name
A String specifying the name of the keyword.

parameter
A Dictionary of Strings specifying the keyword parameters.

data
A sequence of sequences or an AbaqusNDarray object specifying the keyword data. The type of the leaf objects depends on the keyword. The AbaqusNDarray object is returned only if the data is suitable and if the `InputFile.parse()` method was called with the option `usePyArray=True`. In cases where large amounts of numerical data (i.e., large node arrays) are expected, it is recommended that you use the option `usePyArray=True`.

suboptions
A KeywordSequence specifying the suboptions of the keyword.

comments
A sequence of Strings specifying the comments.

AbaqusNDarray object

The AbaqusNDarray object is a sequence object derived from `numpy.ndarray` and is used to store numeric keyword data from an Abaqus input file. This object is similar to the `numpy.ndarray` object, but the numeric elements are returned as standard Python objects, not `numpy` numeric types.

The numeric elements can be:

- All ints.
- All floats.
- First column int, all other columns floats.

In the last of these cases, the member *colZeroIsInt* will be True; in the other two cases, it will be False.

Interaction commands

A specific type of interaction object and a specific type of interaction state object are designed for each type of interaction.
An interaction object stores the non-propagating data of an interaction as well as a number of instances of the corresponding interaction state object, each of which stores the propagating data of the interaction in a single step. Instances of the interaction state object are created and deleted internally by its corresponding interaction object.

In this section:

- *Interaction object*
- *AcousticImpedance object*
- *AcousticImpedanceProp object*
- *AcousticImpedanceState object*
- *ActuatorSensor object*
- *ActuatorSensorProp object*
- *ActuatorSensorState object*
- *CavityRadiation object*
- *CavityRadiationProp object*
- *CavityRadiationState object*
- *CohesiveBehavior object*
- *ConcentratedFilmCondition object*
- *ConcentratedFilmConditionState object*
- *ConcentratedRadiationToAmbient object*
- *ConcentratedRadiationToAmbientState object*
- *ContactControl object*
- *ContactDamage object*
- *ContactDamping object*
- *ContactExp object*
- *ContactInitialization object*
- *ContactMassScalingExp object*
- *ContactProperty object*
- *ContactPropertyAssignment object*
- *ContactStabilization object*
- *ContactStd object*
- *ContactTangentialBehavior object*
- *CyclicSymmetry object*
- *CyclicSymmetryState object*
- *ElasticFoundation object*
- *ElasticFoundationState object*
- *ExpContactControl object*
- *ExpInitialization object*
- *FilmCondition object*
- *FilmConditionProp object*

- *FilmConditionState object*
- *FluidCavity object*
- *FluidCavityProperty object*
- *FluidCavityState object*
- *FluidExchangeActivation object*
- *FluidExchangeActivationState object*
- *FluidExchange object*
- *FluidExchangeProperty object*
- *FluidExchangeState object*
- *FluidInflatorActivation object*
- *FluidInflatorActivationState object*
- *FluidInflator object*
- *FluidInflatorProperty object*
- *FluidInflatorState object*
- *FractureCriterion object*
- *GapElectricalConductance object*
- *GapHeatGeneration object*
- *GeometricProperties object*
- *IncidentWave object*
- *IncidentWaveProperty object*
- *IncidentWaveState object*
- *InitializationAssignment object*
- *InteractionProperty object*
- *InteractionState object*
- *MainSecondaryAssignment object*
- *Model object*
- *ModelChange object*
- *NormalBehavior object*
- *PolarityAssignments object*
- *PressurePenetration object*
- *PressurePenetrationState object*
- *Radiation object*
- *RadiationToAmbient object*
- *RadiationToAmbientState object*
- *RegionPairs object*
- *SelfContactExp object*
- *SelfContactExpState object*
- *SelfContactStd object*
- *SelfContactStdState object*
- *SlidingFormulationAssignment object*
- *slidingTransitionAssignments object*
- *SmoothingAssignment object*
- *StabilizationAssignment object*
- *StdContactControl object*
- *StdInitialization object*
- *StdStabilization object*

- *StdXplCosimulation object*
- *StdXplCosimulationState object*
- *SurfaceBeamSmoothingAssignment object*
- *SurfaceCrushTriggerAssignment object*
- *SurfaceFeatureAssignment object*
- *SurfaceFrictionAssignment object*
- *SurfaceOffsetAssignment object*
- *SurfaceThicknessAssignment object*
- *SurfaceToSurfaceContactExp object*
- *SurfaceToSurfaceContactStd object*
- *SurfaceToSurfaceExpState object*
- *SurfaceToSurfaceStdState object*
- *SurfaceVertexCriteriaAssignment object*
- *ThermalConductance object*
- *WearProperty object*
- *WearSurfacePropertyAssignment object*
- *XFEMCrackGrowth object*
- *XFEMCrackGrowthState object*

Interaction object

The Interaction object is the abstract base type for other Interaction objects. The Interaction object has no explicit constructor. Each of the Interaction objects has the following methods:

- `deactivate`
- `move`
- `reset`
- `resume`
- `suppress`
- `delete`

The methods are described below.

Access

```
import interaction
mdb.models[name].interactions[name]
```

`deactivate(...)`

This method deactivates the interaction in the specified step and all its subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the interaction is deactivated.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`move(...)`

This method moves an interaction from one step to another.

Required arguments

fromStepName A String specifying the name of the step from which to move the interaction.

toStepName A String specifying the name of the step to which to move the interaction.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`reset(...)`

This method reactivates an interaction that was deactivated previously. The `reset` method is available during the step in which the interaction was deactivated originally.

Required arguments

stepName	A String specifying the name of the step in which the interaction is reactivated.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`resume()`

This method resumes an interaction that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses an interaction.

Arguments

None.

Return value
`None.`

Exceptions

None.

`delete(...)`

This method allows you to delete existing interactions.

Required arguments

indices A sequence of Ints specifying the index of each interaction to delete.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

Members

The Interaction object has the following member:

name A String specifying the repository key.

AcousticImpedance object

The AcousticImpedance object defines surface impedance information or nonreflecting boundaries for acoustic and coupled acoustic-structural analyses.

The AcousticImpedance object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

AcousticImpedance(...)

This method creates an AcousticImpedance object.

Path

```
mdb.models[name].AcousticImpedance
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the AcousticImpedance object is created.

surface
A *Region* object specifying the acoustic boundary surface.

Optional arguments

definition
A SymbolicConstant specifying the type of acoustic impedance to be defined. Possible values are TABULAR and NONREFLECTING. The default value is TABULAR.

interactionProperty
A String specifying the *AcousticImpedanceProp* object associated with this interaction.

nonreflectingType
A SymbolicConstant specifying the type of nonreflecting geometry to be defined. Possible values are PLANE, IMPROVED, CIRCULAR, SPHERICAL, ELLIPTICAL, and PROLATE. The default value is PLANE.

This argument is valid only when *definition*=NONREFLECTING.

radius
A Float specifying the radius of the circle or sphere defining the boundary surface. The default value is 1.0.

This argument is valid only when *definition*=NONREFLECTING, and *nonreflectingType*=CIRCULAR or SPHERICAL.

semimajorAxis
A Float specifying the semimajor axis length of the ellipse or prolate spheroid defining the boundary surface. The default value is 1.0.

This argument is valid only when *definition*=NONREFLECTING, and *nonreflectingType*=ELLIPTICAL or PROLATE.

eccentricity

A Float specifying the eccentricity of the ellipse or prolate spheroid defining the boundary surface. The default value is 0.0.

This argument is valid only when *definition*=NONREFLECTING, and *nonreflectingType*=ELLIPTICAL or PROLATE.

centerCoordinates

A sequence of three Floats specifying the X, Y, and Z coordinates of the center of the ellipse or prolate spheroid defining the boundary surface. The default value is (0, 0, 0).

This argument is valid only when *definition*=NONREFLECTING, and *nonreflectingType*=ELLIPTICAL or PROLATE.

directionCosine

A sequence of three Floats specifying the X, Y, and Z components of the direction cosine of the major axis of the ellipse or prolate spheroid defining the boundary surface. The default value is (0, 0, 1).

This argument is valid only when *definition*=NONREFLECTING, and *nonreflectingType*=ELLIPTICAL or PROLATE.

Return value

An AcousticImpedance object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing AcousticImpedance object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticImpedance* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing AcousticImpedance object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

interactionProperty A String specifying the *AcousticImpedanceProp* object associated with this interaction.

Return value

`None.`

Exceptions

None.

Members

The AcousticImpedance object has members with the same names and descriptions as the arguments to the *AcousticImpedance* method except the optional arguments to the *setValuesInStep* method.

Corresponding analysis keywords

- **SIMPEDANCE*

AcousticImpedanceProp object

The AcousticImpedanceProp object is an interaction property that defines the properties referred to by an *AcousticImpedance* object.

The AcousticImpedanceProp object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

AcousticImpedanceProp(...)

This method creates an AcousticImpedanceProp object.

Path

```
mdb.models[name].AcousticImpedanceProp
```

Required arguments

name
A String specifying the interaction property repository key.

tableType
A SymbolicConstant specifying the type of tabular data to be defined. Possible values are IMPEDANCE and ADMITTANCE.

table
A sequence of sequences of Floats specifying acoustic impedance properties.

If *tableType*=IMPEDANCE, each sequence of the table data specifies:

- The real part of the complex impedance.
- The imaginary part of the complex impedance.
- Frequency, if the data depend on frequency.

If *tableType*=ADMITTANCE, each sequence of the table data specifies:

- The real part of the complex admittance.
- The imaginary part of the complex admittance.
- Frequency, if the data depend on frequency.

Optional arguments

frequencyDependency
A Boolean specifying whether the *table* data depend on frequency. The default value is OFF.

Return value

An AcousticImpedanceProp object.

Exceptions

None.

setValues(...)

This method modifies the AcousticImpedanceProp object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticImpedanceProp* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The AcousticImpedanceProp object has members with the same names and descriptions as the arguments to the *AcousticImpedanceProp* method.

Corresponding analysis keywords

- **IMPEDANCE PROPERTY*

AcousticImpedanceState object

The AcousticImpedanceState object stores the propagating data of an *AcousticImpedance* object in a step. One instance of this object is created internally by the *AcousticImpedance* object for each step. The instance is also deleted internally by the *AcousticImpedance* object.

The AcousticImpedanceState object has no constructor or methods.

The AcousticImpedanceState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The AcousticImpedanceState object has the following members:

interactionPropertyState

A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty

A String specifying the name of the *AcousticImpedanceProp* object associated with this interaction.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **SIMPEDANCE*

ActuatorSensor object

The ActuatorSensor object defines a single point actuator where the actuation is determined by a user subroutine (*UEL*). The subroutine senses the data at the same point as the actuator.

The ActuatorSensor object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ActuatorSensor(...)

This method creates an ActuatorSensor object.

Path

```
mdb.models[name].ActuatorSensor
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the actuator/sensor interaction is created. *createStepName* must be set to 'Initial'.
point	A *Region* object specifying the point at which the constraint is applied.
interactionProperty	A String specifying the *ActuatorSensorProp* object associated with this interaction.
noCoordComponents	An Int specifying the number of coordinate components supplied to the user subroutine (*UEL*).
unsymm	A Boolean specifying whether the element matrices are symmetric (ON) or unsymmetric (OFF). The default value is OFF.
noSolutionDepVar	An Int specifying the number of solution-dependent variables. The default value is 0.
userSubUel	A String specifying the name of the user subroutine (*UEL*) that defines the user element.
dof	A String specifying the degrees of freedom, separated by commas.
solutionDepVars	A sequence of Floats specifying the initial values of the solution-dependent variables.

Optional arguments

None.

Return value

An ActuatorSensor object.

Exceptions

None.

`setValues(...)`

This method modifies the ActuatorSensor object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ActuatorSensor* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The ActuatorSensor object has members with the same names and descriptions as the arguments to the *ActuatorSensor* method.

Corresponding analysis keywords

- **ELEMENT*
- **USER ELEMENT*
- **INITIAL CONDITIONS*, TYPE=SOLUTION

ActuatorSensorProp object

The ActuatorSensorProp object is an interaction property that defines the properties referred to by an *ActuatorSensor* object.

The ActuatorSensorProp object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

ActuatorSensorProp(...)

This method creates an ActuatorSensorProp object.

Path

```
mdb.models[name].ActuatorSensorProp
```

Required arguments

name A String specifying the interaction property repository key.

Optional arguments

realProperties A sequence of Floats specifying the `PROPS` array used by user subroutine *UEL*. The default value is an empty sequence.

integerProperties A sequence of Ints specifying the `JPROPS` array used by user subroutine *UEL*. The default value is an empty sequence.

Return value

An ActuatorSensorProp object.

Exceptions

None.

setValues(...)

This method modifies the ActuatorSensorProp object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ActuatorSensorProp* method, except for the *name* argument.

Return value
`None.`

Exceptions
None.

Members
The ActuatorSensorProp object has members with the same names and descriptions as the arguments to the *ActuatorSensorProp* method.

Corresponding analysis keywords
- **UEL PROPERTY*

ActuatorSensorState object

The ActuatorSensorState object stores the propagating data of an actuator sensor in a step. One instance of this object is created internally by the *ActuatorSensor* object for each step. The instance is also deleted internally by the *ActuatorSensor* object.

The ActuatorSensorState object has no constructor, methods, or members.

The ActuatorSensorState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The ActuatorSensorState object has the following member:

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

CavityRadiation object

The CavityRadiation object defines cavities for thermal radiation heat transfer and controls the calculation of viewfactors.

The CavityRadiation object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

CavityRadiation(...)

This method creates a CavityRadiation object.

Path

```
mdb.models[name].CavityRadiation
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the cavity radiation interaction should be created.

surfaces
A *RegionArray* object specifying the surfaces for which radiation viewfactor control is being specified.

Optional arguments

surfaceEmissivities
A sequence of Strings specifying the names of the Cavity Radiation properties containing the surface emissivity data. One name per specified surface. The emissivity data is ignored when *surfaceReflection*=OFF.

ambientTemp
`None` or a Float specifying the reference ambient temperature value, θ^0. Specifying a value indicates an open cavity. The default value is `None`.

blocking
A SymbolicConstant specifying the blocking checks to be performed in the viewfactor calculations. Possible values are BLOCKING_ALL, NO_BLOCKING, and PARTIAL_BLOCKING. The default value is BLOCKING_ALL.

blockingSurfaces
A *RegionArray* object specifying the surfaces that provide blocking inside the cavity. This argument applies only when *blocking*=PARTIAL_BLOCKING.

rangeOfView
`None` or a Float specifying the maximum distance between surface facets at which viewfactors are calculated. More distant facets are deemed too far apart to exchange significant amounts of heat through radiation effects, and the viewfactors between these facets are assumed to be zero. If *rangeOfView*=`None`, there is no upper limit. The default value is `None`.

surfaceReflection
A Boolean specifying whether heat reflections are to be included in the cavity radiation calculations. The default value is ON.

viewfactorAccurTol
A Float specifying the acceptable tolerance for the viewfactor calculations. The default value is 0.05.

minInfinitesimalRatio
A Float specifying the facet area ratio above which the infinitesimal-to-finite area approximation is used for viewfactor calculations. The default value is 64.0.

numPointsPerEdge
An Int specifying the number of Gauss integration points to be used along each edge when the numerical integration of contour integrals is used for viewfactor calculations. One to five integration points are allowed. The default value is 3.

minLumpedAreaDS
A Float specifying the nondimensional distance-square value above which the lumped area approximation is used for viewfactor calculations. The default value is 5.0.

cyclicSymmetry
A Boolean specifying whether cyclic symmetry will be applied. This argument cannot be specified for axisymmetric models. The default value is OFF.

cyclicImages
An Int specifying the number of cyclically similar images that compose the cavity formed as a result of this symmetry. This argument applies only when *cyclicSymmetry*=ON. The default value is 2.

cyclicRotPt
A *ModelDot* object specifying the rotation axis point. This argument applies only when *cyclicSymmetry*=ON.

cyclicRotEndPt
A *ModelDot* object specifying the rotation axis end point. This argument applies only for three-dimensional models, and only when *cyclicSymmetry*=ON.

cyclicSymPt
A *ModelDot* object specifying the symmetry axis end point. This argument applies only when *cyclicSymmetry*=ON.

periodicSymmetries
An Int specifying the number of periodic symmetries that will be applied. The default value is 0.

periodicImages_1
An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the first periodic symmetry. The result of this symmetry is a cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_1*. This argument applies only when *periodicSymmetries* is greater than zero. The default value is 2.

periodicImages_2
An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the second periodic symmetry. The result of this symmetry is a cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_2*. This argument applies only when *periodicSymmetries* is greater than one. The default value is 2.

periodicImages_3
An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the third periodic symmetry. The result of this symmetry is a

cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_3*. This argument applies only when *periodicSymmetries* = 3. The default value is 2.

periodicSymAxis_1

A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object indicating the first line of symmetry in two-dimensional models. This argument applies only for 2D models, and when *periodicSymmetries* is greater than zero.

periodicSymAxis_2

A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object indicating the second line of symmetry in two-dimensional models. This argument applies only for two-dimensional models, and when *periodicSymmetries* = 2.

periodicSymPlane_1

A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the first plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *periodicSymmetries* is greater than zero.

periodicSymPlane_2

A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the second plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *periodicSymmetries* is greater than one.

periodicSymPlane_3

A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the third plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *periodicSymmetries* = 3.

periodicDistance_1

A sequence of sequences of Floats specifying the two points of the vector that describes the periodic distance for the first periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* is greater than zero. The default value is an empty sequence.

periodicDistance_2

A sequence of sequences of Floats specifying the two points of the vector that describes the periodic distance for the second periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* is greater than one. The default value is an empty sequence.

periodicDistance_3

A sequence of sequences of Floats specifying the two points of the vector that describes the periodic distance for the third periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* = 3. The default value is an empty sequence.

periodicSymZ

`None` or a Float specifying the Z value indicating the symmetry reference line in axisymmetric models. This argument applies only for axisymmetric models, and when *periodicSymmetries* = 1. The default value is `None`.

periodicDistZ

`None` or a Float specifying the Z value indicating the periodic distance in axisymmetric models. This argument applies only for axisymmetric models, and when *periodicSymmetries* = 1. The default value is `None`.

reflectionSymmetries

An Int specifying the number of reflection symmetries will be applied. The default value is 0.

reflectionSymAxis_1
A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object indicating the first line of symmetry in two-dimensional models. This argument applies only for two-dimensional models, and when *reflectionSymmetries* is greater than zero.

reflectionSymAxis_2
A straight Edge, a *Datum* object representing a datum axis, or an ElementEdge object indicating the second line of symmetry in two-dimensional models. This argument applies only for two-dimensional models, and when *reflectionSymmetries* = 2.

reflectionSymPlane_1
A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the first plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *reflectionSymmetries* is greater than zero.

reflectionSymPlane_2
A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the second plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *reflectionSymmetries* is greater than one.

reflectionSymPlane_3
A planar Face, an ElementFace, or a *Datum* object representing a datum plane; indicating the third plane of symmetry in three-dimensional models. This argument applies only for three-dimensional models, and when *reflectionSymmetries* = 3.

reflectionSymZ
`None` or a Float specifying the Z value indicating the symmetry reference line in axisymmetric models. This argument applies only for axisymmetric models, and when *reflectionSymmetries* = 1. The default value is `None`.

Return value

A CavityRadiation object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing CavityRadiation object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CavityRadiation* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing CavityRadiation object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the interaction is modified.

Optional arguments

blocking
A SymbolicConstant specifying the blocking checks to be performed in the viewfactor calculations. Possible values are BLOCKING_ALL, NO_BLOCKING, and PARTIAL_BLOCKING. The default value is BLOCKING_ALL.

blockingSurfaces
A *RegionArray* object specifying the surfaces that provide blocking inside the cavity. This argument applies only when *blocking*=PARTIAL_BLOCKING.

rangeOfView
`None` or a Float specifying the maximum distance between surface facets at which viewfactors are calculated. More distant facets are deemed too far apart to exchange significant amounts of heat through radiation effects, and the viewfactors between these facets are assumed to be zero. If *rangeOfView*=`None`, there is no upper limit. The default value is `None`.

surfaceReflection
A Boolean specifying whether heat reflections are to be included in the cavity radiation calculations. The default value is ON.

viewfactorAccurTol
A Float specifying the acceptable tolerance for the viewfactor calculations. The default value is 0.05.

Return value

`None.`

Exceptions

None.

Members

The CavityRadiation object can have the following members:

name
A String specifying the repository key.

ambientTemp
`None` or a Float specifying the reference ambient temperature value, θ^0. Specifying a value indicates an open cavity. The default value is `None`.

minInfinitesimalRatio
A Float specifying the facet area ratio above which the infinitesimal-to-finite area approximation is used for viewfactor calculations. The default value is 64.0.

numPointsPerEdge

An Int specifying the number of Gauss integration points to be used along each edge when the numerical integration of contour integrals is used for viewfactor calculations. One to five integration points are allowed. The default value is 3.

minLumpedAreaDS

A Float specifying the nondimensional distance-square value above which the lumped area approximation is used for viewfactor calculations. The default value is 5.0.

cyclicSymmetry

A Boolean specifying whether cyclic symmetry will be applied. This argument cannot be specified for axisymmetric models. The default value is OFF.

cyclicImages

An Int specifying the number of cyclically similar images that compose the cavity formed as a result of this symmetry. This argument applies only when *cyclicSymmetry*=ON. The default value is 2.

periodicSymmetries

An Int specifying the number of periodic symmetries that will be applied. The default value is 0.

periodicImages_1

An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the first periodic symmetry. The result of this symmetry is a cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_1*. This argument applies only when *periodicSymmetries* is greater than zero. The default value is 2.

periodicImages_2

An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the second periodic symmetry. The result of this symmetry is a cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_2*. This argument applies only when *periodicSymmetries* is greater than one. The default value is 2.

periodicImages_3

An Int specifying the number of repetitions used in the numerical calculation of the cavity viewfactors resulting from the third periodic symmetry. The result of this symmetry is a cavity composed of the cavity surface defined in the model plus twice the value of *periodicImages_3*. This argument applies only when *periodicSymmetries* = 3. The default value is 2.

periodicSymZ

`None` or a Float specifying the Z value indicating the symmetry reference line in axisymmetric models. This argument applies only for axisymmetric models, and when *periodicSymmetries* = 1. The default value is `None`.

periodicDistZ

`None` or a Float specifying the Z value indicating the periodic distance in axisymmetric models. This argument applies only for axisymmetric models, and when *periodicSymmetries* = 1. The default value is `None`.

reflectionSymmetries

An Int specifying the number of reflection symmetries will be applied. The default value is 0.

reflectionSymZ

`None` or a Float specifying the Z value indicating the symmetry reference line in axisymmetric models. This argument applies only for axisymmetric models, and when *reflectionSymmetries* = 1. The default value is `None`.

createStepName
A String specifying the name of the step in which the cavity radiation interaction should be created.

surfaces
A *RegionArray* object specifying the surfaces for which radiation viewfactor control is being specified.

surfaceEmissivities
A tuple of Strings specifying the names of the Cavity Radiation properties containing the surface emissivity data. One name per specified surface. The emissivity data is ignored when *surfaceReflection*=OFF.

cyclicRotPt
A *ModelDot* object specifying the rotation axis point. This argument applies only when *cyclicSymmetry*=ON.

cyclicRotEndPt
A *ModelDot* object specifying the rotation axis end point. This argument applies only for three-dimensional models, and only when *cyclicSymmetry*=ON.

cyclicSymPt
A *ModelDot* object specifying the symmetry axis end point. This argument applies only when *cyclicSymmetry*=ON.

periodicDistance_1
A tuple of tuples of Floats specifying the two points of the vector that describes the periodic distance for the first periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* is greater than zero. The default value is an empty sequence.

periodicDistance_2
A tuple of tuples of Floats specifying the two points of the vector that describes the periodic distance for the second periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* is greater than one. The default value is an empty sequence.

periodicDistance_3
A tuple of tuples of Floats specifying the two points of the vector that describes the periodic distance for the third periodic symmetry. Each point is defined by a tuple of three coordinates indicating its position. This argument applies only when *periodicSymmetries* = 3. The default value is an empty sequence.

Corresponding analysis keywords

- **CAVITY DEFINITION*, SET PROPERTY
- **CYCLIC*
- **EMISSIVITY*
- **PERIODIC*
- **RADIATION SYMMETRY*
- **RADIATION VIEWFACTOR*
- **REFLECTION*

CavityRadiationProp object

The CavityRadiationProp object is an interaction property that defines emissivity as a function of temperature and field variables.

The CavityRadiationProp object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

CavityRadiationProp(...)

This method creates a CavityRadiationProp object.

Path

```
mdb.models[name].CavityRadiationProp
```

Required arguments

name A String specifying the interaction property repository key.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

property A sequence of sequences of Floats specifying the following:

- The emissivity, ϵ.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CavityRadiationProp object.

Exceptions

None.

setValues(...)

This method modifies the CavityRadiationProp object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CavityRadiationProp* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The CavityRadiationProp object has members with the same names and descriptions as the arguments to the *CavityRadiationProp* method.

Corresponding analysis keywords

- **EMISSIVITY*

CavityRadiationState object

The CavityRadiationState object stores the propagating data for a *CavityRadiation* object. One instance of this object is created internally by the *CavityRadiation* object for each step. The instance is also deleted internally by the *CavityRadiation* object.

The CavityRadiationState object has no constructor or methods.

The CavityRadiationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The CavityRadiationState object has the following members:

blocking

A SymbolicConstant specifying the blocking checks to be performed in the viewfactor calculations. Possible values are BLOCKING_ALL, NO_BLOCKING, and PARTIAL_BLOCKING.

blockingState

A SymbolicConstant specifying the propagation state of the blocking member. Possible values are UNSET, SET, UNCHANGED, and FREED.

blockingSurfacesState

A SymbolicConstant specifying the propagation state of the *blockingSurfaces* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

rangeOfView

A Float specifying the distance beyond which factors need not be calculated because surfaces are judged to be too far apart to “see” each other (due to blocking by other surfaces).

rangeOfViewState

A SymbolicConstant specifying the propagation state of the *rangeOfView* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

surfaceReflection

A Boolean specifying whether reflection must be included in the cavity radiation calculations. The default value is ON.

surfaceReflectionState

A SymbolicConstant specifying the propagation state of the *surfaceReflection* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

viewfactorAccuracyTol

A Float specifying the acceptable tolerance for the viewfactor calculations.

viewfactorAccuracyTolState

A SymbolicConstant specifying the propagation state of the *viewfactorAccuracyTol* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

blockingSurfaces

A tuple of Strings specifying the surfaces that provide blocking inside the cavity.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **RADIATION VIEWFACTOR*

CohesiveBehavior object

The CohesiveBehavior object specifies cohesive behavior for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].cohesiveBehavior
```

CohesiveBehavior(...)

This method creates a CohesiveBehavior object.

Path

```
mdb.models[name].interactionProperties[name].CohesiveBehavior
```

Required arguments

None.

Optional arguments

repeatedContacts A Boolean specifying whether to enforce cohesive behavior for recurrent contacts at nodes on the secondary surface subsequent to ultimate failure. The default value is OFF.

eligibility A SymbolicConstant specifying the eligible secondary nodes. Possible values are ALL_NODES, INITIAL_NODES, and SPECIFIED. The default value is ALL_NODES.

defaultPenalties A Boolean specifying whether to use the default contact penalties. The default value is ON.

coupling A SymbolicConstant specifying whether the traction-separation coefficients are coupled or uncoupled. This argument is valid only for *defaultPenalties*=OFF. Possible values are UNCOUPLED and COUPLED. The default value is UNCOUPLED.

temperatureDependency A Boolean specifying whether the coefficient data depend on temperature. This argument is valid only for *defaultPenalties*=OFF. The default value is OFF.

dependencies An Int specifying the number of field variables. This argument is valid only for *defaultPenalties*=OFF. The default value is 0.

table A sequence of sequences of Floats specifying the traction-separation coefficients. The items in the table data are described below. This argument is valid only for *defaultPenalties*=OFF.

Table data

If *coupling*=UNCOUPLED, the table data specify the following:

- Stiffness coefficient in the normal direction, K_{nn}.

- Stiffness coefficient in the first shear direction, K_{ss}.
- Stiffness coefficient in the second shear direction, K_{tt}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *coupling*=COUPLED, the table data specify the following:

- Stiffness coefficient in the normal direction, K_{nn}.
- Stiffness coefficient in the first shear direction, K_{ss}.
- Stiffness coefficient in the second shear direction, K_{tt}.
- Coupled stiffness coefficient, K_{ns}.
- Coupled stiffness coefficient, K_{nt}.
- Coupled stiffness coefficient, K_{st}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CohesiveBehavior object.

Exceptions

None.

`setValues(...)`

This method modifies the CohesiveBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CohesiveBehavior* method.

Return value

`None.`

Exceptions

None.

Members

The CohesiveBehavior object has members with the same names and descriptions as the arguments to the *CohesiveBehavior* method.

Corresponding analysis keywords

- **COHESIVE BEHAVIOR*

ConcentratedFilmCondition object

The ConcentratedFilmCondition object defines concentrated film coefficients and associated sink temperatures.

The ConcentratedFilmCondition object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ConcentratedFilmCondition(...)

This method creates a ConcentratedFilmCondition object.

Path

```
mdb.models[name].ConcentratedFilmCondition
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the ConcentratedFilmCondition object is created.

region
A *Region* object specifying the region to which the concentrated film condition interaction is applied. The interaction is applied to each node in the region.

definition
A SymbolicConstant specifying how the concentrated film condition is defined. Possible values are EMBEDDED_COEFF, PROPERTY_REF, USER_SUB, and FIELD.

Optional arguments

nodalArea
A Float specifying the area associated with the node where the concentrated film condition is applied. The default value is 1.0.

explicitRegionType
A SymbolicConstant specifying how the concentrated film condition is applied to the boundary of an adaptive mesh domain. Possible values are LAGRANGIAN, SLIDING, and EULERIAN. The default value is LAGRANGIAN.

This argument applies only during an Abaqus/Explicit analysis.

interactionProperty
A String specifying the name of the *FilmConditionProp* object associated with this interaction. The *interactionProperty* argument applies only when *definition*=PROPERTY_REF. The default value is an empty string.

field
A String specifying the name of the *AnalyticalField* object associated with this interaction. The *field* argument applies only when *definition*=FIELD. The default value is an empty string.

sinkTemperature

A Float specifying the reference sink temperature, θ^0. The default value is 0.0.

sinkAmplitude

A String specifying the name of the *Amplitude* object that gives the variation of the sink temperature, θ^0, with time. The default value is an empty string.



Note:

Use `None` in an Abaqus/Standard analysis to specify that the reference sink temperature is applied immediately at the beginning of the step or linearly over the step. Use `None` in an Abaqus/Explicit analysis to specify that the reference sink temperature is applied throughout the step.

filmCoeff

A Float specifying the reference film coefficient value, h. The *filmCoeff* argument applies when *definition*=EMBEDDED_COEFF, *definition*=USER_SUB, or *definition*=FIELD. The default value is 0.0.

filmCoeffAmplitude

A String specifying the name of the *Amplitude* object that gives the variation of the film coefficient, h, with time. The default value is an empty string.



Note:

Use `None` in an Abaqus/Standard analysis to specify that the reference film coefficient is applied immediately at the beginning of the step or linearly over the step. Use `None` in an Abaqus/Explicit analysis to specify that the reference film coefficient is applied throughout the step.

sinkFieldName

A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with the sink temperature. The *sinkFieldName* argument applies only when *sinkDistributionType*=ANALYTICAL_FIELD or *sinkDistributionType*=DISCRETE_FIELD. The default value is an empty string.

sinkDistributionType

A SymbolicConstant specifying how the sink temperature is distributed. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

Return value

A ConcentratedFilmCondition object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConcentratedFilmCondition object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcentratedFilmCondition* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing ConcentratedFilmCondition object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

The optional arguments to `setValuesInStep` are the same as the optional arguments to the `ConcentratedFilmConditionState` method.

Return value

`None.`

Exceptions

None.

Members

The ConcentratedFilmCondition object has members with the same names and descriptions as the arguments to the *ConcentratedFilmCondition* method except the optional arguments to the *setValuesInStep* method.

ConcentratedFilmConditionState object

The ConcentratedFilmConditionState object stores the propagating data for a *ConcentratedFilmCondition* object. One instance of this object is created internally by the *ConcentratedFilmCondition* object for each step. The instance is also deleted internally by the *ConcentratedFilmCondition* object.

The ConcentratedFilmConditionState object has no constructor or methods.

The ConcentratedFilmConditionState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The ConcentratedFilmConditionState object has the following members:

interactionPropertyState
A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

sinkTemperature
A Float specifying the sink temperature.

sinkTemperatureState
A SymbolicConstant specifying the propagation state of the *sinkTemperature* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

sinkAmplitudeState
A SymbolicConstant specifying the propagation state of the *sinkAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

filmCoeff
A Float specifying the film coefficient.

filmCoeffState
A SymbolicConstant specifying the propagation state of the *filmCoeff* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

filmCoeffAmplitudeState
A SymbolicConstant specifying the propagation state of the *filmCoeffAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

nodalArea
A Float specifying the area associated with the node where the concentrated film condition is applied.

nodalAreaState
A SymbolicConstant specifying the propagation state of the *nodalArea* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty
A String specifying the *FilmConditionProp* object associated with this interaction.

sinkAmplitude
A String specifying the name of the Amplitude object that gives the variation of the sink temperature.

filmCoeffAmplitude

A String specifying the name of the Amplitude object that gives the variation of the film coefficient.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CFILM*

ConcentratedRadiationToAmbient object

The ConcentratedRadiationToAmbient object defines radiant heat transfer between a point and its nonreflecting environment.

The ConcentratedRadiationToAmbient object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ConcentratedRadiationToAmbient(...)

This method creates a ConcentratedRadiationToAmbient object.

Path

```
mdb.models[name].ConcentratedRadiationToAmbient
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the ConcentratedRadiationToAmbient object is created.

region
A *Region* object specifying the region to which the concentrated radiation interaction is applied. The interaction is applied to each node in the region.

ambientTemperature
A Float specifying the reference ambient temperature, θ^0.

ambientTemperatureAmp
A String specifying the name of the *Amplitude* object that gives the variation of the ambient temperature with time.



Note:

Use `None` in an Abaqus/Standard analysis to specify that the reference ambient temperature is applied immediately at the beginning of the step or linearly over the step. Use `None` in an Abaqus/Explicit analysis to specify that the reference ambient temperature is applied throughout the step.

emissivity
A Float specifying the emissivity, ϵ.

Optional arguments

nodalArea
A Float specifying the area associated with the node where the concentrated radiation interaction is applied. The default value is 1.0.

explicitRegionType A SymbolicConstant specifying how the concentrated radiation is applied to the boundary of an adaptive mesh domain. Possible values are LAGRANGIAN, SLIDING, and EULERIAN. The default value is LAGRANGIAN.



Note:

explicitRegionType applies only during an Abaqus/Explicit analysis.

field A String specifying the name of the *AnalyticalField* object associated with this interaction. The *field* argument applies only when *distributionType*=ANALYTICAL_FIELD. The default value is an empty string.

distributionType A SymbolicConstant specifying how the radiation is defined. Possible values are UNIFORM and ANALYTICAL_FIELD. The default value is UNIFORM.

Return value

A ConcentratedRadiationToAmbient object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConcentratedRadiationToAmbient object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcentratedRadiationToAmbient* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing ConcentratedRadiationToAmbient object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

The optional arguments to `setValuesInStep` are the same as the arguments to the *ConcentratedRadiationToAmbient* method, except for the *name*, *createStepName*, *region*, and *explicitRegionType* arguments.

Return value

`None.`

Exceptions

None.

Members

The ConcentratedRadiationToAmbient object has members with the same names and descriptions as the arguments to the *ConcentratedRadiationToAmbient* method.

ConcentratedRadiationToAmbientState object

The ConcentratedRadiationToAmbientState object stores the propagating data for a *ConcentratedRadiationToAmbient* object. One instance of this object is created internally by the *ConcentratedRadiationToAmbient* object for each step. The instance is also deleted internally by the *ConcentratedRadiationToAmbient* object.

The ConcentratedRadiationToAmbientState object has no constructor or methods.

The ConcentratedRadiationToAmbientState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The ConcentratedRadiationToAmbientState object has the following members:

ambientTemperature

A Float specifying the ambient temperature.

ambientTemperatureState

A SymbolicConstant specifying the propagation state of the *ambientTemperature* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

ambientTemperatureAmpState

A SymbolicConstant specifying the propagation state of the *ambientTemperatureAmp* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

emissivity

A Float specifying the emissivity.

emissivityState

A SymbolicConstant specifying the propagation state of the *emissivity* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

nodalArea

A Float specifying the area associated with the node where the concentrated radiation is applied.

nodalAreaState

A SymbolicConstant specifying the propagation state of the *nodalArea* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

ambientTemperatureAmp

A String specifying the name of the Amplitude object that gives the variation of the ambient temperature with time.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE

- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CRADIATE*

ContactControl object

The ContactControl object is the abstract base type for other ContactControl objects. The ContactControl object has no explicit constructor, members, or methods.

Access

```
import interaction
mdb.models[name].contactControls[name]
```

Members

The ContactControl object has no members.

ContactDamage object

The ContactDamage object specifies damage options for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].damage
```

Damage(...)

This method creates a ContactDamage object.

Path

```
mdb.models[name].interactionProperties[name].Damage
```

Required arguments

initTable
A sequence of sequences of Floats specifying the values defining the damage initiation. The items in the table data are described below.

Optional arguments

criterion
A SymbolicConstant specifying the type of data used to define the initiation of damage. Possible values are MAX_STRESS, MAX_SEPARATION, QUAD_TRACTION, and QUAD_SEPARATION. The default value is MAX_STRESS.

initTempDep
A Boolean specifying whether the initiation data depend on temperature. The default value is OFF.

initDependencies
An Int specifying the number of initiation data field variables. The default value is 0.

useEvolution
A Boolean specifying whether evolution data will be defined. The default value is OFF.

evolutionType
A SymbolicConstant specifying the type of data used to define the evolution of damage. This argument is valid only when *useEvolution*=ON. Possible values are DISPLACEMENT and ENERGY. The default value is DISPLACEMENT.

softening
A SymbolicConstant specifying the type of data used to define the evolution softening response. This argument is valid only when *useEvolution*=ON. The TABULAR value can be used only when *evolutionType*=DISPLACEMENT. Possible values are LINEAR, EXPONENTIAL, and TABULAR. The default value is LINEAR.

useMixedMode
A Boolean specifying whether evolution data be defined using dependent behavior modes. This argument is valid only when *useEvolution*=ON. The default value is OFF.

mixedModeType
A SymbolicConstant specifying the mode mix fracture criterion. This argument is valid only when *useEvolution*=ON and when *useMixedMode*=ON. The POWER_LAW and BK values

can be used only when *evolutionType*=ENERGY. Possible values are TABULAR, POWER_LAW, and BK. The default value is TABULAR.

modeMixRatio

A SymbolicConstant specifying the mode mix ratio type. This argument is valid only when *useEvolution*=ON and when *useMixedMode*=ON. The TRACTION value can be used only when *mixedModeType*=TABULAR. Possible values are ENERGY and TRACTION. The default value is ENERGY.

exponent

`None` or a Float specifying the exponent in the power-law or BK criterion that defines the variation of fracture energy with mode mix. This argument is valid only when *useEvolution*=ON and when *mixedModeType*=POWER_LAW or BK. The default value is `None`.

evolTempDep

A Boolean specifying whether the evolution data depend on temperature. This argument is valid only when *useEvolution*=ON. The default value is OFF.

evolDependencies

An Int specifying the number of evolution data field variables. This argument is valid only when *useEvolution*=ON. The default value is 0.

evolTable

A sequence of sequences of Floats specifying the values defining the damage evolution. The items in the table data are described below. This argument is valid only when *useEvolution*=ON.

useStabilization

A Boolean specifying whether stabilization data will be defined. This argument is valid only when *useEvolution*=ON. The default value is OFF.

viscosityCoef

`None` or a Float specifying the viscosity coefficient. This argument is valid only when *useStabilization*=ON. The default value is `None`.

Table data

Table data for *initTable*:

If *criterion*=MAX_STRESS or QUAD_TRACTION, the table data specify the following:

- Maximum nominal stress in the normal-only mode.
- Maximum nominal stress in the first shear direction (for a mode that involves separation only in this direction).
- Maximum nominal stress in the second shear direction (for a mode that involves separation only in this direction).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *criterion*=MAX_SEPARATION or QUAD_SEPARATION, the table data specify the following:

- Separation at damage initiation in a normal-only mode.
- Separation at damage initiation in a shear-only mode that involves separation only along the first shear direction.
- Separation at damage initiation in a shear-only mode that involves separation only along the second shear direction.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Table data for *evolTable*:

If *evolutionType*=DISPLACEMENT, *softening*=LINEAR, and *useMixedMode*=OFF, the table data specify the following:

- Effective total or plastic displacement at failure, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=ENERGY, *softening*=LINEAR or EXPONENTIAL, and *useMixedMode*=OFF, the table data specify the following:

- Fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=DISPLACEMENT, *softening*=LINEAR, *useMixedMode*=ON, *mixedModeType*=TABULAR, and *modeMixRatio*=ENERGY or TRACTION, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=ENERGY, *softening*=LINEAR or EXPONENTIAL, *useMixedMode*=ON, *mixedModeType*=TABULAR, and *modeMixRatio*=ENERGY or TRACTION, the table data specify the following:

- Fracture energy.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=DISPLACEMENT, *softening*=EXPONENTIAL, and *useMixedMode*=OFF, the table data specify the following:

- Effective total or plastic displacement at failure, measured from the time of damage initiation.

- Exponential law parameter.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=DISPLACEMENT, *softening*=EXPONENTIAL, *useMixedMode*=ON, *mixedModeType*=TABULAR, and *modeMixRatio*=ENERGY or TRACTION, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.
- Exponential law parameter.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=DISPLACEMENT, *softening*=TABULAR, and *useMixedMode*=OFF, the table data specify the following:

- Damage variable.
- Effective total or plastic displacement at failure, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=DISPLACEMENT, *softening*=TABULAR, *useMixedMode*=ON, *mixedModeType*=TABULAR, and *modeMixRatio*=ENERGY or TRACTION, the table data specify the following:

- Damage variable.
- Effective total displacement, measured from the time of damage initiation.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=ENERGY, *softening*=LINEAR or EXPONENTIAL, *useMixedMode*=ON, *mixedModeType*=POWER_LAW or BK, and *modeMixRatio*=ENERGY, the table data specify the following:

- Normal mode fracture energy.
- Shear mode fracture energy for failure in the first shear direction.
- Shear mode fracture energy for failure in the second shear direction.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ContactDamage object.

Exceptions

None.

`setValues(...)`

This method modifies the ContactDamage object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContactDamage* method.

Return value

`None.`

Exceptions

None.

Members

The ContactDamage object has the following members:

criterion
A SymbolicConstant specifying the type of data used to define the initiation of damage. Possible values are MAX_STRESS, MAX_SEPARATION, QUAD_TRACTION, and QUAD_SEPARATION. The default value is MAX_STRESS.

initTempDep
A Boolean specifying whether the initiation data depend on temperature. The default value is OFF.

initDependencies
An Int specifying the number of initiation data field variables. The default value is 0.

useEvolution
A Boolean specifying whether evolution data will be defined. The default value is OFF.

evolutionType
A SymbolicConstant specifying the type of data used to define the evolution of damage. This argument is valid only when *useEvolution*=ON. Possible values are DISPLACEMENT and ENERGY. The default value is DISPLACEMENT.

softening
A SymbolicConstant specifying the type of data used to define the evolution softening response. This argument is valid only when *useEvolution*=ON. The TABULAR value can be used only when *evolutionType*=DISPLACEMENT. Possible values are LINEAR, EXPONENTIAL, and TABULAR. The default value is LINEAR.

useMixedMode
A Boolean specifying whether evolution data be defined using dependent behavior modes. This argument is valid only when *useEvolution*=ON. The default value is OFF.

mixedModeType
A SymbolicConstant specifying the mode mix fracture criterion. This argument is valid only when *useEvolution*=ON and when *useMixedMode*=ON. The POWER_LAW and BK values can be used only when *evolutionType*=ENERGY. Possible values are TABULAR, POWER_LAW, and BK. The default value is TABULAR.

modeMixRatio
A SymbolicConstant specifying the mode mix ratio type. This argument is valid only when *useEvolution*=ON and when *useMixedMode*=ON. The TRACTION value can be used only when *mixedModeType*=TABULAR. Possible values are ENERGY and TRACTION. The default value is ENERGY.

exponent
`None` or a Float specifying the exponent in the power-law or BK criterion that defines the variation of fracture energy with mode mix. This argument is valid only when *useEvolution*=ON and when *mixedModeType*=POWER_LAW or BK. The default value is `None`.

evolTempDep
A Boolean specifying whether the evolution data depend on temperature. This argument is valid only when *useEvolution*=ON. The default value is OFF.

evolDependencies
An Int specifying the number of evolution data field variables. This argument is valid only when *useEvolution*=ON. The default value is 0.

useStabilization
A Boolean specifying whether stabilization data will be defined. This argument is valid only when *useEvolution*=ON. The default value is OFF.

viscosityCoef
`None` or a Float specifying the viscosity coefficient. This argument is valid only when *useStabilization*=ON. The default value is `None`.

initTable
A tuple of tuples of Floats specifying the values defining the damage initiation. The items in the table data are described below.

evolTable
A tuple of tuples of Floats specifying the values defining the damage evolution. The items in the table data are described below. This argument is valid only when *useEvolution*=ON.

Corresponding analysis keywords

- **DAMAGE INITIATION*
- **DAMAGE EVOLUTION*
- **DAMAGE STABILIZATION*

ContactDamping object

The ContactDamping object specifies damping for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].damping
```

Damping(...)

This method creates a ContactDamping object.

Path

```
mdb.models[name].interactionProperties[name].Damping
```

Required arguments

None.

Optional arguments

definition
A SymbolicConstant specifying the method used to define the damping. Possible values are DAMPING_COEFFICIENT and CRITICAL_DAMPING_FRACTION. The default value is DAMPING_COEFFICIENT.

tangentFraction
The SymbolicConstant DEFAULT or a Float specifying the tangential damping coefficient divided by the normal damping coefficient. The default value is DEFAULT.

clearanceDependence
A SymbolicConstant specifying the variation of the damping coefficient or fraction with respect to clearance. Possible values are STEP, LINEAR, and BILINEAR. The default value is STEP.

If *definition*=CRITICAL_DAMPING_FRACTION, the only possible value is STEP.

table
A sequence of pairs of Floats specifying the damping properties. The items in the table data are described below.

Table data

If *definition*=DAMPING_COEFFICIENT and *clearanceDependence*=STEP, the table data specify the following:

- Damping coefficient.

If *definition*=DAMPING_COEFFICIENT and *clearanceDependence*=LINEAR or BILINEAR, the table data specify the following:

- Damping coefficient.
- Clearance.

Two pairs must be given for *clearanceDependence*=LINEAR and three pairs for *clearanceDependence*=BILINEAR. The first pair must have *clearance*=0.0, and the last pair must have *coefficient*=0.0.

If *definition*=CRITICAL_DAMPING_FRACTION, the table data specify the following:

- Critical damping fraction.

Return value

A ContactDamping object.

Exceptions

None.

setValues(...)

This method modifies the ContactDamping object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContactDamping* method.

Return value

`None.`

Exceptions

None.

Members

The ContactDamping object has the following members:

definition

A SymbolicConstant specifying the method used to define the damping. Possible values are DAMPING_COEFFICIENT and CRITICAL_DAMPING_FRACTION. The default value is DAMPING_COEFFICIENT.

tangentFraction

The SymbolicConstant DEFAULT or a Float specifying the tangential damping coefficient divided by the normal damping coefficient. The default value is DEFAULT.

clearanceDependence

A SymbolicConstant specifying the variation of the damping coefficient or fraction with respect to clearance. Possible values are STEP, LINEAR, and BILINEAR. The default value is STEP.

If *definition*=CRITICAL_DAMPING_FRACTION, the only possible value is STEP.

table

A tuple of pairs of Floats specifying the damping properties. The items in the table data are described below.

Corresponding analysis keywords

- **CONTACT DAMPING*

ContactExp object

The ContactExp object defines the contact domain and associated properties during contact in an Abaqus/Explicit analysis.

The ContactExp object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ContactExp(...)

This method creates a ContactExp object.

Path

```
mdb.models[name].ContactExp
```

Required arguments

name A String specifying the repository key.

createStepName A String specifying the name of the step in which this contact interaction is created.

Optional arguments

useAllstar A Boolean specifying whether the contacting surface pair consists of all exterior faces, shell edges, beam segments, analytical rigid surfaces, and, when applicable, Eulerian material surfaces.

globalSmoothing A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

includedPairs A *RegionPairs* object specifying the domain pairs included in contact.

excludedPairs A *RegionPairs* object specifying the domain pairs excluded from contact.

contactPropertyAssignments A *ContactPropertyAssignment* object specifying the contact property assignments in the contact domain.

surfaceThicknessAssignments A *SurfaceThicknessAssignment* object specifying the surface thickness assignments in the contact domain.

surfaceOffsetAssignments A *SurfaceOffsetAssignment* object specifying the surface offset fraction assignments in the contact domain.

surfaceFeatureAssignments A *SurfaceFeatureAssignment* object specifying the surface feature angle assignments in the contact domain.

smoothingAssignments A *SmoothingAssignment* object specifying the surface smoothing assignments in the contact domain.

surfaceCrushTriggerAssignments A *SurfaceCrushTriggerAssignment* object specifying the surface crush trigger assignments in the contact domain.

surfaceFrictionAssignments A *SurfaceFrictionAssignment* object specifying the surface friction assignments in the contact domain.

mainSecondaryAssignments A *MainSecondaryAssignment* object specifying the main-secondary assignments in the contact domain.

PolarityAssignments A *PolarityAssignments* object specifying the polarity assignments in the contact domain.

Return value

A ContactExp object.

Exceptions

None.

ContactExp(...)

This method creates a ContactExp object.

Path

```
mdb.models[name].ContactExp
```

Required arguments

name A String specifying the repository key.

createStepName A String specifying the name of the step in which this contact interaction is created.

Optional arguments

globalSmoothing A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

surfaceCrushTriggerAssignments A sequence of tuples specifying the surface crush trigger assignments. Each tuple contains four entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the feature angle is assigned.
- A SymbolicConstant specifying the trigger option to be used for the surface. Possible values of the SymbolicConstant are TRIGGER, NO_TRIGGER, or NO_CRUSH.

- A Float specifying the crush stress value to be used for the surface.
- A Float specifying the crush initiation angle value to be used for the surface.
- A Float specifying the crush continuation angle value to be used for the surface.

surfaceFrictionAssignments

A sequence of tuples specifying the surface friction assignments. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the friction coefficient is assigned.
- A Float specifying the overriding friction coefficient to be used in the contact definition.

useAllstar

A Boolean specifying whether the contacting surface pair consists of all exterior faces, shell edges, beam segments, analytical rigid surfaces, and, when applicable, Eulerian material surfaces.

includedPairs

A sequence of pairs of *Region* objects or SymbolicConstants that specifies the surface pairs in contact. Possible values of the SymbolicConstants are ALLSTAR and SELF. This argument is valid only when *useAllstar*=OFF.

excludedPairs

A sequence of pairs of *Region* objects or SymbolicConstants that specify the surface pairs excluded from contact. Possible values of the SymbolicConstants are ALLSTAR and SELF.

contactPropertyAssignments

A sequence of tuples specifying the properties assigned to each surface pair. Each tuple contains three entries:

- A *Region* object or the SymbolicConstant GLOBAL.
- A *Region* object or the SymbolicConstant SELF.
- A String specifying an *InteractionProperty* object associated with this pair of regions.

surfaceThicknessAssignments

A sequence of tuples specifying the surface thickness assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface thickness is assigned.
- A Float or a SymbolicConstant specifying the overriding thickness value to be used in the contact definition. Possible values of the SymbolicConstant are ORIGINAL or THINNING.
- A Float specifying a scale factor that multiplies the thickness value specified in the second entry.

surfaceOffsetAssignments

A sequence of tuples specifying the surface offset fraction assignments in the contact domain. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface offset fraction is assigned.

- A Float or a SymbolicConstant specifying the offset fraction value to be used in the contact definition. Possible values of the SymbolicConstant are ORIGINAL, SPOS, or SNEG.

surfaceFeatureAssignments

A sequence of tuples specifying the surface feature angle assignments in the contact domain. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface feature angle is assigned.
- A Float or a SymbolicConstant specifying the overriding feature angle value to be used in the contact definition. Possible values of the SymbolicConstant are PERIMETER, ALL, PICKED, or NONE.

smoothingAssignments

A sequence of tuples specifying the surface smoothing assignments in the contact domain. Each tuple contains two entries:

- A region object specifying the surface to which the smoothing option is assigned.
- A SymbolicConstant specifying the smoothing option to be used in the contact definition. Possible values of the SymbolicConstant are NONE, REVOLUTION, SPHERICAL, or TOROIDAL.

mainSecondaryAssignments

A sequence of tuples specifying pure main-secondary assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the first surface that defines the main-secondary assignment.
- A region object specifying the second surface in the main-secondary assignment definition.
- A SymbolicConstant specifying the status of the first surface. Possible values are MAIN and SECONDARY.

polarityAssignments

A sequence of tuples specifying polarity assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the first surface that defines the polarity assignment.
- A region object specifying the second surface in the polarity assignment definition.
- A SymbolicConstant specifying the polarity of the second surface. Possible values are SPOS, SNEG, and TWO_SIDED.

Return value

A ContactExp object.

Exceptions

None.

Members

The ContactExp object can have the following members:

name
A String specifying the repository key.

globalSmoothing
A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

includedPairs
A *RegionPairs* object specifying the domain pairs included in contact.

excludedPairs
A *RegionPairs* object specifying the domain pairs excluded from contact.

contactPropertyAssignments
A *ContactPropertyAssignment* object specifying the contact property assignments in the contact domain.

surfaceThicknessAssignments
A *SurfaceThicknessAssignment* object specifying the surface thickness assignments in the contact domain.

surfaceOffsetAssignments
A *SurfaceOffsetAssignment* object specifying the surface offset fraction assignments in the contact domain.

surfaceFeatureAssignments
A *SurfaceFeatureAssignment* object specifying the surface feature angle assignments in the contact domain.

smoothingAssignments
A *SmoothingAssignment* object specifying the surface smoothing assignments in the contact domain.

mainSecondaryAssignments
A *MainSecondaryAssignment* object specifying the main-secondary assignments in the contact domain.

polarityAssignments
A *PolarityAssignment* object specifying the polarity assignments in the contact domain.

assignBeamCrossSection
A Boolean specifying whether to assign the beam cross section. The default value is False.

beamCrossSection
A SymbolicConstant specifying whether the contact calculations should be based on a circumferential approximation or the actual cross section of the beam element. Possible values are CIRCUMSCRIBED_CIRCLE and EXACT.

Corresponding analysis keywords

- **CONTACT*

ContactInitialization object

The ContactInitialization object is the abstract base type for other ContactInitialization objects. The ContactInitialization object has no explicit constructor, members, or methods.

Access

```
import interaction
mdb.models[name].contactInitializations[name]
```

Members

The ContactInitialization object has no members.

ContactMassScalingExp object

The ContactMassScalingExp object defines contact mass scaling during an Abaqus/Explicit analysis.

The ContactMassScalingExp object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ContactMassScalingExp(...)

This method creates a ContactMassScalingExp object.

Path

```
mdb.models[name].ContactMassScalingExp
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the ContactMassScalingExp object is created.
location	A SymbolicConstant specifying the location. Possible values are ELEMENT MASS SCALING (default), NONE, ALL CONTACT SURFACES, and SPECIFIED SURFACES.
surfaces	A tuple of *Region* objects specifying the surfaces when *LOCATION*=SPECIFIED SURFACES.

Return value

A ContactMassScalingExp object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ContactMassScalingExp object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContactMassScalingExp* method, except for the *name* and *createStepName* arguments.

Return value
`None.`

Exceptions
None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing ContactMassScalingExp object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments
The optional arguments to `setValuesInStep` are the same as the arguments to the *ContactMassScalingExp* method, except for the *name* and *createStepName* arguments.

Return value
`None.`

Exceptions
None.

ContactProperty object

The ContactProperty object defines a contact interaction property.

The ContactProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

ContactProperty(...)

This method creates a ContactProperty object.

Path

```
mdb.models[name].ContactProperty
```

Required arguments

name A String specifying the interaction property repository key.

Optional arguments

None.

Return value

A ContactProperty object.

Exceptions

None.

Members

The ContactProperty object can have the following members:

tangentialBehavior A *ContactTangentialBehavior* object.

normalBehavior A *NormalBehavior* object.

damping A *ContactDamping* object.

damage A *ContactDamage* object.

fractureCriterion A *FractureCriterion* object.

cohesiveBehavior A *CohesiveBehavior* object.

thermalConductance	A *ThermalConductance* object.
heatGeneration	A *GapHeatGeneration* object.
radiation	A *Radiation* object.
geometricProperties	A *GeometricProperties* object.
electricalConductance	A *GapElectricalConductance* object.

Corresponding analysis keywords

- **SURFACE INTERACTION*

ContactPropertyAssignment object

The ContactPropertyAssignment object stores the contact property assignment definition for domain pairs in *ContactExp* and *ContactStd* objects. The ContactPropertyAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].contactPropertyAssignments
```

changeValuesInStep(...)

This method allows modification of contact property assignments to domain pairs already defined in a given step.

Required arguments

stepName A String specifying the name of the step in which the contact property assignments are to be modified.

index An Int specifying the position of the contact property assignment whose value is to be modified.

value A String specifying the value of the contact property to be assigned to the domain pair whose index is referenced.

Optional arguments

None.

Return value

None.

Exceptions

None.

appendInStep(...)

This method allows addition of contact property assignments to new domain pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which new contact property assignments are to be defined.

assignments A sequence of tuples specifying the properties assigned to each surface pair. Each tuple contains three entries:

- A region or a material object or the SymbolicConstant GLOBAL.

- A region or a material object or the SymbolicConstant SELF. When used with a *ContactExp* object, this parameter can also be a string that references an Eulerian material surface.
- A String specifying a *ContactProperty* object associated with this pair of regions.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing contact property assignments.

Required arguments

indices

A sequence of Ints specifying the index of each contact property assignment to delete. The *indices* and *surfPair* arguments are mutually exclusive.

surfPair

A sequence of tuples specifying the surface pair of each contact property assignment to delete. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL.
- A region or a material object or the SymbolicConstant SELF. When used with a *ContactExp* object, this parameter can also be a string that references an Eulerian material surface.

surfPair and *indices* arguments are mutually exclusive.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The ContactPropertyAssignment object has no members.

Corresponding analysis keywords

- **CONTACT PROPERTY ASSIGNMENT*

ContactStabilization object

The ContactStabilization object is the abstract base type for other ContactStabilization objects. The ContactStabilization object has no explicit constructor, members, or methods.

Access

```
import interaction
mdb.models[name].contactStabilizations[name]
```

Members

The ContactStabilization object has no members.

ContactStd object

The ContactStd object defines the contact domain and associated properties during contact in an Abaqus/Standard analysis.

The ContactStd object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ContactStd(...)

This method creates a ContactStd object.

Path

```
mdb.models[name].ContactStd
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which this contact interaction is created.

Optional arguments

useAllstar
A Boolean specifying whether the contacting surface pairs consist of all exterior faces in the model.

globalSmoothing
A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

includedPairs
A *RegionPairs* object specifying the domain pairs included in contact.

excludedPairs
A *RegionPairs* object specifying the domain pairs excluded from contact.

contactPropertyAssignments
A *ContactPropertyAssignment* object specifying the contact property assignments in the contact domain.

surfaceThicknessAssignments
A *SurfaceThicknessAssignment* object specifying the surface thickness assignments in the contact domain.

surfaceOffsetAssignments
A *SurfaceOffsetAssignment* object specifying the surface offset fraction assignments in the contact domain.

surfaceFeatureAssignments
A *SurfaceFeatureAssignment* object specifying the surface feature angle assignments in the contact domain.

surfaceBeamSmoothingAssignments A *SurfaceBeamSmoothingAssignment* object specifying the surface beam smoothing assignments in the contact domain.

surfaceVertexCriteriaAssignments A *SurfaceVertexCriteriaAssignment* object specifying the surface vertex criteria assignments in the contact domain.

mainSecondaryAssignments A *MainSecondaryAssignment* object specifying the main-secondary assignments in the contact domain.

initializationAssignments An *InitializationAssignment* object specifying the contact initialization assignments in the contact domain.

stabilizationAssignments A *StabilizationAssignment* object specifying the contact stabilization assignments in the contact domain.

smoothingAssignments A *SmoothingAssignment* object specifying the surface smoothing assignments in the contact domain.

slidingTransitionAssignments A *SlidingTransitionAssignment* object specifying the sliding transition assignments in the contact domain.

slidingFormulationAssignments A *SlidingFormulationAssignment* object specifying the sliding formulation assignments in the contact domain.

Return value

A ContactStd object.

Exceptions

None.

ContactStd(...)

This method creates a ContactStd object.

Path

```
mdb.models[name].ContactStd
```

Required arguments

name A String specifying the repository key.

createStepName A String specifying the name of the step in which this contact interaction is created.

Optional arguments

globalSmoothing

A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

surfaceBeamSmoothingAssignments

A sequence of tuples specifying the surface beam smoothing assignments. Each tuple contains two entries:

- A region object specifying the surface to which the smoothing is assigned.
- A Float specifying the surface smoothing value to be used for the surface.

surfaceVertexCriteriaAssignments

A sequence of tuples specifying the surface vertex criteria assignments. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the vertex criteria is assigned.
- A Float or a SymbolicConstant specifying the vertex criteria value to be used for the surface. Possible values of the SymbolicConstant are ALL_VERTICES or NO_VERTICES.

slidingFormulationAssignments

A sequence of tuples specifying the sliding formulation assignments. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the sliding formulation attribute is assigned.
- A SymbolicConstant specifying the overriding the smoothness value to be used for the first surface. Possible values of the SymbolicConstant are NONE and SMALL_SLIDING.

useAllstar

A Boolean specifying whether the contacting surface pairs consist of all exterior faces in the model.

includedPairs

A sequence of pairs of *Region* objects or SymbolicConstants that specifies the surface pairs in contact. Possible values of the SymbolicConstants are ALLSTAR and SELF. This argument is valid only when *useAllstar*=OFF.

excludedPairs

A sequence of pairs of *Region* objects or SymbolicConstants that specifies the surface pairs excluded from contact. Possible values of the SymbolicConstants are ALLSTAR and SELF.

contactPropertyAssignments

A sequence of tuples specifying the properties assigned to each surface pair. Each tuple contains three entries:

- A *Region* object or the SymbolicConstant GLOBAL.
- A *Region* object or the SymbolicConstant SELF.
- A String specifying an *InteractionProperty* object associated with this pair of regions.

surfaceFeatureAssignments

A sequence of tuples specifying the surface feature angle assignments in the contact domain. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface feature angle is assigned.
- A Float or a SymbolicConstant specifying the overriding feature angle value to be used in the contact definition. Possible values of the SymbolicConstant are PERIMETER or NONE.

surfaceThicknessAssignments

A sequence of tuples specifying the surface thickness assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface thickness is assigned.
- A Float or a SymbolicConstant specifying the overriding thickness value to be used in the contact definition. The SymbolicConstant ORIGINAL may be specified instead of a floating point value.
- A Float specifying a scale factor that multiplies the thickness value specified in the second entry.

surfaceOffsetAssignments

A sequence of tuples specifying the surface offset fraction assignments in the contact domain. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the surface offset fraction is assigned.
- A Float or a SymbolicConstant specifying the offset fraction value to be used in the contact definition. Possible values of the SymbolicConstant are ORIGINAL, SPOS, or SNEG.

mainSecondaryAssignments

A sequence of tuples specifying main-secondary assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the first surface that defines the main-secondary assignment.
- A region object specifying the second surface in the main-secondary assignment definition.
- A SymbolicConstant specifying the status of the first surface. Possible values are MAIN, SECONDARY, and BALANCED.

initializationAssignments

A sequence of tuples specifying the contact initialization data assigned to each surface pair. Each tuple contains three entries:

- A *Region* object or the SymbolicConstant GLOBAL.
- A *Region* object or the SymbolicConstant SELF.
- A String specifying a *StdInitialization* object associated with this pair of regions.

stabilizationAssignments

A sequence of tuples specifying the contact stabilization assigned to each surface pair. Each tuple contains three entries:

- A *Region* object or the SymbolicConstant GLOBAL.

- A *Region* object or the SymbolicConstant SELF.
- A String specifying a *StdStabilization* object associated with this pair of regions.

smoothingAssignments

A sequence of tuples specifying the surface smoothing assignments in the contact domain. Each tuple contains two entries:

- A region object specifying the surface to which the smoothing option is assigned.
- A SymbolicConstant specifying the smoothing option to be used in the contact definition. Possible values of the SymbolicConstant are NONE, REVOLUTION, SPHERICAL, or TOROIDAL.

slidingTransitionAssignments

A sequence of tuples specifying sliding transition assignments in the contact domain. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL specifying the first surface that defines the sliding transition assignment.
- A region object specifying the second surface in the sliding transition assignment definition.
- A SymbolicConstant specifying the smoothness value. Possible values are ELEMENT_ORDER_SMOOTHING, LINEAR_SMOOTHING, and QUADRATIC_SMOOTHING.

Return value

A ContactStd object.

Exceptions

None.

Members

The ContactStd object can have the following members:

name

A String specifying the repository key.

globalSmoothing

A Boolean specifying whether surface smoothing (geometric correction) is automatically applied to all eligible surfaces. The default value is ON.

includedPairs

A *RegionPairs* object specifying the domain pairs included in contact.

excludedPairs

A *RegionPairs* object specifying the domain pairs excluded from contact.

contactPropertyAssignments

A *ContactPropertyAssignment* object specifying the contact property assignments in the contact domain.

surfaceThicknessAssignments

A *SurfaceThicknessAssignment* object specifying the surface thickness assignments in the contact domain.

surfaceOffsetAssignments

A *SurfaceOffsetAssignment* object specifying the surface offset fraction assignments in the contact domain.

mainSecondaryAssignments

A *MainSecondaryAssignment* object specifying the main-secondary assignments in the contact domain.

initializationAssignments

An *InitializationAssignment* object specifying the contact initialization assignments in the contact domain.

stabilizationAssignments

A *StabilizationAssignment* object specifying the contact stabilization assignments in the contact domain.

smoothingAssignments

A *SmoothingAssignment* object specifying the surface smoothing assignments in the contact domain.

surfaceFeatureAssignments

A *SurfaceFeatureAssignment* object specifying the surface feature angle assignments in the contact domain.

slidingTransitionAssignments

A *SlidingTransitionAssignment* object specifying the sliding transition assignments in the contact domain.

assignBeamCrossSection

A Boolean specifying whether to assign the beam cross section. The default value is False.

beamCrossSection

A SymbolicConstant specifying whether the contact calculations should be based on a circumferential approximation or the actual cross section of the beam element. Possible values are CIRCUMSCRIBED_CIRCLE and EXACT.

assignEdgeToEdgeformulation

A Boolean specifying whether to assign the edge-to-edge formulation. The default value is False.

edgeToEdgeFormulation

A symbolic constant specifying edge-to-edge formulation. The default value is NO.

Corresponding analysis keywords

- **CONTACT*

ContactTangentialBehavior object

The ContactTangentialBehavior object specifies tangential behavior for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].tangentialBehavior
```

TangentialBehavior(...)

This method creates a ContactTangentialBehavior object.

Path

```
mdb.models[name].interactionProperties[name].TangentialBehavior
```

Required arguments

None.

Optional arguments

formulation

A SymbolicConstant specifying the friction formulation. Possible values are FRICTIONLESS, PENALTY, EXPONENTIAL_DECAY, ROUGH, LAGRANGE, and USER_DEFINED. The default value is FRICTIONLESS.

directionality

A SymbolicConstant specifying the directionality of the friction. Possible values are ISOTROPIC and ANISOTROPIC. The default value is ISOTROPIC.

slipRateDependency

A Boolean specifying whether the data depend on slip rate. The default value is OFF.

pressureDependency

A Boolean specifying whether the data depend on contact pressure. The default value is OFF.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variables. The default value is 0.

exponentialDecayDefinition

A SymbolicConstant specifying the exponential decay definition. Possible values are COEFFICIENTS and TEST_DATA. The default value is COEFFICIENTS.

table

A sequence of sequences of Floats specifying tangential behavior. The items in the table data are described below.

shearStressLimit

`None` or a Float specifying the shear stress limit. If *shearStressLimit*=`None`, there is no upper limit. The default value is `None`.

maximumElasticSlip A SymbolicConstant specifying what the maximum elastic slip will be. Possible values are FRACTION and ABSOLUTE_DISTANCE. The default value is FRACTION.

fraction A Float specifying the fraction of a characteristic surface dimension. The default value is 0.0.

absoluteDistance A Float specifying the absolute distance. The default value is 0.0.

elasticSlipStiffness `None` or a Float specifying the elastic slip stiffness. If *elasticSlipStiffness*=`None`, there is no upper limit. The default value is `None`.

nStateDependentVars An Int specifying the number of state-dependent variables. The default value is 0.

useProperties A Boolean specifying whether property values will be used. The default value is OFF.

Table data

If *formulation*=PENALTY or LAGRANGE, the table data specify the following:

- Friction coefficient in the first slip direction, μ_1.
- Friction coefficient in the second slip direction, μ_2 (if *directionality*=ANISOTROPIC).
- Slip rate, if the data depend on slip rate.
- Contact pressure, if the data depend on contact pressure.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=COEFFICIENTS, the table data specify the following:

- Static friction coefficient.
- Kinetic friction coefficient.
- Decay coefficient.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=TEST_DATA, the table data specify the following:

- Friction coefficient.
- Slip rate.

If *formulation*=USER_DEFINED, the table data specify the following:

- Friction property.

Return value

A ContactTangentialBehavior object.

Exceptions

None.

setValues(...)

This method modifies the ContactTangentialBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContactTangentialBehavior* method.

Return value

`None.`

Exceptions

None.

Members

The ContactTangentialBehavior object has the following members:

formulation

A SymbolicConstant specifying the friction formulation. Possible values are FRICTIONLESS, PENALTY, EXPONENTIAL_DECAY, ROUGH, LAGRANGE, and USER_DEFINED. The default value is FRICTIONLESS.

directionality

A SymbolicConstant specifying the directionality of the friction. Possible values are ISOTROPIC and ANISOTROPIC. The default value is ISOTROPIC.

slipRateDependency

A Boolean specifying whether the data depend on slip rate. The default value is OFF.

pressureDependency

A Boolean specifying whether the data depend on contact pressure. The default value is OFF.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variables. The default value is 0.

exponentialDecayDefinition

A SymbolicConstant specifying the exponential decay definition. Possible values are COEFFICIENTS and TEST_DATA. The default value is COEFFICIENTS.

shearStressLimit
`None` or a Float specifying the shear stress limit. If *shearStressLimit*=`None`, there is no upper limit. The default value is `None`.

maximumElasticSlip
A SymbolicConstant specifying what the maximum elastic slip will be. Possible values are FRACTION and ABSOLUTE_DISTANCE. The default value is FRACTION.

fraction
A Float specifying the fraction of a characteristic surface dimension. The default value is 0.0.

absoluteDistance
A Float specifying the absolute distance. The default value is 0.0.

elasticSlipStiffness
`None` or a Float specifying the elastic slip stiffness. If *elasticSlipStiffness*=`None`, there is no upper limit. The default value is `None`.

nStateDependentVars
An Int specifying the number of state-dependent variables. The default value is 0.

useProperties
A Boolean specifying whether property values will be used. The default value is OFF.

table
A tuple of tuples of Floats specifying tangential behavior. The items in the table data are described below.

Corresponding analysis keywords

- **FRICTION*
- **CHANGE FRICTION*

CyclicSymmetry object

The CyclicSymmetry object defines a cyclic symmetry analysis.

The CyclicSymmetry object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

CyclicSymmetry(...)

This method creates a CyclicSymmetry object.

Path

```
mdb.models[name].CyclicSymmetry
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the cyclic symmetry interaction should be created.

main
A *Region* object specifying the main surface.

secondary
A *Region* object specifying the secondary surface.

repetitiveSectors
An Int specifying the total number of sectors in the cyclic symmetric model.

axisPoint1
A *Region* object specifying the first point of the axis of symmetry. The region should contain exactly one mesh node, vertex, interesting point, reference point, or datum point. In a two-dimensional model *axisPoint1* is the only point used to define the axis of symmetry.

axisPoint2
A *Region* object specifying the second point of the axis of symmetry. The region should contain exactly one mesh node, vertex, interesting point, reference point, or datum point. This point is ignored in a two-dimensional model.

Optional arguments

extractedNodalDiameter
A SymbolicConstant specifying whether Abaqus should extract all possible nodal diameters or the nodal diameters between the user-specified values for *lowestNodalDiameter* and *highestNodalDiameter*. Possible values are ALL_NODAL_DIAMETER and SPECIFIED_NODAL_DIAMETER. The default value is ALL_NODAL_DIAMETER.

lowestNodalDiameter
An Int specifying the lowest nodal diameter to be used in the eigenfrequency analysis. The default value is 0.

highestNodalDiameter An Int specifying the highest nodal diameter to be used in the eigenfrequency analysis. This argument value should be less than or equal to the half of the total number of sectors (as specified in the *repetitiveSectors* parameter). The default value is 0.

excitationNodalDiameter An Int specifying the nodal diameter for which the modal-based steady-state dynamic analysis will be performed. This value should be greater than or equal to the lowest nodal diameter (specified in the *lowestNodalDiameter* parameter), and less than or equal to the highest nodal diameter (specified in the *highestNodalDiameter* parameter). The default value is 0.

adjustTie A Boolean specifying whether or not to adjust the secondary surface of the cyclic symmetry to tie it to the main surface. The default value is ON.

positionTolerance A Float specifying the position tolerance. The*positionTolerance* argument applies only when *positionToleranceMethod*=SPECIFY_TOLERANCE. The default value is 0.0.

positionToleranceMethod A SymbolicConstant specifying the method used to determine the position tolerance. Possible values are COMPUTED_TOLERANCE and SPECIFY_TOLERANCE. The default value is COMPUTED_TOLERANCE.

Return value

A CyclicSymmetry object.

Exceptions

None.

swapSurfaces()

This method switches the main and secondary surfaces of a cyclic symmetry interaction. This command is valid only during the step in which the interaction is created.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the data for an existing CyclicSymmetry object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CyclicSymmetry* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing CyclicSymmetry object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

extractedNodalDiameter A SymbolicConstant specifying whether Abaqus should extract all possible nodal diameters or the nodal diameters between the user-specified values for *lowestNodalDiameter* and *highestNodalDiameter*. Possible values are ALL_NODAL_DIAMETER and SPECIFIED_NODAL_DIAMETER. The default value is ALL_NODAL_DIAMETER.

lowestNodalDiameter An Int specifying the lowest nodal diameter to be used in the eigenfrequency analysis. The default value is 0.

highestNodalDiameter An Int specifying the highest nodal diameter to be used in the eigenfrequency analysis. This argument value should be less than or equal to the half of the total number of sectors (as specified in the *repetitiveSectors* parameter). The default value is 0.

excitationNodalDiameter An Int specifying the nodal diameter for which the modal-based steady-state dynamic analysis will be performed. This value should be greater than or equal to the lowest nodal diameter (specified in the *lowestNodalDiameter* parameter), and less than or equal to the highest nodal diameter (specified in the *highestNodalDiameter* parameter). The default value is 0.

Return value

`None.`

Exceptions

None.

Members

The CyclicSymmetry object has members with the same names and descriptions as the arguments to the *CyclicSymmetry* method except the optional arguments to the *setValuesInStep* method.

Corresponding analysis keywords

- **CLOAD*, CYCLIC MODE
- **CYCLIC SYMMETRY MODEL*
- **DLOAD*, CYCLIC MODE
- **DSLOAD*, CYCLIC MODE
- **SELECT CYCLIC SYMMETRY MODES*
- **TIE*, CYCLIC SYMMETRY

CyclicSymmetryState object

The CyclicSymmetryState object stores the propagating data for a *CyclicSymmetry* object. One instance of this object is created internally by the *CyclicSymmetry* object for each step. The instance is also deleted internally by the *CyclicSymmetry* object.

The CyclicSymmetryState object has no constructor or methods.

The CyclicSymmetryState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The CyclicSymmetryState object has the following members:

extractedNodalDiameter

A SymbolicConstant specifying whether Abaqus should extract all possible nodal diameters or the nodal diameters between the user-specified values for *lowestNodalDiameter* and *highestNodalDiameter*. Possible values are ALL_NODAL_DIAMETER and SPECIFIED_NODAL_DIAMETER. The default value is ALL_NODAL_DIAMETER.

extractedNodalDiameterState

A SymbolicConstant specifying the propagation state of the *extractedNodalDiameter* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

lowestNodalDiameter

An Int specifying the lowest nodal diameter to be used in the eigenfrequency analysis. The default value is 0.

lowestNodalDiameterState

A SymbolicConstant specifying the propagation state of the *lowestNodalDiameter* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

highestNodalDiameter

An Int specifying the highest nodal diameter to be used in the eigenfrequency analysis. This argument value should be less than or equal to the half of the total number of sectors. The default value is 0.

highestNodalDiameterState

A SymbolicConstant specifying the propagation state of the *highestNodalDiameter* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

excitiationNodalDiameter

An Int specifying the nodal diameter for which the modal-based steady-state dynamic analysis will be performed. This value should be greater than or equal to the lowest nodal diameter (specified in the *lowestNodalDiameter* parameter), and less than or equal to the highest nodal diameter (specified in the *highestNodalDiameter* parameter). The default value is 0.

excitiationNodalDiameterState

A SymbolicConstant specifying the propagation state of the *excitiationNodalDiameter* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CLOAD*, CYCLIC MODE
- **DLOAD*, CYCLIC MODE
- **DSLOAD*, CYCLIC MODE
- **SELECT CYCLIC SYMMETRY MODES*

ElasticFoundation object

The ElasticFoundation object defines a mechanical foundation.

The ElasticFoundation object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ElasticFoundation(...)

This method creates an ElasticFoundation object.

Path

```
mdb.models[name].ElasticFoundation
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the ElasticFoundation object is created. *createStepName* must be set to 'Initial'.
surface	A *Region* object specifying the surface to which the foundation applies.
stiffness	A Float specifying the foundation stiffness per area (or per length for beams).

Optional arguments

None.

Return value

An ElasticFoundation object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ElasticFoundation object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ElasticFoundation* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing ElasticFoundation object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

stiffness A Float specifying the foundation stiffness per area (or per length for beams).

Return value

`None.`

Exceptions

None.

Members

The ElasticFoundation object has members with the same names and descriptions as the arguments to the *ElasticFoundation* method except the optional arguments to the *setValuesInStep* method.

Corresponding analysis keywords

- **FOUNDATION*

ElasticFoundationState object

The ElasticFoundationState object stores the propagating data for an *ElasticFoundation* object. One instance of this object is created internally by the *ElasticFoundation* object for each step. The instance is also deleted internally by the *ElasticFoundation* object.

The ElasticFoundationState object has no constructor or methods.

The ElasticFoundationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The ElasticFoundationState object has the following members:

stiffness

A Float specifying the foundation stiffness per area.

stiffnessState

A SymbolicConstant specifying the propagation state of the stiffness member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

ExpContactControl object

The ExpContactControl object is used in Abaqus/Explicit analyses to specify optional solution controls for problems involving contact between bodies.

The ExpContactControl object is derived from the *ContactControl* object.

Access

```
import interaction
mdb.models[name].contactControls[name]
```

ExpContactControl(...)

This method creates an ExpContactControl object.

Path

```
mdb.models[name].ExpContactControl
```

Required arguments

name
A String specifying the contact controls repository key.

Optional arguments

globTrkChoice
A SymbolicConstant specifying whether or not the default value will be used for the maximum number of increments between global contact searches. Possible values are DEFAULT and SPECIFY. The default value is DEFAULT.

globTrkInc
An Int specifying the maximum number of increments between global contact searches. The *globTrkInc* argument applies only when *globTrkChoice*=SPECIFY. The default value is 100 for surface-to-surface contact and 4 for self-contact.

fastLocalTrk
A Boolean specifying whether to use the more computationally efficient local tracking method. The default value is ON.

scalePenalty
A Float specifying the factor by which Abaqus/Explicit will scale the default penalty stiffness to obtain the stiffnesses used for the penalty contact pairs. The default value is 1.0.

warpCheckPeriod
An Int specifying the number of increments between checks for highly warped facets on main surfaces. The default value is 20.

warpCutoff
A Float specifying the out-of-plane warping angle (in degrees), at which a facet will be considered to be highly warped. The default value is 20.0.

Return value

An ExpContactControl object.

Exceptions

RangeError.

setValues(...)

This method modifies the ExpContactControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ExpContactControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ExpContactControl object has members with the same names and descriptions as the arguments to the *ExpContactControl* method.

Corresponding analysis keywords

- **CONTACT CONTROLS*

ExpInitialization object

The ExpInitialization object is used in conjunction with *ContactExp* in Abaqus/Explicit analyses to specify contact initialization data.

The ExpInitialization object is derived from the *ContactInitialization* object.

Access

```
import interaction
mdb.models[name].contactInitializations[name]
```

ExpInitialization(...)

This method creates an ExpInitialization object.

Path

```
mdb.models[name].ExpInitialization
```

Required arguments

name
A String specifying the contact initialization repository key.

Optional arguments

overclosureType
A SymbolicConstant specifying the type of overclosure to be defined. Possible values are ADJUST, INTERFERENCE, and CLEARANCE. The default value is ADJUST.

interferenceDistance
`None` or a Float specifying the interference distance. This argument is valid only when *overclosureType*=INTERFERENCE. The default value is `None`.

clearanceDistance
`None` or a Float specifying the initial clearance distance. This argument is valid only when *overclosureType*=CLEARANCE and must be specified in that case. The default value is `None`.

openingTolerance
`None` or a Float specifying the distance tolerance within which initial openings will undergo strain-free adjustments. This argument is not valid when *overclosureType*=INTERFERENCE unless a value has been specified for *interferenceDistance*. The default value is `None`.

overclosureTolerance
`None` or a Float specifying the distance tolerance within which initial overclosures will undergo strain-free adjustments. The default value is `None`.

adjustNodalCoords
A Boolean specifying whether to resolve clearances/overclosures by adjusting the nodal coordinates without creating strain in the model. *adjustNodalCoords*=True can be used only for clearances/overclosures defined in the first step of an analysis. The default value is `True`.

secondaryNodesetName A String specifying the name of the node set containing the secondary nodes to be included in the initial clearance specification. This argument is not valid when *overclosureType*=INTERFERENCE and if *openingTolerance* or *overclosureTolerance* is specified. The default value is `None`.

stepFraction A Float specifying the fraction of the step time (between 0.0 and 1.0) in which the interference fit has to be solved. The default value is 1.0. This argument is valid only when *overclosureType*=INTERFERENCE.

Return value

An ExpInitialization object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ExpInitialization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ExpInitialization* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ExpInitialization object has members with the same names and descriptions as the arguments to the *ExpInitialization* method.

Corresponding analysis keywords

- **CONTACT INITIALIZATION DATA*

FilmCondition object

The FilmCondition object defines film coefficients and associated sink temperatures for coupled temperature-displacement analyses.

The FilmCondition object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FilmCondition(...)

This method creates a FilmCondition object.

Path

```
mdb.models[name].FilmCondition
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the FilmCondition object is created.

region
A *Region* object specifying the name of the surface to which the film condition interaction is applied or an Element set object specifying the elements to which the element-based film condition is applied.

definition
A SymbolicConstant specifying how the film condition is defined. Possible values are EMBEDDED_COEFF, PROPERTY_REF, USER_SUB, and FIELD.

Optional arguments

interactionProperty
A String specifying the name of the *FilmConditionProp* object associated with this interaction. The *interactionProperty* argument applies only when *definition*=PROPERTY_REF. The default value is an empty string.

sinkTemperature
A Float specifying the reference sink temperature, θ^0. The default value is 0.0.

sinkAmplitude
A String specifying the name of the *Amplitude* object that gives the variation of the sink temperature, θ^0, with time. The default value is an empty string.



Note:

Use empty string in an Abaqus/Standard analysis to specify that the reference sink temperature is applied immediately at the beginning of the step or linearly over the step. Use empty string in an Abaqus/Explicit analysis to specify that the reference sink temperature is applied throughout the step.

filmCoeff

A Float specifying the reference film coefficient value, h. The *filmCoeff* argument applies when *definition*=EMBEDDED_COEFF, *definition*=USER_SUB, or *definition*=FIELD. The default value is 0.0.

filmCoeffAmplitude

A String specifying the name of the *Amplitude* object that gives the variation of the film coefficient, h, with time. The default value is an empty string.



Note: Use empty string in an Abaqus/Standard analysis to specify that the reference film coefficient is applied immediately at the beginning of the step or linearly over the step. Use empty string in an Abaqus/Explicit analysis to specify that the reference film coefficient is applied throughout the step.

field

A String specifying the name of the *AnalyticalField* object associated with this interaction. The *field* argument applies only when *definition*=FIELD. The default value is an empty string.

sinkFieldName

A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with the sink temperature. The *sinkFieldName* argument applies only when *sinkDistributionType*=ANALYTICAL_FIELD or *sinkDistributionType*=DISCRETE_FIELD. The default value is an empty string.

sinkDistributionType

A SymbolicConstant specifying how the sink temperature is distributed. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

dynamicSurfaceActivation

A Boolean specifying the activation of an element-based film condition on evolving faces of an element.

Return value

A FilmCondition object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing FilmCondition object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FilmCondition* method, except for the *name* and *createStepName* arguments.

Return value
`None.`

Exceptions
None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing FilmCondition object in the specified step.

Required arguments

stepName	A String specifying the name of the step in which the interaction is modified.

Optional arguments
The optional arguments to `setValuesInStep` are the same as the optional arguments to the `FilmConditionState` method.

Return value
`None.`

Exceptions
None.

Members
The FilmCondition object has members with the same names and descriptions as the arguments to the *FilmCondition* method except the optional arguments to the *setValuesInStep* method.

FilmConditionProp object

The FilmConditionProp object is an interaction property that defines a film coefficient as a function of temperature and field variables.

The FilmConditionProp object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

FilmConditionProp(...)

This method creates a FilmConditionProp object.

Path

```
mdb.models[name].FilmConditionProp
```

Required arguments

name A String specifying the interaction property repository key.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

property A sequence of sequences of Floats specifying the following:

- The film coefficient, h.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FilmConditionProp object.

Exceptions

None.

setValues(...)

This method modifies the FilmConditionProp object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FilmConditionProp* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FilmConditionProp object has members with the same names and descriptions as the arguments to the *FilmConditionProp* method.

Corresponding analysis keywords

- **FILM PROPERTY*

FilmConditionState object

The FilmConditionState object stores the propagating data for a *FilmCondition* object. One instance of this object is created internally by the *FilmCondition* object for each step. The instance is also deleted internally by the *FilmCondition* object.

The FilmConditionState object has no constructor or methods.

The FilmConditionState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FilmConditionState object has the following members:

interactionPropertyState

A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

sinkTemperature

A Float specifying the sink temperature.

sinkTemperatureState

A SymbolicConstant specifying the propagation state of the *sinkTemperature* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

sinkAmplitudeState

A SymbolicConstant specifying the propagation state of the *sinkAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

filmCoeff

A Float specifying the film coefficient.

filmCoeffState

A SymbolicConstant specifying the propagation state of the *filmCoeff* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

filmCoeffAmplitudeState

A SymbolicConstant specifying the propagation state of the *filmCoeffAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty

A String specifying the *FilmConditionProp* object associated with this interaction.

sinkAmplitude

A String specifying the name of the Amplitude object that gives the variation of the sink temperature.

filmCoeffAmplitude

A String specifying the name of the Amplitude object that gives the variation of the film coefficient.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED

- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **SFILM*
- **FILM*

FluidCavity object

The FluidCavity object defines a surface-based cavity.

The FluidCavity object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FluidCavity(...)

This method creates an FluidCavity object.

Path

```
mdb.models[name].FluidCavity
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the FluidCavity object is created.
cavityPoint	A *Region* object specifying the fluid cavity reference point.
cavitySurface	A *Region* object specifying the surface forming the boundary of the fluid cavity.
interactionProperty	A String specifying the *FluidCavityProperty* object associated with this interaction.

Optional arguments

ambientPressure	A Float specifying the magnitude of the ambient pressure. The default value is 0.0.
thickness	A Float specifying the out-of-plane thickness of the surface for two-dimensional models. This argument is valid only when using two-dimensional models. The default value is 1.0.
useAdiabatic	A Boolean specifying whether adiabatic behavior is assumed for the ideal gas. This argument is valid only when *interactionProperty* specifies a pneumatic definition. The default value is OFF.
checkNormals	A Boolean specifying whether the analysis will check the consistency of the surface normals. The default value is ON.

Return value

A FluidCavity object.

Exceptions

None.

setValues(...)

This method modifies the FluidCavity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidCavity* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The FluidCavity object has members with the same names and descriptions as the arguments to the *FluidCavity* method.

Corresponding analysis keywords

- **FLUID CAVITY*

FluidCavityProperty object

The FluidCavityProperty object is an interaction property that defines the fluid behavior for a surface-based fluid cavity.

The FluidCavityProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

FluidCavityProperty(...)

This method creates a FluidCavityProperty object.

Path

```
mdb.models[name].FluidCavityProperty
```

Required arguments

name A String specifying the interaction property repository key.

Optional arguments

definition A SymbolicConstant specifying the type of fluid cavity property to be defined. Possible values are HYDRAULIC and PNEUMATIC. The default value is HYDRAULIC.

fluidDensity `None` or a Float specifying the reference fluid density. This argument is applicable only when *definition*=HYDRAULIC, and is required in that case. The default value is `None`.

molecularWeight `None` or a Float specifying the molecular weight of the ideal gas species. This argument is applicable only when *definition*=PNEUMATIC, and is required in that case. The default value is `None`.

useExpansion A Boolean specifying whether thermal expansion coefficients will be defined. This argument is applicable only when *definition*=HYDRAULIC. The default value is OFF.

expansionTempDep A Boolean specifying whether the thermal fluid expansion data will have temperature dependency. This argument is applicable only when *definition*=HYDRAULIC and when *useExpansion*=True. The default value is OFF.

expansionDependencies An Int specifying the number of field variable dependencies in the thermal fluid expansion data. This argument is applicable only when *definition*=HYDRAULIC and when *useExpansion*=True. The default value is 0.

referenceTemperature A Float specifying the reference temperature for the coefficient of thermal expansion. This argument is applicable only when *definition*=HYDRAULIC, when

useExpansion=True, and when either *expansionTempDep*=True or when *expansionDependencies* is greater than 0. The default value is 0.0.

expansionTable

A sequence of sequences of Floats specifying the thermal expansion coefficients. This argument is applicable only when *definition*=HYDRAULIC and when *useExpansion*=True. Each sequence contains the following data:

- The mean coefficient of thermal expansion.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

useBulkModulus

A Boolean specifying whether fluid bulk modulus values will be defined. This argument is applicable only when *definition*=HYDRAULIC. The default value is OFF.

bulkModulusTempDep

A Boolean specifying whether the fluid bulk modulus data will have temperature dependency. This argument is applicable only when *definition*=HYDRAULIC and when *useBulkModulus*=True. The default value is OFF.

bulkModulusDependencies

An Int specifying the number of field variable dependencies in the fluid bulk modulus data. This argument is applicable only when *definition*=HYDRAULIC and when *useBulkModulus*=True. The default value is 0.

bulkModulusTable

A sequence of sequences of Floats specifying the fluid bulk modulus values. This argument is applicable only when *definition*=HYDRAULIC and when *useBulkModulus*=True. Each sequence contains the following data:

- The fluid bulk modulus.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

useCapacity

A Boolean specifying whether molar heat capacity values will be defined. This argument is applicable only when *definition*=PNEUMATIC. The default value is OFF.

capacityType

A SymbolicConstant specifying the method to define the molar heat capacity. Possible values are POLYNOMIAL and TABULAR. The default value is POLYNOMIAL.

capacityTempDep

A Boolean specifying whether the molar heat capacity data will have temperature dependency. This argument is applicable only when *definition*=PNEUMATIC, when *useCapacity*=True, and when *capacityType*=TABULAR. The default value is OFF.

capacityDependencies

An Int specifying the number of field variable dependencies in the molar heat capacity data. This argument is applicable only when *definition*=PNEUMATIC, when *useCapacity*=True, and when *capacityType*=TABULAR. The default value is 0.

capacityTable

A sequence of sequences of Floats specifying the molar heat capacity values in the form of a polynomial expression. This argument is applicable only when *definition*=PNEUMATIC, when *useCapacity*=True, and when *capacityType*=POLYNOMIAL. In this form, only one sequence is specified and that sequence contains the following data:

- The first molar heat capacity coefficient.
- The second molar heat capacity coefficient.
- The third molar heat capacity coefficient.
- The fourth molar heat capacity coefficient.
- The fifth molar heat capacity coefficient.

Alternatively, the sequence data may specify the molar heat capacity values at constant pressure for an ideal gas species. This argument is applicable only when *definition*=PNEUMATIC, when *useCapacity*=True, and when *capacityType*=TABULAR. Each sequence contains the following data:

- The molar heat capacity at constant pressure.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FluidCavityProperty object.

Exceptions

None.

`setValues(...)`

This method modifies the FluidCavityProperty object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidCavityProperty* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FluidCavityProperty object has members with the same names and descriptions as the arguments to the *FluidCavityProperty* method.

Corresponding analysis keywords

- **FLUID BEHAVIOR*
- **CAPACITY*
- **FLUID BULK MODULUS*
- **FLUID DENSITY*
- **FLUID EXPANSION*
- **MOLECULAR WEIGHT*

FluidCavityState object

The FluidCavityState object stores the propagating data for an *FluidCavity* object. One instance of this object is created internally by the *FluidCavity* object for each step. The instance is also deleted internally by the *FluidCavity* object.

The FluidCavityState object has no constructor or methods.

The FluidCavityState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FluidCavityState object has the following member:

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

FluidExchangeActivation object

The FluidExchnageActivation object is used to define the activation of fluid exchanges within the fluid cavity.

The FluidExchangeActivation object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FluidExchangeActivation(...)

This method creates a FluidExchangeActivation object.

Path

```
mdb.models[name].FluidExchangeActivation
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the FluidExchange object is created.
exchanges	A List specifying fluid exchanges to be activated.

Optional arguments

amplitude	A String specifying the name of the amplitude curve defining a mapping between the inflation time and the actual time.
isBlockage	A Boolean specifying the vent and leakage area obstruction by contacted surfaces.
isOnlyOutflow	A Boolean specifying if the flow of fluid is only from the first fluid cavity to the second fluid cavity defined in the *FluidExchange* object.
deltaLeakageArea	A Float specifying the ratio of the actual surface area over the initial surface area at which you want the fluid to leak.

Return value

A FluidExchangeActivation object.

Exceptions

None.

`setValues(...)`

This method modifies the FluidExchangeActivation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidExchangeActivation* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The FluidExchangeActivation object has members with the same names and descriptions as the arguments to the *FluidExchangeActivation* method.

Corresponding analysis keywords

- **FLUIDEXCHANGEACTIVATION*

FluidExchangeActivationState object

The FluidExchangeActivationState object stores the propagating data for a FluidExchangeActivation object. One instance of this object is created internally by the *FluidExchangeActivation* object for each step. The instance is also deleted internally by the *FluidExchangeActivation* object.

The FluidExchangeActivationState object has no constructor or methods.

The FluidExchangeActivationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FluidExchangeActivationState object has the following members:

exchangesState
A SymbolicConstant specifying the propagation state of the *exchanges* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

exchanges
A String specifying the name of the *FluidExchange* object associated with this interaction.

amplitudeState
A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitude
A String specifying the name of the *Amplitude* object associated with this interaction.

isBlockageState
A SymbolicConstant specifying the propagation state of the *isBlockage* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

isBlockage
A Boolean specifying whether to consider vent and leakage area obstruction by contacted surfaces.

isOnlyOutflowState
A SymbolicConstant specifying the propagation state of the *isOnlyOutflow* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

isOnlyOutflow
A Boolean specifying whether the flow of fluid is allowed only from the first fluid cavity to the second fluid cavity defined in the *FluidExchange* object.

deltaLeakageAreaState
A SymbolicConstant specifying the propagation state of the *deltaLeakageArea* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

deltaLeakageArea
A Float specifying the ratio of the actual surface area over the initial surface area at which you want the fluid to leak.

status
A *SymbolicConstant* specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE

- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

FluidExchange object

The FluidExchange object is used to define fluid exchange between two fluid cavities or between a fluid cavity and its environment.

The FluidExchange object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FluidExchange(...)

This method creates an FluidExchange object.

Path

```
mdb.models[name].FluidExchange
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the FluidExchange object is created.

firstCavity
A String specifying the first *FluidCavity* object associated with this interaction. This will be the only cavity specified if *definition*=TO_ENVIRONMENT.

interactionProperty
A String specifying the *FluidExchangeProperty* object associated with this interaction.

Optional arguments

definition
A SymbolicConstant specifying the type of fluid exchange to be defined. Possible values are TO_ENVIRONMENT and BETWEEN_CAVITIES. The default value is TO_ENVIRONMENT.

secondCavity
A String specifying the second *FluidCavity* object associated with this interaction. This argument is applicable only when *definition*=BETWEEN_CAVITIES.

exchangeArea
A Float specifying the effective exchange area. The default value is 1.0.

Return value

A FluidExchange object.

Exceptions

None.

`setValues(...)`

This method modifies the FluidExchange object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidExchange* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The FluidExchange object has members with the same names and descriptions as the arguments to the *FluidExchange* method.

Corresponding analysis keywords

- **FLUID EXCHANGE*

FluidExchangeProperty object

The FluidExchangeProperty object is an interaction property that defines the fluid exchange property for a flow between two fluid cavities or between a fluid cavity and its environment.

The FluidExchangeProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

FluidExchangeProperty(...)

This method creates a FluidExchangeProperty object.

Path

```
mdb.models[name].FluidExchangeProperty
```

Required arguments

name A String specifying the interaction property repository key.

dataTable A sequence of sequences of Floats specifying the viscous and hydrodynamic resistance coefficients when *definition*=BULK_VISCOSITY. Each sequence contains the following data:

- The viscous resistance coefficient.
- The hydrodynamic resistance coefficient.
- The average absolute pressure, if the data depend on pressure.
- The average temperature, if the data depend on temperature.
- The value of the first field variable, if the data depend on field variables.
- The value of the second field variable.
- Etc.

Alternatively, the sequence data may specify the mass flow rate. This is applicable only when *definition*=MASS_FLUX. In this form, only one sequence is specified and that sequence contains the following data:

- The mass flow rate per unit area.

Alternatively, the sequence data may specify the mass rate leakage. This is applicable only when *definition*=MASS_RATE_LEAK. Each sequence contains the following data:

- The absolute value of the mass flow rate per unit area. (The first tabular value entered must always be zero.)
- The absolute value of the pressure difference. (The first tabular value entered must always be zero.)
- The average absolute pressure, if the data depend on pressure.
- The average temperature, if the data depend on temperature.
- The value of the first field variable, if the data depend on field variables.

- The value of the second field variable.
- Etc.

Alternatively, the sequence data may specify the volume flow rate. This is applicable only when *definition*=VOL_FLUX. In this form, only one sequence is specified and that sequence contains the following data:

- The volumetric flow rate per unit area.

Alternatively, the sequence data may specify the volume rate leakage. This is applicable only when *definition*=VOL_RATE_LEAK. Each sequence contains the following data:

- The absolute value of the volumetric flow rate per unit area. (The first tabular value entered must always be zero.)
- The absolute value of the pressure difference. (The first tabular value entered must always be zero.)
- The average absolute pressure, if the data depend on pressure.
- The average temperature, if the data depend on temperature.
- The value of the first field variable, if the data depend on field variables.
- The value of the second field variable.
- Etc.

Optional arguments

definition A SymbolicConstant specifying the type of fluid exchange property to be defined. Possible values are BULK_VISCOSITY, MASS_FLUX, MASS_RATE_LEAK, VOL_FLUX, and VOL_RATE_LEAK. The default value is BULK_VISCOSITY.

pressureDependency A Boolean specifying whether the data will have pressure dependency. This argument is applicable only when *definition*=BULK_VISCOSITY, or when *definition*=MASS_RATE_LEAK, or when *definition*=VOL_RATE_LEAK. The default value is OFF.

temperatureDependency A Boolean specifying whether the data will have temperature dependency. This argument is applicable only when *definition*=BULK_VISCOSITY, or when *definition*=MASS_RATE_LEAK, or when *definition*=VOL_RATE_LEAK. The default value is OFF.

fieldDependencies An Int specifying the number of field variable dependencies in the data. This argument is applicable only when *definition*=BULK_VISCOSITY, or when *definition*=MASS_RATE_LEAK, or when *definition*=VOL_RATE_LEAK. The default value is 0.

Return value

A FluidExchangeProperty object.

Exceptions

None.

`setValues(...)`

This method modifies the FluidExchangeProperty object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidExchangeProperty* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FluidExchangeProperty object has members with the same names and descriptions as the arguments to the *FluidExchangeProperty* method.

Corresponding analysis keywords

- **FLUID EXCHANGE PROPERTY*

FluidExchangeState object

The FluidExchangeState object stores the propagating data for an *FluidExchange* object. One instance of this object is created internally by the *FluidExchange* object for each step. The instance is also deleted internally by the *FluidExchange* object.

The FluidExchangeState object has no constructor or methods.

The FluidExchangeState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FluidExchangeState object has the following member:

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

FluidInflatorActivation object

The FluidInflatorActivation object is used to define the activation of fluid inflators to model the deployment of an airbag.

The FluidInflatorActivation object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FluidInflatorActivation(...)

This method creates a FluidInflatorActivation object.

Path

```
mdb.models[name].FluidInflatorActivation
```

Required arguments

name	A string specifying the repository key.
createStepName	A string specifying the name of the step in which the FluidInflator object is created.
inflators	A list specifying fluid inflators to be activated.

Optional arguments

inflationTimeAmplitude	A string specifying the name of the amplitude curve defining a mapping between the inflation time and the actual time.
massFlowAmplitude	A string specifying the name of the amplitude curve by which to modify the mass flow rate.

Return value

A FluidInflatorActivation object.

Exceptions

None.

setValues(...)

This method modifies the FluidInflatorActivation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidInflatorActivation* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The FluidInflatorActivation object has members with the same names and descriptions as the arguments to the *FluidInflatorActivation* method.

Corresponding analysis keywords

- **FLUIDINFLATORACTIVATION*

FluidInflatorActivationState object

The FluidInflatorActivationState object stores the propagating data for a FluidInflatorActivation object.
One instance of this object is created internally by the *FluidInflatorActivation* object for each step. The instance is also deleted internally by the *FluidInflatorActivation* object.

The FluidInflatorActivationState object has no constructor or methods.

The FluidInflatorActivationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FluidInflatorActivationState object has the following member:

inflatorsState
A SymbolicConstant specifying the propagation state of the *inflators* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

inflators
A String specifying the name of the *FluidInflator* object associated with this interaction.

inflationTimeAmplitudeState
A SymbolicConstant specifying the propagation state of the *inflationTimeAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

inflationTimeAmplitude
A String specifying the name of the *Amplitude* object associated with this interaction.

massFlowAmplitudeState
A SymbolicConstant specifying the propagation state of the *massFlowAmplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

massFlowAmplitude
A String specifying the name of the *Amplitude* object associated with this interaction.

status
A *SymbolicConstant* specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

FluidInflator object

The FluidInflator object is used to define a fluid inflator to model deployment of an airbag.

The FluidInflator object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

FluidInflator(...)

This method creates a FluidInflator object.

Path

```
mdb.models[name].FluidInflator
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the FluidInflator object is created.
cavity	A String specifying the first *FluidCavity* object associated with this interaction.
interactionProperty	A String specifying the *FluidInflatorProperty* object associated with this interaction.

Return value

A FluidInflator object.

Exceptions

None.

setValues(...)

This method modifies the FluidInflator object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidInflator* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The FluidInflator object has members with the same names and descriptions as the arguments to the *FluidInflator* method.

Corresponding analysis keywords

- **FLUID INFLATOR*

FluidInflatorProperty object

The FluidInflatorProperty object is an interaction property that defines a fluid inflator property to model the deployment of an airbag.
The inflator property defines the mass flow rate and temperature as a function of inflation time either directly or by entering tank test data. It also defines the mixture of gases entering the fluid cavity.

The FluidInflatorProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

FluidInflatorProperty(...)

This method creates a FluidInflatorProperty object.

Path

```
mdb.models[name].FluidInflatorProperty
```

Required arguments

name A String specifying the interaction property repository key.

definition A Symbolic constant specifying the method used for modeling the flow characteristics of inflators. The default value is *definition*=DUAL PRESSURE. The possible values are DUAL PRESSURE, PRESSURE AND MASS, TANK TEST, and TEMPERATURE AND MASS.

effectiveArea A Float specifying the total inflator orifice area. This argument is applicable only if *definition*=DUAL PRESSURE or *definition*=PRESSURE AND MASS.

tankVolume A Float specifying the tank volume. This argument is applicable only if *definition*=DUAL PRESSURE or *definition*=TANK TEST.

Optional arguments

dischargeCoefficient A Float specifying the discharge coefficient. This argument is applicable only if *definition*=DUAL PRESSURE or *definition*=PRESSURE AND MASS.

dataTable A sequence of sequences of Floats specifying the items described in the "Table data" section below.

numFluids An Int specifying the number of gas species used for this inflator.

mixtureType A Symbolic constant specifying whether to use mass fraction or the molar fraction for a mixture of ideal gases. The default value is MASS FRACTION. The possible values are MASS FRACTION or MOLAR FRACTION.

inflationTime A sequence of sequences of Floats specifying the inflation time.

fluidbehaviorName	A sequence of sequences of Strings specifying fluid behavior names.
massFraction	A sequence of sequences of Floats specifying the mass fraction or the molar fraction corresponding to entered fluid behavior.

Table data

If *definition*=DUAL PRESSURE, the table data specify the following:

- Inflation time.
- Inflator pressure.
- Tank pressure.

If *definition*=PRESSURE AND MASS, the table data specify the following:

- Inflation time.
- Inflator pressure.
- Inflator mass flow rate.

If *definition*=TANK TEST, the table data specify the following:

- Inflation time.
- Inflator gas temperature.
- Tank pressure.

If *definition*=TEMPERATURE AND MASS, the table data specify the following:

- Inflation time.
- Inflator gas temperature.
- Inflator mass flow rate.

Return value

A FluidInflatorProperty object.

Exceptions

None.

`setValues(...)`

This method modifies the FluidInflatorProperty object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidInflatorProperty* method, except for the *name* argument.

Return value
`None.`

Exceptions
None.

Members
The FluidInflatorProperty object has members with the same names and descriptions as the arguments to the *FluidInflatorProperty* method.

Corresponding analysis keywords
- **FLUID INFLATOR PROPERTY*

FluidInflatorState object

The FluidInflatorState object stores the propagating data for a FluidInflator object.
One instance of this object is created internally by the *FluidInflator* object for each step. The instance is also deleted internally by the *FluidInflator* object.

The FluidInflatorState object has no constructor or methods.

The FluidInflatorState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The FluidInflatorState object has the following member:

status A *SymbolicConstant* specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

FractureCriterion object

The FractureCriterion object specifies fractureCriterion options for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].fractureCriterion
```

FractureCriterion(...)

This method creates a FractureCriterion object.

Path

```
mdb.models[name].interactionProperties[name].FractureCriterion
```

Required arguments

initTable
A sequence of sequences of Floats specifying the value defining the fracture criterion. The items in the table data are described below.

Optional arguments

type
A SymbolicConstant specifying the type of data used to define the fracture criterion. Possible values are VCCT and ENHANCED VCCT. The default value is VCCT.

mixedModeBehavior
A SymbolicConstant specifying the mixed mode behavior type used to define fracture criterion. Possible values are BK, POWER, and REEDER. The default value is BK.

temperatureDependency
A Boolean specifying whether the fracture criterion data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of fracture criterion data field variables. The default value is 0.

tolerance
A Float specifying the tolerance for VCCT\Enhanced VCCT type. The default value is 0.2.

specifyUnstableCrackProp
A SymbolicConstant specifying whether to include unstable crack growth tolerance in fracture criterion. Possible values are ON and OFF. The default value is OFF.

unstableTolerance
The SymbolicConstant DEFAULT or a Float specifying the tolerance for unstable crack propagation. This parameter specified only if *specifyUnstableCrackProp*=ON. The default value is DEFAULT.

Table data

Table data for *initTable*:

If *type*=VCCT for *mixedModeBehavior*=BK or REEDER, the table data specify the following:

- Mode I critical energy release rate, G_{IC}.
- Mode II critical energy release rate, G_{IIC}.
- Mode III critical energy release rate, G_{IIIC}.
- Exponent, η.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=VCCT for *mixedModeBehavior*=POWER, the table data specify the following:

- Mode I critical energy release rate, G_{IC}.
- Mode II critical energy release rate, G_{IIC}.
- Mode III critical energy release rate, G_{IIIC}.
- Exponent, am.
- Exponent, an.
- Exponent, ao.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENHANCED VCCT for *mixedModeBehavior*=BK or REEDER, the table data specify the following:

- Mode I critical energy release rate for onset crack, G_{IC}.
- Mode II critical energy release rate for onset crack, G_{IIC}.
- Mode III critical energy release rate for onset crack, G_{IIIC}.
- Mode I critical energy release rate for crack propagation, G_{IC}.
- Mode II critical energy release rate for crack propagation, G_{IIC}.
- Mode III critical energy release rate for crack propagation, G_{IIIC}.
- Exponent, η.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENHANCED VCCT for *mixedModeBehavior*=POWER, the table data specify the following:

- Mode I critical energy release rate for onset crack, G_{IC}.
- Mode II critical energy release rate for onset crack, G_{IIC}.

- Mode III critical energy release rate for onset crack, G_{IIIC}.
- Mode I critical energy release rate for crack propagation, G_{IC}.
- Mode II critical energy release rate for crack propagation, G_{IIC}.
- Mode III critical energy release rate for crack propagation, G_{IIIC}.
- Exponent, am.
- Exponent, an.
- Exponent, ao.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FractureCriterion object.

Exceptions

None.

setValues(...)

This method modifies the FractureCriterion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FractureCriterion* method.

Return value

`None.`

Exceptions

None.

Members

The FractureCriterion object has members with the same names and descriptions as the arguments to the *FractureCriterion* method.

Corresponding analysis keywords

- **FRACTURE CRITERION*

GapElectricalConductance object

The GapElectricalConductance object specifies electrical conductance for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].electricalConductance
```

ElectricalConductance(...)

This method creates a GapElectricalConductance object.

Path

```
mdb.models[name].interactionProperties[name].ElectricalConductance
```

Required arguments

None.

Optional arguments

definition
A SymbolicConstant specifying how the electrical conductance is defined. Possible values are TABULAR and USER_DEFINED. The default value is TABULAR.

Type
A SymbolicConstant specifying the Type of material. Possible values are SOLID and LIQUID. The default value is SOLID.

clearanceDependency
A Boolean specifying whether to use clearance-dependent data. The default value is ON.

pressureDependency
A Boolean specifying whether to use pressure-dependent data. The default value is OFF.

temperatureDependencyC
A Boolean specifying whether to use temperature-dependent data with clearance dependency. The default value is OFF.

ionConcentrationDependencyC
A Boolean specifying whether to use ion-concentration-dependent data with clearance dependency. The default value is OFF.

dependenciesC
An Int specifying the number of field variables to use with clearance dependency. The default value is 0.

clearanceDepTable
A sequence of sequences of Floats specifying clearance dependency data. The items in the table data are described below.

temperatureDependencyP
A Boolean specifying whether to use temperature-dependent data with pressure dependency. The default value is OFF.

ionConcentrationDependencyP A Boolean specifying whether to use ion-concentration–dependent data with pressure dependency. The default value is OFF.

dependenciesP An Int specifying the number of field variables to use with pressure dependency. The default value is 0.

pressureDepTable A sequence of sequences of Floats specifying pressure dependency data. The items in the table data are described below.

Table data

For *Type* = *SOLID* with The *clearanceDepTable* data specify the following:

- Conductivity.
- Clearance.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

For *Type* = *SOLID* with the *pressureDepTable* data specify the following:

- Conductivity.
- Pressure.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

For *Type* = *LIQUID* with The *clearanceDepTable* data specify the following:

- Conductivity.
- Clearance.
- Temperature, if the data depend on temperature.
- ion Concentration, if the data depend on ion Concentration.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

For *Type* = *LIQUID* with the *pressureDepTable* data specify the following:

- Conductivity.
- Pressure.
- Temperature, if the data depend on temperature.
- ion Concentration, if the data depend on ion Concentration.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

Return value

A GapElectricalConductance object.

Exceptions

None.

`setValues(...)`

This method modifies the GapElectricalConductance object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapElectricalConductance* method.

Return value

`None.`

Exceptions

None.

Members

The GapElectricalConductance object has members with the same names and descriptions as the arguments to the *GapElectricalConductance* method.

Corresponding analysis keywords

- **GAP ELECTRICAL CONDUCTANCE*

GapHeatGeneration object

The GapHeatGeneration object specifies heat generation for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].heatGeneration
```

HeatGeneration(...)

This method creates a GapHeatGeneration object.

Path

```
mdb.models[name].interactionProperties[name].HeatGeneration
```

Required arguments

None.

Optional arguments

conversionFraction
A Float specifying the fraction of dissipated energy caused by friction or electric currents that is converted to heat. The default value is 1.0.

secondaryFraction
A Float specifying the fraction of converted heat distributed to the secondary surface. The default value is 0.5.

mechanicalInteractionFraction
A Float specifying the fraction of dissipated energy converted into heat generated by mechanical interaction.

mechanicalInteractionWeightingFactor
A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by mechanical interaction.

electricalConductionSolidsFraction
A Float specifying the fraction of dissipated energy converted into heat generated by electrical conduction in the solids.

electricalConductionSolidsWeightingFactor
A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by electrical conduction in the solids.

electricalConductionLiquidsFraction
A Float specifying the fraction of dissipated energy converted into heat generated by electrical conduction in the liquids.

electricalConductionLiquidsWeightingFactor
A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by electrical conduction in the liquids.

Return value

A GapHeatGeneration object.

Exceptions

None.

setValues(...)

This method modifies the GapHeatGeneration object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapHeatGeneration* method.

Return value

`None.`

Exceptions

None.

Members

The GapHeatGeneration object has the following members:

conversionFraction

A Float specifying the fraction of dissipated energy caused by friction or electric currents that is converted to heat. The default value is 1.0.

secondaryFraction

A Float specifying the fraction of converted heat distributed to the secondary surface. The default value is 0.5.

mechanicalInteractionFraction

A Float specifying the fraction of dissipated energy converted into heat generated by mechanical interaction.

mechanicalInteractionWeightingFactor

A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by mechanical interaction.

electricalConductionSolidsFraction

A Float specifying the fraction of dissipated energy converted into heat generated by electrical conduction in the solids.

electricalConductionSolidsWeightingFactor

A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by electrical conduction in the solids.

electricalConductionLiquidsFraction

A Float specifying the fraction of dissipated energy converted into heat generated by electrical conduction in the liquids.

electricalConductionLiquidsWeightingFactor A Float specifying the weighting factor for the distribution of heat between the interacting surfaces generated by electrical conduction in the liquids.

Corresponding analysis keywords

- **GAP HEAT GENERATION*

GeometricProperties object

The GeometricProperties object specifies surface interaction properties.

Access

```
import interaction
mdb.models[name].interactionProperties[name].geometricProperties
```

GeometricProperties(...)

This method creates a GeometricProperties object.

Path

```
mdb.models[name].interactionProperties[name].GeometricProperties
```

Required arguments

None.

Optional arguments

contactArea

A Float specifying the out-of-plane thickness of the surface for a two-dimensional model or cross-sectional area for every node in the node-based surface. The default value is 1.0.

padThickness

`None` or a Float specifying the thickness of an interfacial layer between the contacting surfaces. If *padThickness*=`None`, there is no interfacial layer. The default value is `None`.

trackingThickness

`None` or a Float specifying the thickness that determines the contacting surfaces to be tracked. The input value for this parameter cannot be negative. An internal default value is used if a zero value is input or if the parameter is omitted.

dependentVariables

An Int specifying the number of state-dependent variables. The default value is 0. This argument is applicable only if *modelType*=MODELTYPE_USER or *modelType*=MODELTYPE_USER_INTERACTION.

numProperties

An Int specifying the number of property values required. The default value is 0. This argument is applicable only if *modelType*=MODELTYPE_USER or *modelType*=MODELTYPE_USER_INTERACTION.

useUnsymmetricEqunProcedure

A Boolean specifying whether to use unsymmetric equation solution procedures. This argument is applicable only if *modelType*=MODELTYPE_USER or *modelType*=MODELTYPE_USER_INTERACTION.

modelType

A SymbolicConstant specifying the surface interaction model type.

Return value

A GeometricProperties object.

Exceptions

None.

`setValues(...)`

This method modifies the GeometricProperties object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GeometricProperties* method.

Return value

`None.`

Exceptions

None.

Members

The GeometricProperties object has members with the same names and descriptions as the arguments to the *GeometricProperties* method.

Corresponding analysis keywords

- **SURFACE INTERACTION*

IncidentWave object

The IncidentWave object defines incident wave interactions for acoustic and coupled acoustic-structural analyses.

The IncidentWave object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

IncidentWave(...)

This method creates an IncidentWave object.

Path

```
mdb.models[name].IncidentWave
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the IncidentWave object is created.

sourcePoint
A *Region* object specifying the incident wave source point.

standoffPoint
A *Region* object specifying the incident wave standoff point.

This argument is not valid when *definition*=CONWEP.

surface
A *Region* object specifying the surface defining the incident wave interaction. In problems involving fluid/surface boundaries, both the fluid surface and the solid surface comprising the boundary must have an incident wave interaction specified.

interactionProperty
A String specifying the *IncidentWaveProperty* object associated with this interaction.

Optional arguments

definition
A SymbolicConstant specifying the type of incident wave to be defined. The value must be PRESSURE for linear perturbation steps. An Explicit step is required when the value is set to CONWEP. Possible values are PRESSURE, ACCELERATION, UNDEX, and CONWEP. The default value is PRESSURE.

amplitude
A String specifying the name of the *Amplitude* object that defines the fluid pressure time history at the standoff point, if *definition*=PRESSURE. If *definition*=ACCELERATION, then this string specifies the name of the *Amplitude* object that defines the fluid particle acceleration time history at the standoff point. This member can be specified only if *definition*=PRESSURE or ACCELERATION. The default value is an empty string.

imaginaryAmplitude A String specifying the name of the *Amplitude* object that defines the imaginary component of the fluid pressure time history at the standoff point. This member is applicable only for linear perturbation steps and if *definition*=PRESSURE. The default value is an empty string.

surfaceNormal A sequence of three Floats specifying the X, Y, and Z components of the direction cosine of the fluid surface normal.

This argument is valid only when *definition*=UNDEX.

initialDepth `None` or a Float specifying the initial depth of the UNDEX bubble. The default value is `None`.

This argument is valid only when *definition*=UNDEX.

referenceMagnitude A Float specifying the reference magnitude.

This argument is not valid when *definition*=CONWEP.

detonationTime A Float specifying the time of detonation, given in total time.

This argument is valid only when *definition*=CONWEP.

magnitudeFactor A Float specifying the magnitude scale factor. The default value is 1.0.

This argument is valid only when *definition*=CONWEP.

Return value

An IncidentWave object.

Exceptions

None.

`setValues(...)`

This method modifies the IncidentWave object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IncidentWave* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The IncidentWave object has members with the same names and descriptions as the arguments to the *IncidentWave* method.

Corresponding analysis keywords

- **INCIDENT WAVE INTERACTION*

IncidentWaveProperty object

The IncidentWaveProperty object is an interaction property that defines the properties referred to by an *IncidentWave* object.

The IncidentWaveProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

IncidentWaveProperty(...)

This method creates an IncidentWaveProperty object.

Path

```
mdb.models[name].IncidentWaveProperty
```

Required arguments

name
A String specifying the interaction property repository key.

Optional arguments

definition
A SymbolicConstant specifying the type of wave to be defined. Possible values are PLANAR, SPHERICAL, DIFFUSE, AIR_BLAST, and SURFACE_BLAST. The default value is PLANAR.

propagationModel
A SymbolicConstant specifying the spherical propagation model. Possible values are ACOUSTIC, UNDEX_CHARGE, and GENERALIZED_DECAY. The default value is ACOUSTIC.

This argument is valid only when *definition*=SPHERICAL.

soundSpeed
A Float specifying the speed of sound in the fluid.

This argument is not valid when *definition*=AIR_BLAST or when *definition*=SURFACE_BLAST.

fluidDensity
A Float specifying the fluid mass density.

This argument is not valid when *definition*=AIR_BLAST or when *definition*=SURFACE_BLAST.

specificHeatRatio
`None` or a Float specifying the ratio of specific heats for gas. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

gravity
`None` or a Float specifying the acceleration due to gravity. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

atmosphericPressure

`None` or a Float specifying the atmospheric pressure. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

dragCoefficient

`None` or a Float specifying the fluid drag coefficient. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

dragExponent

A Float specifying the fluid drag exponent. The default value is 2.0.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

waveEffects

A Boolean specifying whether or not to include wave effects in the fluid and gas. The default value is ON.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

chargeDensity

`None` or a Float specifying the density of the charge material. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

chargeMass

`None` or a Float specifying the mass of the charge material. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

constantK1

`None` or a Float specifying the charge material constant K. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

constantK2

`None` or a Float specifying the charge material constant k. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

constantA

`None` or a Float specifying the charge material constant A. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

constantB

`None` or a Float specifying the charge material constant B. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

constantKc

`None` or a Float specifying the charge material constant Kc. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

duration

`None` or a Float specifying the time duration for the bubble simulation. The default value is `None`.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

maximumSteps

An Int specifying the maximum number of time steps for the bubble simulation. The default value is 1500.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

relativeStepControl

A Float specifying the relative step size control parameter. The default value is 1×10^{-11}.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

absoluteStepControl

A Float specifying the absolute step size control parameter. The default value is 1×10^{-11}.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

stepControlExponent

A Float specifying the step size control exponent. The default value is 0.2.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=UNDEX_CHARGE.

genDecayA

A Float specifying the constant A associated with the generalized decay propagation model. The default value is 0.0.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=GENERALIZED_DECAY.

genDecayB

A Float specifying the constant B associated with the generalized decay propagation model. The default value is 0.0.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=GENERALIZED_DECAY.

genDecayC

A Float specifying the constant C associated with the generalized decay propagation model. The default value is 0.0.

This argument is valid only when *definition*=SPHERICAL and *propagationModel*=GENERALIZED_DECAY.

seedNumber

An Int specifying the seed number (N) for the diffuse source calculation. N^2 sources will be used in the simulation.

This argument is valid only when *definition*=DIFFUSE.

massTNT

A Float specifying the equivalent mass of TNT, in any preferred mass unit.

This argument is valid only when *definition*=AIR_BLAST or *definition*=SURFACE_BLAST.

massFactor

A Float specifying the multiplication factor to convert from the preferred mass unit to kilograms. The default value is 1.0.

This argument is valid only when *definition*=AIR_BLAST or *definition*=SURFACE_BLAST.

lengthFactor

A Float specifying the multiplication factor to convert from the analysis length unit to meters. The default value is 1.0.

This argument is valid only when *definition*=AIR_BLAST or *definition*=SURFACE_BLAST.

timeFactor

A Float specifying the multiplication factor to convert from the analysis time unit to seconds. The default value is 1.0.

This argument is valid only when *definition*=AIR_BLAST or *definition*=SURFACE_BLAST.

pressureFactor

A Float specifying the multiplication factor to convert from the analysis pressure unit to pascals. The default value is 1.0.

This argument is valid only when *definition*=AIR_BLAST or *definition*=SURFACE_BLAST.

Return value

An IncidentWaveProperty object.

Exceptions

None.

`setValues(...)`

This method modifies the IncidentWaveProperty object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IncidentWaveProperty* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The IncidentWaveProperty object has members with the same names and descriptions as the arguments to the *IncidentWaveProperty* method.

Corresponding analysis keywords

- **INCIDENT WAVE INTERACTION PROPERTY*
- **UNDEX CHARGE PROPERTY*
- **CONWEP CHARGE PROPERTY*

IncidentWaveState object

The IncidentWaveState object stores the propagating data of an *IncidentWave* object in a step. One instance of this object is created internally by the *IncidentWave* object for each step. The instance is also deleted internally by the *IncidentWave* object.

The IncidentWaveState object has no constructor or methods.

The IncidentWaveState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The IncidentWaveState object has the following member:

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **INCIDENT WAVE INTERACTION*

InitializationAssignment object

The InitializationAssignment object stores the contact initialization assignment definition for domain pairs in a *ContactStd* or *ContactExp* object. The InitializationAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].initializationAssignments
```

changeValuesInStep(...)

This method allows modification of contact initialization assignments to domain pairs already defined in a given step.

Required arguments

stepName A String specifying the name of the step in which the contact initialization assignments are to be modified.

index An Int specifying the position of the contact initialization assignment whose value is to be modified.

value A String specifying the value of the contact initialization to be assigned to the domain pair whose index is referenced.

Optional arguments

None.

Return value

None.

Exceptions

None.

appendInStep(...)

This method allows addition of contact initialization assignments to new domain pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which new contact initialization assignments are to be defined.

assignments A sequence of tuples specifying the initializations assigned to each surface pair. Each tuple contains four entries (fourth entry is for Abaqus/Explicit and is optional):

- A region object or the SymbolicConstant GLOBAL (for Abaqus/Standard only).
- A region object or the SymbolicConstant SELF (for Abaqus/Standard only).

- A String specifying a *StdInitialization* or *ExpInitialization*object associated with this pair of regions.
- A String specifying a secondary surface type. This entry is applicable only if the *ExpInitialization* object is defined with *overclosureType*=CLEARANCE and *adjustNodalCoords*=True.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing contact initialization assignments from a *ContactStd* or *ContactExp* object.

Required arguments

indices A sequence of Ints specifying the index of each contact initialization assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The InitializationAssignment object has no members.

Corresponding analysis keywords

- **CONTACT INITIALIZATION ASSIGNMENT*

InteractionProperty object

The InteractionProperty object is the abstract base type for other InteractionProperty objects. The InteractionProperty object has no explicit constructor, members, or methods.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

Members

The InteractionProperty object has no members.

InteractionState object

The InteractionState object is the abstract base type for other InteractionState objects. The InteractionState object has no explicit constructor. The members of the InteractionState object are common to all objects derived from InteractionState.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The InteractionState object has the following member:

status

A SymbolicConstant specifying the propagation state of the InteractionState object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

MainSecondaryAssignment object

The MainSecondaryAssignment object stores the main-secondary assignment definition for surfaces in *ContactExp* and *ContactStd* objects. The MainSecondaryAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].mainSecondaryAssignments
```

changeValuesInStep(...)

This method allows modification of main-secondary assignments already defined on surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the main-secondary assignments are to be modified.

index An Int specifying the position of the main-secondary assignment whose value is to be modified.

value A SymbolicConstant specifying the value of the main-secondary role to be assigned to the surface whose index is referenced. Possible values are MAIN, SECONDARY, and BALANCED. The SymbolicConstant BALANCED can be specified only in an Abaqus/Standard analysis.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of main-secondary assignments to new surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the main-secondary assignments are to be defined.

assignments A sequence of tuples specifying the main-secondary assignments. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the main-secondary attribute is assigned.

- A SymbolicConstant specifying the overriding main-secondary value to be used for the first surface. Possible values of the SymbolicConstant are MAIN, SECONDARY, and BALANCED. The SymbolicConstant BALANCED can be specified only in an Abaqus/Standard analysis.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing main-secondary assignments.

Required arguments

indices A sequence of Ints specifying the index of each main-secondary assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The MainSecondaryAssignment object has no members.

Corresponding analysis keywords

- **CONTACT FORMULATION*, TYPE=PURE MAIN-SECONDARY
- **CONTACT FORMULATION*, TYPE=MAIN SECONDARY ROLES

Model object

The following commands operate on Model objects. For more information about the Model object, see *Model object*.

Access

```
import interaction
```

`contactDetection(...)`

This method uses contact detection to create *SurfaceToSurfaceContactStd*, *SurfaceToSurfaceContactExp*, and *Tie* objects.

Required arguments

None.

Optional arguments

name

A String specifying the prefix used to generate repository keys. The default value is "CP-"

createStepName

A String specifying the name of the step in which the *SurfaceToSurfaceContactStd*, *SurfaceToSurfaceContactExp*, and *Tie* objects are created. The default value is "Initial."

searchDomain

A SymbolicConstant MODEL or a sequence of Strings specifying the names of instances to search. MODEL indicates the whole model is searched. The default value is MODEL.

defaultType

A SymbolicConstant specifying the default type of object to create. Possible values are CONTACT, CONTACT_STANDARD, CONTACT_EXPLICIT, and TIE. If CONTACT is used, the behavior is determined by the type of *Step* in the model. If an *ExplicitDynamicsStep* or *TempDisplacementDynamicsStep* exists, then *SurfaceToSurfaceContactExp* is created by default. Otherwise *SurfaceToSurfaceContactStd* is created by default. The default value is CONTACT.

interactionProperty

A String specifying the name of the *ContactProperty* object associated with any interactions created.

separationTolerance

A Float specifying the maximum separation for considering two surfaces to be candidates for contact, where separation is the maximum distance between the points of closest approach on the two surfaces. The default value is a function of the model.

extendByAngle

None or a Float specifying the angle for extending surface definitions to include adjacent faces. The default value is 20.

mergeWithinAngle
None or a Float specifying the angle for merging adjacent contact pairs that lie within the angle. The default value is 20.

searchSingleInstances
A Boolean specifying whether to include surface pairs within a single instance. The default value is OFF.

nameEachSurfaceFound
A Boolean specifying whether to assign a name to each surface found. The default value is ON.

createUnionOfMainSurfaces
A Boolean specifying whether to create a surface that is the union of all main surfaces found. The default value is OFF.

createUnionOfSecondarySurfaces
A Boolean specifying whether to create a surface that is the union of all secondary surfaces found. The default value is OFF.

createUnionOfMainSecondarySurfaces
A Boolean specifying whether to create a surface that is the union of all main and secondary surfaces found. The default value is OFF.

includePlanar
A Boolean specifying whether to include planar geometry. The default value is ON.

includeCylindricalSphericalToric
A Boolean specifying whether to include cylindrical, spherical, and toric geometry. The default value is ON.

includeSplineBased
A Boolean specifying whether to include spline-based geometry. The default value is ON.

includeMeshSolid
A Boolean specifying whether to include solid mesh entities. The default value is ON.

includeMeshShell
A Boolean specifying whether to include shell mesh entities. The default value is ON.

includeMeshMembrane
A Boolean specifying whether to include mesh membrane entities. The default value is OFF.

includeOverclosed
A Boolean specifying whether to include overclosed pairs. The default value is ON.

includeNonOverlapping
A Boolean specifying whether to include opposing geometry surfaces that do not overlap. The default value is OFF.

meshedGeometrySearchTechnique
A SymbolicConstant USE_GEOMETRY or USE_MESH specifying whether to locate pairs in meshed geometry using the geometric entities or mesh entities. The default value is USE_GEOMETRY.

useShellThickness
A Boolean specifying whether to account for shell thickness and offset during contact detection. The default value is ON.

surfaceSmoothing A SymbolicConstant specifying whether to use surface smoothing for geometric surfaces in *SurfaceToSurfaceContactStd* interactions. Possible values are NONE and AUTOMATIC. The default value isAUTOMATIC.

Return value

None

Exceptions

None.

`getSurfaceSeparation(...)`

This method returns a list of all possible contacts that can be created using the *ContactDetection* method.

Required arguments

None.

Optional arguments

The optional arguments to `getSurfaceSeparation` are the same as the arguments to the *ContactDetection* method, with the exception of the *createStepName*, *defaultType*, *interactionProperty*, and *nameEachSurfaceFound* arguments.

Return value

Tuple of tuples, where each tuple holds information, to be used in contact creation as follows:

A string specifying the name of the main surface used in contact.

A string specifying the name of the secondary surface used in contact.

A float specifying the separation distance between the main surface and the secondary surface.

A boolean specifying whether or not contact surfaces are overclosed.

Exceptions

None.

ModelChange object

The ModelChange object defines model change interactions for element removal and reactivation.

The ModelChange object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

ModelChange(...)

This method creates a ModelChange object.

Path

```
mdb.models[name].ModelChange
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the ModelChange object is created.

Optional arguments

isRestart
A Boolean specifying whether this interaction is being used solely to indicate that model change may be required in a subsequent restart analysis (either for elements or contact pairs). The default value is OFF.

regionType
A SymbolicConstant specifying the region selection type. This argument is valid only when *isRestart*=False. Possible values are GEOMETRY, SKINS, STRINGERS, and ELEMENTS. The default value is GEOMETRY.

region
A *Region* object specifying the elements to be removed or reactivated. This argument is valid only when *isRestart*=False.

activeInStep
A Boolean specifying whether elements are being removed or reactivated. This argument is valid only when *isRestart*=False. The default value is OFF.

includeStrain
A Boolean specifying whether stress/displacement elements are reactivated with strain. This argument is valid only when *isRestart*=False and when *activeInStep*=True. The default value is OFF.

Return value

A ModelChange object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ModelChange object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ModelChange* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data of an existing ModelChange object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

activeInStep A Boolean specifying whether elements are being removed or reactivated. This argument is valid only when *isRestart*=False. The default value is OFF.

includeStrain A Boolean specifying whether stress/displacement elements are reactivated with strain. This argument is valid only when *isRestart*=False and when *activeInStep*=True. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The ModelChange object has members with the same names and descriptions as the arguments to the *ModelChange* method.

Corresponding analysis keywords

- **MODEL CHANGE*, TYPE=ELEMENT
- **MODEL CHANGE*, ACTIVATE

NormalBehavior object

The NormalBehavior object specifies normal behavior for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].normalBehavior
```

NormalBehavior(...)

This method creates a NormalBehavior object.

Path

```
mdb.models[name].interactionProperties[name].NormalBehavior
```

Required arguments

None.

Optional arguments

contactStiffness

The SymbolicConstant DEFAULT or a Float specifying the contact stiffness. This argument is valid for *pressureOverclosure*=LINEAR. This argument is also valid for *pressureOverclosure*=HARD when *constraintEnforcementMethod*=AUGMENTED_LAGRANGE or PENALTY. A value of DEFAULT is valid only when the later conditions are met. A value of zero is equivalent to specifying DEFAULT. The default value is DEFAULT.

pressureOverclosure

A SymbolicConstant specifying the pressure-overclosure relationship to be used. Possible values are HARD, EXPONENTIAL, LINEAR, TABULAR, and SCALE_FACTOR. The default value is HARD.

allowSeparation

A Boolean specifying whether to allow separation after contact. The default value is ON.

maxStiffness

`None` or a Float specifying the maximum stiffness. If *maxStiffness*=`None`, there is no upper limit. The default value is `None`.

table

A sequence of sequences of Floats specifying the normal behavior properties. This argument is valid only for *pressureOverclosure*=EXPONENTIAL or TABULAR. The items in the table data are described below.

constraintEnforcementMethod

A SymbolicConstant specifying the method for enforcement of the contact constraint. Possible values are DEFAULT, AUGMENTED_LAGRANGE, PENALTY, and DIRECT. The default value is DEFAULT.

overclosureFactor

A Float specifying the overclosure measure (used to delineate the segments of the pressure-overclosure curve) as a percentage of the minimum element size in the contact region. The default value is 0.0.

overclosureMeasure
A Float specifying the overclosure measure (used to delineate the segments of the pressure-overclosure curve) directly. The default value is 0.0.

contactStiffnessScaleFactor
A Float specifying scale factor for the penalty stiffness or the geometric scaling of the "base" stiffness. The default value is 1.0.

initialStiffnessScaleFactor
A Float specifying an additional scale factor for the "base" default contact stiffness. The default value is 1.0.

clearanceAtZeroContactPressure
A Float specifying the clearance at which the contact pressure is zero. The default value is 0.0.

stiffnessBehavior
A SymbolicConstant specifying the type of penalty stiffness to be defined. This argument is valid only when *constraintEnforcementMethod*=PENALTY. Possible values are LINEAR and NONLINEAR. The default value is LINEAR.

stiffnessRatio
A Float specifying the ratio of the initial stiffness divided by the final stiffness. This argument is valid only when *stiffnessBehavior*=NONLINEAR. Possible values are $0 \leq stiffnessRatio < 1$. The default value is 0.01.

upperQuadraticFactor
A Float specifying the ratio of the overclosure at the maximum stiffness divided by the characteristic facet length. This argument is valid only when *stiffnessBehavior*=NONLINEAR. The default value is 0.03.

lowerQuadraticRatio
A Float specifying the ratio of the overclosure at the initial stiffness divided by the overclosure at the maximum stiffness, both relative to the clearance at which the contact pressure is zero. This argument is valid only when *stiffnessBehavior*=NONLINEAR. Possible values are $0 \leq stiffnessRatio < 1$. The default value is 0.33333.

Table data

If *pressureOverclosure*=EXPONENTIAL, the table data specify the following:

- Pressure at zero clearance, p_0.
- Clearance at which the contact pressure is zero, c_0.

If *pressureOverclosure*=TABULAR, the table data specify the following:

- Pressure.
- Overclosure.

Return value

A NormalBehavior object.

Exceptions

None.

`setValues(...)`

This method modifies the NormalBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *NormalBehavior* method.

Return value

`None.`

Exceptions

None.

Members

The NormalBehavior object has members with the same names and descriptions as the arguments to the *NormalBehavior* method.

Corresponding analysis keywords

- **SURFACE BEHAVIOR*

PolarityAssignments object

The PolarityAssignments object stores the polarity assignment definition for surfaces in *ContactExp* objects. The PolarityAssignments object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].polarityAssignments
```

changeValuesInStep(...)

This method allows you to modify polarity assignments already defined on surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the main-secondary assignments are to be modified.

index An Int specifying the position of the polarity assignment whose value is to be modified.

value A SymbolicConstant specifying the value of the polarity to be assigned to the surface whose index is referenced. Possible values are SPOS, SNEG, and TWO_SIDED.

Optional arguments

None.

Return value

None.

Exceptions

None.

appendInStep(...)

This method allows you to add polarity assignments to new surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the polarity assignments are to be defined.

assignments A sequence of tuples specifying the polarity assignments. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the polarity attribute is assigned.
- A SymbolicConstant specifying the overriding polarity value to be used for the first surface. Possible values of the SymbolicConstant are SPOS, SNEG, and TWO_SIDED.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing polarity assignments.

Required arguments

indices A sequence of Ints specifying the index of each polarity assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The PolarityAssignments object has no members.

Corresponding analysis keywords

- **CONTACT FORMULATION*, TYPE=POLARITY

PressurePenetration object

The PressurePenetration object defines pressure penetration loading simulated with surface-to-surface contact.

The PressurePenetration object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

PressurePenetration(...)

This method creates a PressurePenetration object.

Path

```
mdb.models[name].PressurePenetration
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the PressurePenetration object is created.

contactInteraction
A String specifying the name of the Surface-to-surface contact (Standard) interaction.

mainPoints
A *RegionArray* object specifying the points on the main surface that are exposed to the fluid.

secondaryPoints
A *RegionArray* object specifying the points on the secondary surface that are exposed to the fluid.

penetrationPressure
A tuple of Floats specifying the fluid pressure magnitude. For steady state dynamic analyses, a tuple of Complexes specifying the fluid pressure magnitude.

criticalPressure
A tuple of Floats specifying the critical contact pressure below which fluid penetration starts to occur.

Optional arguments

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

penetrationTime
A Float specifying the fraction of the current step time over which the fluid pressure on newly penetrated contact surface segments is ramped up to the current magnitude. The default value is 0.001.

Return value

A PressurePenetration object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing PressurePenetration object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PressurePenetration* method, except for the *name* and *createStepName* arguments.

Return value

```
None.
```

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing PressurePenetration object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

penetrationPressure A tuple of Floats specifying the fluid pressure magnitude. For steady state dynamic analyses, a tuple of Complexes specifying the fluid pressure magnitude.

criticalPressure A tuple of Floats specifying the critical contact pressure below which fluid penetration starts to occur.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

penetrationTime A Float specifying the fraction of the current step time over which the fluid pressure on newly penetrated contact surface segments is ramped up to the current magnitude. The default value is 0.001.

Return value

`None.`

Exceptions

None.

Members

The PressurePenetration object has the following members:

name A String specifying the repository key.

createStepName A String specifying the name of the step in which the PressurePenetration object is created.

contactInteraction A String specifying the name of the Surface-to-surface contact (Standard) interaction.

mainPoints A *RegionArray* object specifying the points on the main surface that are exposed to the fluid.

secondaryPoints A *RegionArray* object specifying the points on the secondary surface that are exposed to the fluid.

Corresponding analysis keywords

- **PRESSURE PENETRATION*

PressurePenetrationState object

The PressurePenetrationState object stores the propagating data of a *PressurePenetration* object in a step. One instance of this object is created internally by the *PressurePenetration* object for each step. The instance is also deleted internally by the *PressurePenetration* object.

The PressurePenetrationState object has no constructor or methods.

The PressurePenetrationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The PressurePenetrationState object has the following members:

penetrationTime

A Float specifying the fraction of the current step time over which the fluid pressure on newly penetrated contact surface segments is ramped up to the current magnitude. The default value is 10^{-3}.

penetrationTimeState

A SymbolicConstant specifying the propagation state of the *penetrationTime* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

criticalPressureState

A SymbolicConstant specifying the propagation state of the *criticalPressure* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

penetrationPressureState

A SymbolicConstant specifying the propagation state of the *penetrationPressure* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

penetrationPressure

A tuple of Floats specifying the fluid pressure magnitude. For steady state dynamic analyses, a tuple of Complexes specifying the fluid pressure magnitude.

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

criticalPressure

A tuple of Floats specifying the critical contact pressure below which fluid penetration starts to occur.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED

- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **PRESSURE PENETRATION*

Radiation object

The Radiation object specifies radiation for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].radiation
```

Radiation(...)

This method creates a Radiation object.

Path

```
mdb.models[name].interactionProperties[name].Radiation
```

Required arguments

mainEmissivity

A Float specifying the emissivity of the main surface.

secondaryEmissivity

A Float specifying the emissivity of the secondary surface.

table

A sequence of sequences of Floats specifying the following:

- Effective viewfactor, F.
- Gap clearance, d.

Optional arguments

None.

Return value

A Radiation object.

Exceptions

None.

`setValues(...)`

This method modifies the Radiation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Radiation* method.

Return value

`None.`

Exceptions

None.

Members

The Radiation object has members with the same names and descriptions as the arguments to the *Radiation* method.

Corresponding analysis keywords

- **GAP RADIATION*

RadiationToAmbient object

The RadiationToAmbient object defines radiant heat transfer between a surface and its environment.

The RadiationToAmbient object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

RadiationToAmbient(...)

This method creates a RadiationToAmbient object.

Path

```
mdb.models[name].RadiationToAmbient
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the RadiationToAmbient object is created.

region
A *Region* object specifying the surface to which the radiation interaction is applied or an Element set object specifying the elements to which the element-based radiation interaction is applied.

emissivity
A Float specifying the emissivity, ϵ.

Optional arguments

field
A String specifying the name of the *AnalyticalField* object associated with this interaction. The *field* argument applies only when *distributionType*=ANALYTICAL_FIELD. The default value is an empty string.

distributionType
A SymbolicConstant specifying how the radiation is distributed. This argument applies only when *radiationType*=AMBIENT. Possible values are UNIFORM and ANALYTICAL_FIELD. The default value is UNIFORM.

radiationType
A SymbolicConstant specifying whether to use the default surface radiation behavior, or the cavity radiation approximation. Possible values are AMBIENT and CAVITY. The default value is AMBIENT.

ambientTemperature
A Float specifying the reference ambient temperature, θ^0. This argument applies only when *radiationType*=AMBIENT. The default value is 0.0.

ambientTemperatureAmp A String specifying the name of the *Amplitude* object that gives the variation of the ambient temperature with time.



Note: Use `None` in an Abaqus/Standard analysis to specify that the reference ambient temperature is applied immediately at the beginning of the step or linearly over the step. Use `None` in an Abaqus/Explicit analysis to specify that the reference ambient temperature is applied throughout the step. This argument applies only when *radiationType*=AMBIENT.

dynamicSurfaceActivation A Boolean specifying the activation of element-based radiation interaction on evolving faces of an element.

Return value

A RadiationToAmbient object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing RadiationToAmbient object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RadiationToAmbient* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data of an existing RadiationToAmbient object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

The optional arguments to `setValuesInStep` are *emissivity*, *ambientTemperature*, and *ambientTemperatureAmp* arguments.

Return value

`None.`

Exceptions

None.

Members

The RadiationToAmbient object has members with the same names and descriptions as the arguments to the *RadiationToAmbient* method except the optional arguments to the *setValuesInStep* method.

RadiationToAmbientState object

The RadiationToAmbientState object stores the propagating data for a *RadiationToAmbient* object. One instance of this object is created internally by the *RadiationToAmbient* object for each step. The instance is also deleted internally by the *RadiationToAmbient* object.

The RadiationToAmbientState object has no constructor or methods.

The RadiationToAmbientState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The RadiationToAmbientState object has the following members:

ambientTemperature

A Float specifying the ambient temperature.

ambientTemperatureState

A SymbolicConstant specifying the propagation state of the *ambientTemperature* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

ambientTemperatureAmpState

A SymbolicConstant specifying the propagation state of the *ambientTemperatureAmp* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

emissivity

A Float specifying the emissivity.

emissivityState

A SymbolicConstant specifying the propagation state of the *emissivity* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

ambientTemperatureAmp

A String specifying the name of the Amplitude object that gives the variation of the ambient temperature with time.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **SRADIATE*

RegionPairs object

The RegionPairs object stores the domain pair definition for *ContactExp* and *ContactStd* objects. The RegionPairs object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].excludedPairs
mdb.models[name].interactions[name].includedPairs
```

setValuesInStep(...)

This method allows addition and removal of domain pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the region pair assignments are to be modified.

Optional arguments

useAllstar A Boolean specifying whether the contacting surface pair consists of all exterior faces and -- in an Abaqus/Explicit analysis -- analytical rigid surfaces, shell edges, and beam segments in the model.

addPairs A sequence of pairs of region objects or SymbolicConstants that specify the surface pairs to add to the included pairs of the *ContactExp* or *ContactStd* object in the given step. Possible values of the SymbolicConstants are GLOBAL and SELF. When used with a *ContactExp* object, the second parameter of each pair can also be a string that references an Eulerian material surface.

removePairs A sequence of pairs of region objects or SymbolicConstants that specify the surface pairs to remove from the included pairs of the *ContactExp* or *ContactStd* object in the given step. Possible values of the SymbolicConstants are GLOBAL and SELF. When used with a *ContactExp* object, the second parameter of each pair can also be a string that references an Eulerian material surface.

Return value

`None.`

Exceptions

None.

Members

The RegionPairs object has no members.

Corresponding analysis keywords

- **CONTACT INCLUSIONS*
- **CONTACT EXCLUSIONS*

SelfContactExp object

The SelfContactExp object defines self-contact during an Abaqus/Explicit analysis.

The SelfContactExp object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

SelfContactExp(...)

This method creates a SelfContactExp object.

Path

```
mdb.models[name].SelfContactExp
```

Required arguments

name	A String specifying the repository key.
createStepName	A String specifying the name of the step in which the SelfContactExp object is created.
surface	A *Region* object specifying the surface where self-contact is defined.
interactionProperty	A String specifying the name of the *ContactProperty* object associated with this interaction.

Optional arguments

mechanicalConstraint	A SymbolicConstant specifying the mechanical constraint formulation. Possible values are KINEMATIC and PENALTY. The default value is KINEMATIC.
contactControls	A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

A SelfContactExp object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SelfContactExp object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SelfContactExp* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing SelfContactExp object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

interactionProperty A String specifying the name of the *ContactProperty* object associated with this interaction.

contactControls A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

`None.`

Exceptions

None.

Members

The SelfContactExp object has members with the same names and descriptions as the arguments to the *SelfContactExp* method except the optional arguments to the *setValuesInStep* method.

SelfContactExpState object

The SelfContactExpState object stores the propagating data for a *SelfContactExp* object. One instance of this object is created internally by the *SelfContactExp* object for each step. The instance is also deleted internally by the *SelfContactExp* object.

The SelfContactExpState object has no constructor or methods.

The SelfContactExpState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The SelfContactExpState object has the following members:

interactionPropertyState

A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

contactControlsState

A SymbolicConstant specifying the propagation state of the *contactControls* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty

A String specifying the name of the *ContactProperty* object associated with this interaction.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CONTACT CONTROLS*
- **CONTACT PAIR*

- **MODEL CHANGE*, TYPE=CONTACT PAIR

SelfContactStd object

The SelfContactStd object defines self-contact during an Abaqus/Standard analysis.

The SelfContactStd object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

SelfContactStd(...)

This method creates a SelfContactStd object.

Path

```
mdb.models[name].SelfContactStd
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the SelfContactStd object is created.

surface
A *Region* object specifying the surface where self-contact is defined.

interactionProperty
A String specifying the name of the *ContactProperty* object associated with this interaction.

Optional arguments

enforcement
A SymbolicConstant specifying the discretization method. Possible values are NODE_TO_SURFACE and SURFACE_TO_SURFACE. The default value is SURFACE_TO_SURFACE.

thickness
A Boolean specifying whether shell/membrane element thickness is considered. The default value is ON.

This argument in valid only when *enforcement*=SURFACE_TO_SURFACE.

smooth
A Float specifying the degree of smoothing used for deformable or rigid main surfaces involved when *enforcement*=NODE_TO_SURFACE. The value given must lie between 0.0 and 0.5. The default value is 0.2.

contactControls
A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

A SelfContactStd object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SelfContactStd object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SelfContactStd* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data of an existing SelfContactStd object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

interactionProperty A String specifying the name of the *ContactProperty* object associated with this interaction.

contactControls A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

`None.`

Exceptions

None.

Members

The SelfContactStd object has members with the same names and descriptions as the arguments to the *SelfContactStd* method except the optional arguments to the *setValuesInStep* method.

In addition, the SelfContactStd object has the following members:

contactTracking

A SymbolicConstant specifying the choice of contact tracking algorithm. The STATE tracking algorithm uses only normal projections and is specified by using ONE_CONFIG. The PATH tracking algorithm uses crossing and normal projections and is specified by using TWO_CONFIG. Possible values are ONE_CONFIG and TWO_CONFIG. The default value is TWO_CONFIG.

This argument is valid only when *enforcement*=SURFACE_TO_SURFACE.

supplementaryContact

A SymbolicConstant specifying the manner in which mid-face constraints are employed. Possible values are SELECTIVE, NEVER, and ALWAYS. The default value is SELECTIVE.

This argument is valid only when *enforcement*=NODE_TO_SURFACE.

SelfContactStdState object

The SelfContactStdState object stores the propagating data for a *SelfContactStd* object. One instance of this object is created internally by the *SelfContactStd* object for each step. The instance is also deleted internally by the *SelfContactStd* object.

The SelfContactStdState object has no constructor or methods.

The SelfContactStdState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The SelfContactStdState object has the following members:

interactionPropertyState

A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

contactControlsState

A SymbolicConstant specifying the propagation state of the *contactControls* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty

A String specifying the *ContactProperty* object associated with this interaction.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CONTACT CONTROLS*
- **CONTACT PAIR*
- **MODEL CHANGE*, TYPE=CONTACT PAIR

SlidingFormulationAssignment object

The SlidingFormulationAssignment object stores the sliding formulation assignment definition for surfaces in *ContactStd* objects. The SlidingFormulationAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].slidingFormulationAssignments
```

changeValuesInStep(...)

This method allows you to modify sliding formulation assignments already defined on surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the sliding formulation assignments are to be modified.

index An Int specifying the position of the sliding formulation assignment whose value is to be modified.

value A SymbolicConstant specifying the value of the smoothness of the surface-to-surface formulation on sliding to be assigned to the surface whose index is referenced. Possible values are NONE and SMALL_SLIDING.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows you to add sliding formulation assignments to new surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the sliding formulation assignments are to be defined.

assignments A sequence of tuples specifying the sliding formulation assignments. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the sliding formulation attribute is assigned.

- A SymbolicConstant specifying the overriding the smoothness value to be used for the first surface. Possible values of the SymbolicConstant are NONE and SMALL_SLIDING.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

delete(...)

The `delete` method allows you to delete existing sliding formulation assignments.

Required arguments

indices A sequence of Ints specifying the index of each sliding formulation assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SlidingFormulationAssignment object has no members.

Corresponding analysis keywords

- **CONTACT FORMULATION*, TYPE=SLIDING FORMULATION

slidingTransitionAssignments object

The SlidingTransitionAssignment object stores the sliding transition assignment definition for surfaces in *ContactStd* objects. The SlidingTransitionAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].slidingTransitionAssignments
```

changeValuesInStep(...)

This method allows you to modify sliding transition assignments already defined on surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the sliding transition assignments are to be modified.

index An Int specifying the position of the sliding transition assignment whose value is to be modified.

value A SymbolicConstant specifying the value of the smoothness of the surface-to-surface formulation on sliding to be assigned to the surface whose index is referenced. Possible values are ELEMENT_ORDER_SMOOTHING, LINEAR_SMOOTHING, and QUADRATIC_SMOOTHING.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows you to add sliding transition assignments to new surface pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which the sliding transition assignments are to be defined.

assignments A sequence of tuples specifying the sliding transition assignments. Each tuple contains two entries:

- A region object or the SymbolicConstant GLOBAL specifying the surface to which the sliding transition attribute is assigned.

- A SymbolicConstant specifying the overriding the smoothness value to be used for the first surface. Possible values of the SymbolicConstant are ELEMENT_ORDER_SMOOTHING, LINEAR_SMOOTHING, and QUADRATIC_SMOOTHING.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing sliding transition assignments.

Required arguments

indices A sequence of Ints specifying the index of each sliding transition assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SlidingTransitionAssignment object has no members.

Corresponding analysis keywords

- **CONTACT FORMULATION*, TYPE=SLIDING TRANSITION

SmoothingAssignment object

The SmoothingAssignment object stores the surface smoothing assignment definition for surfaces in *ContactExp* and *ContactStd* objects. The SmoothingAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].smoothingAssignments
```

changeValuesInStep(...)

This method allows modification of surface smoothing assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface smoothing assignments are to be modified.

index An Int specifying the position of the surface smoothing assignment whose value is to be modified.

value A tuple specifying the value of the surface smoothing assignments for the surface whose index is referenced. Each tuple contains one entry:

- A SymbolicConstant specifying the surface smoothing value to be used for the surface. Possible values of the SymbolicConstant are NONE, REVOLUTION, SPHERICAL, and TOROIDAL.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface smoothing assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface smoothing assignments are to be defined.

assignments A sequence of tuples specifying the surface smoothing assignments. Each tuple contains two entries:

- A region object specifying the surface to which the smoothing is assigned.
- A SymbolicConstant specifying the surface smoothing value to be used for the surface. Possible values of the SymbolicConstant are NONE, REVOLUTION, SPHERICAL, and TOROIDAL.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface smoothing assignments from *ContactExp* and *ContactStd* objects.

Required arguments

indices A sequence of Ints specifying the index of each surface smoothing assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SmoothingAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=GEOMETRIC CORRECTION

StabilizationAssignment object

The StabilizationAssignment object stores the contact stabilization assignment definition for domain pairs in a *ContactStd* object. The StabilizationAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].stabilizationAssignments
```

changeValuesInStep(...)

This method allows modification of contact stabilization assignments to domain pairs already defined in a given step.

Required arguments

stepName A String specifying the name of the step in which the contact stabilization assignments are to be modified.

index An Int specifying the position of the contact stabilization assignment whose value is to be modified.

value A String specifying the value of the contact stabilization to be assigned to the domain pair whose index is referenced.

Optional arguments

None.

Return value

None.

Exceptions

None.

appendInStep(...)

This method allows addition of contact stabilization assignments to new domain pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which new contact stabilization assignments are to be defined.

assignments A sequence of tuples specifying the stabilizations assigned to each surface pair. Each tuple contains three entries:

- A region object or the SymbolicConstant GLOBAL.
- A region object or the SymbolicConstant SELF.

- A String specifying a *StdStabilization* object associated with this pair of regions.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing contact stabilization assignments from a *ContactStd* object.

Required arguments

indices A sequence of Ints specifying the index of each contact stabilization assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The StabilizationAssignment object has no members.

Corresponding analysis keywords

- **CONTACT STABILIZATION*

StdContactControl object

The StdContactControl object is used in Abaqus/Standard analyses to specify optional solution controls for problems involving contact between bodies.

The StdContactControl object is derived from the *ContactControl* object.

Access

```
import interaction
mdb.models[name].contactControls[name]
```

StdContactControl(...)

This method creates an StdContactControl object.

Path

```
mdb.models[name].StdContactControl
```

Required arguments

name
A String specifying the contact controls repository key.

Optional arguments

stiffnessScaleFactor
A Float specifying the factor by which Abaqus/Standard will scale the default penalty stiffness to obtain the stiffnesses used for the contact pairs. Only contact interactions defined with augmented Lagrangian surface behavior will be affected by this argument. The default value is 1.0.

penetrationTolChoice
A SymbolicConstant specifying whether the allowable penetration is an absolute value or a value relative to the characteristic contact surface face dimension. Only contact interactions defined with augmented Lagrangian surface behavior will be affected by this argument. Possible values are RELATIVE and ABSOLUTE. The default value is RELATIVE.

relativePenetrationTolerance
A Float specifying the ratio of the allowable penetration to the characteristic contact surface face dimension. The float values represent percentages (e.g.: 0.001=0.1%). Only contact interactions defined with augmented Lagrangian surface behavior will be affected by this argument. The default value is 10^{-3}.

The *relativePenetrationTolerance* argument applies only when *penetrationTolChoice*=RELATIVE. The *relativePenetrationTolerance* and *absolutePenetrationTolerance* arguments are mutually exclusive.

absolutePenetrationTolerance
`None` or a Float specifying the allowable penetration. Only contact interactions defined with augmented Lagrangian surface behavior will be affected by this argument. The *absolutePenetrationTolerance* argument applies only when *penetrationTolChoice*=ABSOLUTE. The *relativePenetrationTolerance* and *absolutePenetrationTolerance* arguments are mutually exclusive. The default value is `None`.

frictionOnset

A SymbolicConstant specifying when the application of friction occurs. Possible values are:

- IMMEDIATE, specifying the friction is included in the increment when contact occurs.
- DELAYED, specifying the application of friction is delayed until the increment after contact occurs.

The default value is IMMEDIATE.

automaticTolerances

A Boolean specifying whether Abaqus/Standard should automatically compute an overclosure tolerance and a separation tolerance to prevent chattering in contact. The default value is OFF.

The *automaticTolerances* argument cannot be used with the *maxchp*, *perrmx*, and *uerrmx* arguments.

maxchp

An Int specifying the maximum number of points that are permitted to violate contact conditions in any increment. The default value is 0.

Either the *perrmx* or the *uerrmx* argument must be specified in conjunction with the *maxchp* argument.

perrmx

A Float specifying the maximum value of tensile stress (tensile force in GAP- or ITT-type contact elements) allowed to be transmitted at a contact point. The default value is 0.0.

The *perrmx* argument must be specified in conjunction with the *maxchp* argument.

uerrmx

A Float specifying the maximum overclosure distance allowed at a secondary node that is considered to be open. The default value is 0.0.

The *uerrmx* argument must be specified in conjunction with the *maxchp* argument.

stabilizeChoice

A SymbolicConstant specifying whether or not viscous damping will be specified, and if so, how it will be specified. Possible values are NONE, AUTOMATIC, and COEFFICIENT. The default value is NONE.

dampFactor

A Float specifying the value of the damping factor. This value is multiplied by the calculated damping coefficient. The default value is 1.0.

This argument is only valid when *stabilizeChoice*=AUTOMATIC.

dampCoef

A Float specifying the damping coefficient. The default value is 0.0.

This argument is only valid when *stabilizeChoice*=COEFFICIENT.

tangFraction

A Float specifying the tangential stabilization as a fraction of the normal stabilization (damping). The default value is 1.0.

This argument is valid only if *stabilizeChoice* = AUTOMATIC or COEFFICIENT.

eosFraction

A Float specifying the fraction of the damping that remains at the end of the step. The default value is 0.0.

This argument is valid only if *stabilizeChoice* = AUTOMATIC or COEFFICIENT.

zeroDampingChoice

A SymbolicConstant specifying how the zero-damping clearance will be specified. Possible values are COMPUTE and SPECIFY. The default value is COMPUTE.

This argument is valid only if *stabilizeChoice* = AUTOMATIC or COEFFICIENT.

zeroDamping

`None` or a Float specifying the clearance at which damping becomes zero. This argument is valid only when *zeroDampingChoice*=SPECIFY. This argument is valid only if *stabilizeChoice* = AUTOMATIC or COEFFICIENT. The default value is `None`.

enforceWithLagrangeMultipliers

A SymbolicConstant specifying whether to enforce the contact constraints with Lagrange multipliers. Possible values are DEFAULT, ENFORCEMENT_OFF, and ENFORCEMENT_ON. The default value is DEFAULT.

Return value

A StdContactControl object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the StdContactControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StdContactControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The StdContactControl object has members with the same names and descriptions as the arguments to the *StdContactControl* method.

Corresponding analysis keywords

- **CONTACT CONTROLS*

StdInitialization object

The StdInitialization object is used in conjunction with *ContactStd* in Abaqus/Standard analyses to specify contact initialization data.

The StdInitialization object is derived from the *ContactInitialization* object.

Access

```
import interaction
mdb.models[name].contactInitializations[name]
```

StdInitialization(...)

This method creates a StdInitialization object.

Path

```
mdb.models[name].StdInitialization
```

Required arguments

name
A String specifying the contact initialization repository key.

Optional arguments

overclosureType
A SymbolicConstant specifying the type of overclosure to be defined. Possible values are ADJUST, INTERFERENCE, and CLEARANCE. The default value is ADJUST.

interferenceDistance
`None` or a Float specifying the interference distance. This argument is valid only when *overclosureType*=INTERFERENCE. The default value is `None`.

clearanceDistance
`None` or a Float specifying the initial clearance distance. This argument is valid only when *overclosureType*=CLEARANCE, and must be specified in that case. The default value is `None`.

openingTolerance
`None` or a Float specifying the distance tolerance within which initial openings will undergo strain-free adjustments. This argument is not valid when *overclosureType*=INTERFERENCE unless a value has been specified for *interferenceDistance*. The default value is `None`.

overclosureTolerance
`None` or a Float specifying the distance tolerance within which initial overclosures will undergo strain-free adjustments.. The default value is `None`.

Return value

A StdInitialization object.

Exceptions

RangeError.

setValues(...)

This method modifies the StdInitialization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StdInitialization* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The StdInitialization object has members with the same names and descriptions as the arguments to the *StdInitialization* method.

Corresponding analysis keywords

- **CONTACT INITIALIZATION DATA*

StdStabilization object

The StdStabilization object is used in conjunction with *ContactStd* in Abaqus/Standard analyses to specify contact stabilization.

The StdStabilization object is derived from the *ContactStabilization* object.

Access

```
import interaction
mdb.models[name].contactStabilizations[name]
```

StdStabilization(...)

This method creates a StdStabilization object.

Path

```
mdb.models[name].StdStabilization
```

Required arguments

name
A String specifying the contact stabilization repository key.

Optional arguments

zeroDistance
`None` or a Float specifying the clearance distance at which the stabilization becomes zero. The default value is `None`.

reductionFactor
A Float specifying the factor by which the analysis will reduce the contact stabilization coefficient per increment. The default value is 0.1.

scaleFactor
A Float specifying the factor by which the analysis will scale the contact stabilization coefficient. The default value is 1.0.

tangentialFactor
A Float specifying the factor that scales the contact stabilization coefficient in the tangential direction. The default value is 0.0.

amplitude
A String specifying the name of the *Amplitude* object that defines a time-dependent scale factor for contact stabilization over the step. The default value is an empty string.

reset
A Boolean specifying whether to cancel carryover effects from contact stabilization specifications involving nondefault amplitudes that appeared in previous steps. The default value is OFF.

Return value

A StdStabilization object.

Exceptions

RangeError.

setValues(...)

This method modifies the StdStabilization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StdStabilization* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The StdStabilization object has members with the same names and descriptions as the arguments to the *StdStabilization* method.

Corresponding analysis keywords

- **CONTACT STABILIZATION*

StdXplCosimulation object

The StdXplCosimulation object defines co-simulation behavior between Abaqus/Standard and Abaqus/Explicit.

The StdXplCosimulation object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

StdXplCosimulation(...)

This method creates a StdXplCosimulation object.

Path

```
mdb.models[name].StdXplCosimulation
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the StdXplCosimulation object is created.

region
A *Region* object specifying the import and export region upon which the co-simulation exchanges data with the coupled analysis program.

Optional arguments

incrementation
A SymbolicConstant specifying whether the analysis programs use the same time increments or one is allowed to use more time increments than the other before exchanging data. Possible values are ALLOW_SUBCYCLING and LOCKSTEP. The default value is ALLOW_SUBCYCLING.

stepSize
A Float specifying the size of the increments to be used by Abaqus/Standard and Abaqus/Explicit. The default value is 0.0.

stepSizeDefinition
A SymbolicConstant specifying whether the increment size is the analysis default or a supplied variable. Possible values are DEFAULT and SPECIFIED. The default value is DEFAULT.

Return value

A StdXplCosimulation object.

Exceptions

None.

`setValues(...)`

This method modifies the StdXplCosimulation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StdXplCosimulation* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

Members

The StdXplCosimulation object has members with the same names and descriptions as the arguments to the *StdXplCosimulation* method.

StdXplCosimulationState object

The StdXplCosimulationState object stores the propagating data for a *StdXplCosimulation* object. One instance of this object is created internally by the *StdXplCosimulation* object for each step. The instance is also deleted internally by the *StdXplCosimulation* object.

The StdXplCosimulationState object has no constructor or methods.

The StdXplCosimulationState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The StdXplCosimulationState object has the following member:

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CO-SIMULATION*,
- **CO-SIMULATION REGION*,

SurfaceBeamSmoothingAssignment object

The SurfaceBeamSmoothingAssignment object stores the surface beam smoothing assignment definition for surfaces in *ContactStd* objects. The SurfaceBeamSmoothingAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceBeamSmoothingAssignments
```

changeValuesInStep(...)

This method allows modification of surface beam smoothing assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface beam smoothing assignments are to be modified.

index An Int specifying the position of the surface beam smoothing assignment whose value is to be modified.

value A tuple specifying the value of the surface beam smoothing assignments for the surface whose index is referenced. Each tuple contains one entry:

- A Float specifying the surface beam smoothing value to be used for the surface.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface beam smoothing assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface beam smoothing assignments are to be defined.

assignments A sequence of tuples specifying the surface beam smoothing assignments. Each tuple contains two entries:

- A region object specifying the surface to which the smoothing is assigned.
- A Float specifying the surface smoothing value to be used for the surface.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface beam smoothing assignments from *ContactStd* objects.

Required arguments

indices A sequence of Ints specifying the index of each surface beam smoothing assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceBeamSmoothingAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=BEAM SMOOTHING

SurfaceCrushTriggerAssignment object

The SurfaceCrushTriggerAssignment object stores the surface crush trigger assignment definition for surfaces in *ContactExp* objects. The SurfaceCrushTriggerAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceCrushTriggerAssignments
```

changeValuesInStep(...)

This method allows modification of surface crush trigger assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface crush trigger assignments are to be modified.

index An Int specifying the position of the surface crush trigger assignment whose value is to be modified.

value A tuple specifying the value of the surface crush trigger assignments for the surface whose index is referenced. Each tuple contains three entries:

- A SymbolicConstant specifying the trigger option to be used for the surface. Possible values of the SymbolicConstant are TRIGGER, NO_TRIGGER, or NO_CRUSH.
- A Float specifying the crush stress value to be used for the surface.
- A Float specifying the crush initiation angle value to be used for the surface.
- A Float specifying the crush continuation angle value to be used for the surface.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface crush trigger assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface crush trigger assignments are to be defined.

assignments A sequence of tuples specifying the surface crush trigger assignments. Each tuple contains four entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the feature angle is assigned.
- A SymbolicConstant specifying the trigger option to be used for the surface. Possible values of the SymbolicConstant are TRIGGER, NO_TRIGGER, or NO_CRUSH.
- A Float specifying the crush stress value to be used for the surface.
- A Float specifying the crush initiation angle value to be used for the surface.
- A Float specifying the crush continuation angle value to be used for the surface.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface crush trigger assignments from a *ContactExp* object.

Required arguments

indices A sequence of Ints specifying the index of each surface crush trigger assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceCrushTriggerAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=CRUSH TRIGGER

SurfaceFeatureAssignment object

The SurfaceFeatureAssignment object stores the surface feature angle assignment definition for surfaces in *ContactExp* or *ContactStd* objects. The SurfaceFeatureAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceFeatureAssignments
```

changeValuesInStep(...)

This method allows modification of surface feature angle assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface feature assignments are to be modified.

index An Int specifying the position of the surface feature angle assignment whose value is to be modified.

value A tuple specifying the value of the surface feature assignments for the surface whose index is referenced. Each tuple contains two entries for Abaqus/Standard and three entries for Abaqus/Explicit:

- A Float or a SymbolicConstant specifying the overriding surface primary feature angle value to be used for the surface. Possible values of the SymbolicConstant are PERIMETER, ALL, PICKED, or NONE. The ALL and PICKED values cannot be specified with the GLOBAL region constant and can be used only in the Abaqus/Explicit version of general contact.
- A Float or a SymbolicConstant specifying the overriding surface secondary feature angle value to be used for the surface. Possible values of the SymbolicConstant are PERIMETER, ALL, PICKED, or NONE. The ALL and PICKED values cannot be specified with the GLOBAL region constant and can be used only in the Abaqus/Explicit version of general contact.
- A SymbolicConstant ORIGINAL or CURRENT specifying the configuration.

Optional arguments

useDynFeatEdge A Boolean specifying whether the dynamic feature edge criterion should be activated for the contact surface in an Abaqus/Explicit analysis. This argument can be set to ON (default) or OFF.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows the addition of surface feature angle assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface feature angle assignments are to be defined.

assignments A sequence of tuples specifying the surface feature angle assignments. Each tuple contains three entries for Abaqus/Standard and four entries for Abaqus/Explicit:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the feature angle is assigned.
- A Float or a SymbolicConstant specifying the overriding surface primary feature angle value to be used for the surface. Possible values of the SymbolicConstant are PERIMETER, ALL, PICKED, or NONE. The ALL and PICKED values cannot be specified with the GLOBAL region constant and can be used only in the Abaqus/Explicit version of general contact.
- A Float or a SymbolicConstant specifying the overriding surface secondary feature angle value to be used for the surface. Possible values of the SymbolicConstant are PERIMETER, ALL, PICKED, or NONE. The ALL and PICKED values cannot be specified with the GLOBAL region constant and can be used only in the Abaqus/Explicit version of general contact.
- A SymbolicConstant ORIGINAL or CURRENT specifying the configuration.

Optional arguments

useDynFeatEdge A Boolean specifying whether the dynamic feature edge criterion should be activated for the contact surface in an Abaqus/Explicit analysis. This argument can be set to ON (default) or OFF.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface feature angle assignments from a *ContactExp* object.

Required arguments

indices A sequence of Ints specifying the index of each surface feature angle assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceFeatureAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=FEATURE EDGE CRITERIA

SurfaceFrictionAssignment object

The SurfaceFrictionAssignment object stores the surface friction assignment definition for surfaces in *ContactExp* objects. The SurfaceFrictionAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceFrictionAssignments
```

changeValuesInStep(...)

This method allows modification of surface friction assignments already defined on surfaces in a given step.

Required arguments

stepName
A String specifying the name of the step in which the surface friction assignments are to be modified.

index
An Int specifying the position of the surface friction assignment whose value is to be modified.

value
A tuple specifying the value of the surface friction assignments for the surface whose index is referenced. Each tuple contains:

- A Float specifying the overriding friction coefficient value to be used in the contact definition.

Optional arguments

None.

Return value

None.

Exceptions

None.

appendInStep(...)

This method allows addition of surface friction assignments to new surfaces in a given step.

Required arguments

stepName
A String specifying the name of the step in which new surface friction assignments are to be defined.

assignments
A sequence of tuples specifying the surface friction assignments. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the friction coefficient is assigned.
- A Float specifying the overriding friction coefficient to be used in the contact definition.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface friction assignments.

Required arguments

indices A sequence of Ints specifying the index of each surface friction assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceFrictionAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=FRICTION

SurfaceOffsetAssignment object

The SurfaceOffsetAssignment object stores the surface offset fraction assignment definition for surfaces in *ContactExp* and *ContactStd* objects. The SurfaceOffsetAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceOffsetAssignments
```

changeValuesInStep(...)

This method allows modification of surface offset fraction assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface offset assignments are to be modified.

index An Int specifying the position of the surface offset fraction assignment whose value is to be modified.

value A tuple specifying the value of the surface offset assignments for the surface whose index is referenced. Each tuple contains one entry:

- A Float or a SymbolicConstant specifying the surface offset fraction value to be used for the surface. Possible values of the SymbolicConstant are ORIGINAL, SPOS, and SNEG.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface offset fraction assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface offset fraction assignments are to be defined.

assignments A sequence of tuples specifying the surface offset fraction assignments. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the offset fraction is assigned.
- A Float or a SymbolicConstant specifying the surface offset fraction value to be used for the surface. Possible values of the SymbolicConstant are ORIGINAL, SPOS, and SNEG.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface offset fraction assignments.

Required arguments

indices A sequence of Ints specifying the index of each surface offset fraction assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceOffsetAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=OFFSET FRACTION

SurfaceThicknessAssignment object

The SurfaceThicknessAssignment object stores the surface thickness assignment definition for surfaces in *ContactExp* and *ContactStd* objects. The SurfaceThicknessAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfaceThicknessAssignments
```

changeValuesInStep(...)

This method allows modification of surface thickness assignments already defined on surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which the surface thickness assignments are to be modified.

index An Int specifying the position of the surface thickness assignment whose value is to be modified.

value A tuple specifying the value of the surface thickness assignments for the surface whose index is referenced. Each tuple contains two entries:

- A Float or a SymbolicConstant specifying the overriding thickness value to be used in the contact definition. Possible values of the SymbolicConstant are ORIGINAL and THINNING. The SymbolicConstant THINNING can be specified only in an Abaqus/Explicit analysis.
- A Float specifying a scale factor that multiplies the thickness value specified in the second entry.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface thickness assignments to new surfaces in a given step.

Required arguments

stepName A String specifying the name of the step in which new surface thickness assignments are to be defined.

assignments A sequence of tuples specifying the surface thickness assignments. Each tuple contains three entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the thickness is assigned.
- A Float or a SymbolicConstant specifying the overriding thickness value to be used in the contact definition. Possible values of the SymbolicConstant are ORIGINAL and THINNING. The SymbolicConstant THINNING can be specified only in an Abaqus/Explicit analysis.
- A Float specifying a scale factor that multiplies the thickness value specified in the second entry.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface thickness assignments.

Required arguments

indices A sequence of Ints specifying the index of each surface thickness assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceThicknessAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=THICKNESS

SurfaceToSurfaceContactExp object

The SurfaceToSurfaceContactExp object defines surface-to-surface contact during an Abaqus/Explicit analysis.

The SurfaceToSurfaceContactExp object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

SurfaceToSurfaceContactExp(...)

This method creates a SurfaceToSurfaceContactExp object.

Path

```
mdb.models[name].SurfaceToSurfaceContactExp
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the SurfaceToSurfaceContactExp object is created.

main
A *Region* object specifying the main surface.

secondary
A *Region* object specifying the secondary surface.

sliding
A SymbolicConstant specifying the contact formulation. Possible values are FINITE and SMALL.

interactionProperty
A String specifying the name of the *ContactProperty* object associated with this interaction.

Optional arguments

mechanicalConstraint
A SymbolicConstant specifying the mechanical constraint formulation. Possible values are KINEMATIC and PENALTY. The default value is KINEMATIC.

weightingFactorType
A SymbolicConstant specifying the weighting for node-to-face contact. Possible values are DEFAULT and SPECIFIED. The default value is DEFAULT.

weightingFactor
A Float specifying the weighting factor for the contact surfaces when *weightingFactorType*=SPECIFIED. The default value is 0.0.

contactControls
A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

initialClearance A SymbolicConstant or a Float specifying the initial clearance at regions of contact. Possible values are OMIT and COMPUTED. The default value is OMIT.

halfThreadAngle `None` or a sequence of Floats specifying the half thread angle used for bolt clearance. The default value is `None`.

pitch `None` or a sequence of Floats specifying the pitch used for bolt clearance. The default value is `None`.

majorBoltDiameter The SymbolicConstant COMPUTED or a Float specifying the major diameter of the bolt used for bolt clearance. The default value is COMPUTED.

meanBoltDiameter The SymbolicConstant COMPUTED or a Float specifying the mean diameter of the bolt used for bolt clearance. The default value is COMPUTED.

datumAxis A *DatumAxis* object specifying the orientation of the bolt hole when specifying bolt clearance.

useReverseDatumAxis A Boolean specifying whether to reverse the bolt clearance direction given by the datum axis. The default value is OFF.

clearanceRegion A *Region* object specifying the contact region for which clearance is specified.

Return value

A SurfaceToSurfaceContactExp object.

Exceptions

None.

swapSurfaces()

This method switches the main and secondary surfaces of a surface-to-surface contact pair. This command is valid only during the step in which the interaction is created.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceToSurfaceContactExp object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceToSurfaceContactExp* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing SurfaceToSurfaceContactExp object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

interactionProperty A String specifying the name of the *ContactProperty* object associated with this interaction.

contactControls A String specifying the name of the *ContactControl* object associated with this interaction. An empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

`None.`

Exceptions

None.

Members

The SurfaceToSurfaceContactExp object has members with the same names and descriptions as the arguments to the *SurfaceToSurfaceContactExp* method except the optional arguments to the *setValuesInStep* method.

SurfaceToSurfaceContactStd object

The SurfaceToSurfaceContactStd object defines surface-to-surface contact during an Abaqus/Standard analysis.

The SurfaceToSurfaceContactStd object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

SurfaceToSurfaceContactStd(...)

This method creates a SurfaceToSurfaceContactStd object.

Path

```
mdb.models[name].SurfaceToSurfaceContactStd
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the SurfaceToSurfaceContactStd object is created.

main
A *Region* object specifying the main surface.

secondary
A *Region* object specifying the secondary surface.

sliding
A SymbolicConstant specifying the contact formulation. Possible values are FINITE and SMALL.

interactionProperty
A String specifying the name of the *ContactProperty* object associated with this interaction.

Optional arguments

interferenceType
A SymbolicConstant specifying the type of time-dependent allowable interference for contact pairs and contact elements. Possible values are:

- NONE, specifying no allowable contact interference.
- SHRINK_FIT.
- UNIFORM.

The default value is NONE.

overclosure
A Float specifying the maximum overclosure distance allowed. This argument applies only when *interferenceType*=UNIFORM. The default value is 0.0.

interferenceDirectionType

A SymbolicConstant specifying the method used to determine the interference direction. Possible values are COMPUTED and DIRECTION_COSINE. The default value is COMPUTED.

direction

A sequence of three Floats specifying the following:

- ***X***-direction cosine of the interference direction vector.
- ***Y***-direction cosine of the interference direction vector.
- ***Z***-direction cosine of the interference direction vector.

This argument is required only when *interferenceDirectionType*=DIRECTION_COSINE.

amplitude

A String specifying the name of the amplitude curve that defines the magnitude of the prescribed interference during the step. Use `None` to specify that the prescribed interference is applied immediately at the beginning of the step and ramped down to zero linearly over the step.

smooth

A Float specifying the degree of smoothing used for deformable or rigid main surfaces involved when *enforcement*=NODE_TO_SURFACE. The value given must lie between 0.0 and 0.5. The default value is 0.2.

hcrit

A Float specifying the distance by which a secondary node must penetrate the main surface before Abaqus/Standard abandons the current increment and tries again with a smaller increment. The default value is 0.0.

extensionZone

A Float specifying a fraction of the end segment or facet edge length by which the main surface is to be extended to avoid numerical round-off errors associated with contact modeling. The value given must lie between 0.0 and 0.2. The default value is 0.1.

adjustMethod

A SymbolicConstant specifying the adjust method. Possible values are NONE, OVERCLOSED, TOLERANCE, and SET. The default value is NONE.

adjustTolerance

A Float specifying the adjust tolerance. The default value is 0.0.

adjustSet

A *Region* object specifying the Set object to which the adjustment is to be applied.

enforcement

A SymbolicConstant specifying the discretization method. Possible values are NODE_TO_SURFACE and SURFACE_TO_SURFACE. The default value is SURFACE_TO_SURFACE.

thickness

A Boolean specifying whether shell/membrane element thickness is considered. The default value is ON.

This argument is not valid when *sliding*=FINITE and *enforcement*=NODE_TO_SURFACE.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction. The empty string indicates that the default contact controls will be used. The default value is an empty string.

tied
A Boolean specifying whether the surfaces are to be "tied" together for the duration of the simulation. The default value is OFF.

initialClearance
A SymbolicConstant or a Float specifying the initial clearance at regions of contact. Possible values are OMIT and COMPUTED. The default value is OMIT.

halfThreadAngle
`None` or a sequence of Floats specifying the half thread angle used for bolt clearance. The default value is `None`.

pitch
`None` or a sequence of Floats specifying the pitch used for bolt clearance. The default value is `None`.

majorBoltDiameter
The SymbolicConstant COMPUTED or a Float specifying the major diameter of the bolt used for bolt clearance. The default value is COMPUTED.

meanBoltDiameter
The SymbolicConstant COMPUTED or a Float specifying the mean diameter of the bolt used for bolt clearance. The default value is COMPUTED.

datumAxis
A *DatumAxis* object specifying the orientation of the bolt hole when specifying bolt clearance.

useReverseDatumAxis
A Boolean specifying whether to reverse the bolt clearance direction given by the datum axis. The default value is OFF.

clearanceRegion
A *Region* object specifying the contact region for which clearance is specified.

surfaceSmoothing
A SymbolicConstant specifying whether to use surface smoothing for geometric surfaces in SurfaceToSurfaceContactStd interactions. Possible values are AUTOMATIC and NONE. The default value is NONE.

bondingSet
A *Region* object specifying the secondary node sub-set for bonding, used only when the contact property *CohesiveBehavior* option specifies use.

handedness
A SymbolicConstant specifying the bolt handedness formulation. Possible values are RIGHT and LEFT. The default value is RIGHT.

normalAdjustment
A SymbolicConstant specifying the bolt normal adjustment formulation for all secondary nodes. Possible values are UNIFORM AXIAL COMPONENT and LOCATION DEPENDENT. The default value is UNIFORM AXIAL COMPONENT.

Return value

A SurfaceToSurfaceContactStd object.

Exceptions

None.

swapSurfaces()

This method switches the main and secondary surfaces of a surface-to-surface contact pair. This command is valid only for the step in which the interaction is created.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceToSurfaceContactStd object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceToSurfaceContactStd* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceToSurfaceContactStd object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

interactionProperty A String specifying the name of the *ContactProperty* object associated with this interaction.

interferenceType A SymbolicConstant specifying the type of time-dependent allowable interference for contact pairs and contact elements. Possible values are:

- NONE, specifying no allowable contact interference.
- SHRINK_FIT.
- UNIFORM.

The default value is NONE.

overclosure

A Float specifying the maximum overclosure distance allowed. This argument applies only when *interferenceType*=UNIFORM. The default value is 0.0.

interferenceDirectionType

A SymbolicConstant specifying the method used to determine the interference direction. Possible values are COMPUTED and DIRECTION_COSINE. The default value is COMPUTED.

direction

A sequence of three Floats specifying the following:

- ***X***-direction cosine of the interference direction vector.
- ***Y***-direction cosine of the interference direction vector.
- ***Z***-direction cosine of the interference direction vector.

This argument is required only when *interferenceDirectionType*=DIRECTION_COSINE.

amplitude

A String specifying the name of the amplitude curve that defines the magnitude of the prescribed interference during the step. Use `None` to specify that the prescribed interference is applied immediately at the beginning of the step and ramped down to zero linearly over the step.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction. The empty string indicates that the default contact controls will be used. The default value is an empty string.

Return value

`None.`

Exceptions

None.

Members

The SurfaceToSurfaceContactStd object has members with the same names and descriptions as the arguments to the *SurfaceToSurfaceContactStd* method except the optional arguments to the *setValuesInStep* method.

In addition, the SurfaceToSurfaceContactStd object has the following members:

contactTracking

A SymbolicConstant specifying the choice of contact tracking algorithm. The STATE tracking algorithm uses only normal projections and is specified by using ONE_CONFIG. The PATH tracking algorithm uses crossing and normal projections and is specified by using TWO_CONFIG. Possible values are ONE_CONFIG and TWO_CONFIG. The default value is TWO_CONFIG.

This argument is valid only when *sliding*=FINITE and *enforcement*=SURFACE_TO_SURFACE.

supplementaryContact

A SymbolicConstant specifying the manner in which midface constraints are employed. Possible values are SELECTIVE, NEVER, and ALWAYS. The default value is SELECTIVE.

This argument is not valid when *sliding*=FINITE and *enforcement*=SURFACE_TO_SURFACE.

SurfaceToSurfaceExpState object

The SurfaceToSurfaceExpState object stores the propagating data for a *SurfaceToSurfaceContactExp* object. One instance of this object is created internally by the *SurfaceToSurfaceContactExp* object for each step. The instance is also deleted internally by the *SurfaceToSurfaceContactExp* object.

The SurfaceToSurfaceExpState object has no constructor or methods.

The SurfaceToSurfaceExpState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The SurfaceToSurfaceExpState object has the following members:

interactionPropertyState

A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

contactControlsState

A SymbolicConstant specifying the propagation state of the *contactControls* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty

A String specifying the name of the *ContactProperty* object associated with this interaction.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CONTACT CONTROLS*
- **CONTACT PAIR*

SurfaceToSurfaceStdState object

The SurfaceToSurfaceStdState object stores the propagating data for a *SurfaceToSurfaceContactStd* object. One instance of this object is created internally by the *SurfaceToSurfaceContactStd* object for each step. The instance is also deleted internally by the *SurfaceToSurfaceContactStd* object.

The SurfaceToSurfaceStdState object has no constructor or methods.

The SurfaceToSurfaceStdState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The SurfaceToSurfaceStdState object has the following members:

interactionPropertyState
A SymbolicConstant specifying the propagation state of the *interactionProperty* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interferenceType
A SymbolicConstant specifying the interference type. Possible values are NONE, SHRINK_FIT, and UNIFORM.

interferenceTypeState
A SymbolicConstant specifying the propagation state of the *interferenceType* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

overclosure
A Float specifying the allowable overclosure.

overclosureState
A SymbolicConstant specifying the propagation state of the *overclosure* member. Possible values are COMPUTED and DIRECTION_COSINE.

interferenceDirectionType
A SymbolicConstant specifying the interference direction type. Possible values are COMPUTED and DIRECTION_COSINE.

interferenceDirectionTypeState
A SymbolicConstant specifying the propagation state of the *interferenceDirectionType* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

directionState
A SymbolicConstant specifying the propagation state of the *direction* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState
A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

contactControlsState
A SymbolicConstant specifying the propagation state of the *contactControls* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

interactionProperty
A String specifying the name of the *ContactProperty* object associated with this interaction.

amplitude

A String specifying the name of the Amplitude object that defines the magnitude of the prescribed interference during the step.

contactControls

A String specifying the name of the *ContactControl* object associated with this interaction.

direction

A tuple of three Floats specifying the following:

- ***X***-direction cosine of the interference direction vector.
- ***Y***-direction cosine of the interference direction vector.
- ***Z***-direction cosine of the interference direction vector.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **CONTACT CONTROLS*
- **CONTACT PAIR*
- **CONTACT INTERFERENCE*

SurfaceVertexCriteriaAssignment object

The SurfaceVertexCriteriaAssignment object stores the surface vertex criteria assignment definition for surfaces in *ContactStd* objects. The SurfaceVertexCriteriaAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].surfacevertexCriteriaAssignments
```

changeValuesInStep(...)

This method allows modification of surface vertex criteria assignments already defined on surfaces in a given step.

Required arguments

stepName
A String specifying the name of the step in which the surface vertex criteria assignments are to be modified.

index
An Int specifying the position of the surface vertex criteria assignment whose value is to be modified.

value
A tuple specifying the value of the surface vertex criteria assignments for the surface whose index is referenced. Each tuple contains:

- A Float or a SymbolicConstant specifying the vertex criteria value to be used for the surface. Possible values of the SymbolicConstant are ALL_VERTICES or NO_VERTICES.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows addition of surface vertex criteria assignments to new surfaces in a given step.

Required arguments

stepName
A String specifying the name of the step in which new surface vertex criteria assignments are to be defined.

assignments
A sequence of tuples specifying the surface vertex criteria assignments. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL specifying the surface to which the vertex criteria is assigned.
- A Float or a SymbolicConstant specifying the vertex criteria value to be used for the surface. Possible values of the SymbolicConstant are ALL_VERTICES or NO_VERTICES.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing surface vertex criteria assignments from a *ContactStd* object.

Required arguments

indices
A sequence of Ints specifying the index of each surface vertex criteria assignment to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The SurfaceVertexCriteriaAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*, PROPERTY=VERTEX CRITERIA

ThermalConductance object

The ThermalConductance object specifies thermal conductance for a contact interaction property.

Access

```
import interaction
mdb.models[name].interactionProperties[name].thermalConductance
```

ThermalConductance(...)

This method creates a ThermalConductance object.

Path

```
mdb.models[name].interactionProperties[name].ThermalConductance
```

Required arguments

None.

Optional arguments

definition
A SymbolicConstant specifying how the thermal conductance is defined. Possible values are TABULAR and USER_DEFINED. The default value is TABULAR.

clearanceDependency
A Boolean specifying whether to use clearance-dependent data. The default value is ON.

pressureDependency
A Boolean specifying whether to use pressure-dependent data. The default value is OFF.

temperatureDependencyC
A Boolean specifying whether to use temperature-dependent data with clearance dependency. The default value is OFF.

massFlowRateDependencyC
A Boolean specifying whether to use mass-flow-rate-dependent data with clearance dependency. The default value is OFF.

dependenciesC
An Int specifying the number of field variables to use with clearance dependency. The default value is 0.

clearanceDepTable
A sequence of sequences of Floats specifying clearance dependency data. The items in the table data are described below.

temperatureDependencyP
A Boolean specifying whether to use temperature-dependent data with pressure dependency. The default value is OFF.

massFlowRateDependencyP
A Boolean specifying whether to use mass-flow-rate-dependent data with pressure dependency. The default value is OFF.

dependenciesP An Int specifying the number of field variables to use with pressure dependency. The default value is 0.

pressureDepTable A sequence of sequences of Floats specifying pressure dependency data. The items in the table data are described below.

Table data

The *clearanceDepTable* data specify the following:

- Conductivity.
- Clearance.
- Temperature, if the data depend on temperature.
- Mass flow rate, if the data depend on mass flow rate.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

The *pressureDepTable* data specify the following:

- Conductivity.
- Pressure.
- Temperature, if the data depend on temperature.
- Mass flow rate, if the data depend on mass flow rate.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ThermalConductance object.

Exceptions

None.

`setValues(...)`

This method modifies the ThermalConductance object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ThermalConductance* method.

Return value

```
None.
```

Exceptions

None.

Members

The ThermalConductance object has members with the same names and descriptions as the arguments to the *ThermalConductance* method.

Corresponding analysis keywords

- **GAP CONDUCTANCE*

WearProperty object

The WearProperty object defines a wear interaction property.

The WearProperty object is derived from the *InteractionProperty* object.

Access

```
import interaction
mdb.models[name].interactionProperties[name]
```

WearProperty(...)

This method creates a WearProperty object.

Path

```
mdb.models[name].WearProperty
```

Required arguments

name	A String specifying the interaction property repository key.

Optional arguments

fricCoefDependency	A Boolean specifying whether the Archard's wear equation depends explicitly on the friction coefficient. The default value is OFF.
unitlessWearCoefDependency	A Boolean specifying whether the wear coefficient is dimensionless. The default value is OFF.
referenceStress	A Float specifying the value of the reference stress. This parameter is required if unitlessWearCoefDependency is ON.
surfaceWearDistanceDependency	A Boolean specifying whether the wear coefficient is dependent on the wear distance. The default value is OFF.
temperatureDependency	A Boolean specifying whether the wear coefficient is dependent on the surface temperature. The default value is OFF.
contactPressureDependency	A Boolean specifying whether the wear coefficient is dependent on the contact pressure. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.

Return value

A WearProperty object.

Exceptions

None.

Members

The WearProperty object can have the following members:

Corresponding analysis keywords

- **SURFACE INTERACTION*

WearSurfacePropertyAssignment object

The WearSurfacePropertyAssignment object stores the wear surface property assignment definition for domain pairs in *ContactExp* and *ContactStd* objects. The WearSurfacePropertyAssignment object has no constructor or members.

Access

```
import interaction
mdb.models[name].interactions[name].wearSurfacePropertyAssignments
```

changeValuesInStep(...)

This method allows modification of contact property assignments to domain pairs already defined in a given step.

Required arguments

stepName A String specifying the name of the step in which the contact property assignments are to be modified.

index An Int specifying the position of the contact property assignment whose value is to be modified.

value A String specifying the value of the wear surface property to be assigned to the domain pair whose index is referenced.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

appendInStep(...)

This method allows the addition of wear surface property assignments to new domain pairs in a given step.

Required arguments

stepName A String specifying the name of the step in which new contact property assignments are to be defined.

assignments A sequence of tuples specifying the properties assigned to each surface pair. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL.
- A String specifying a *WearProperty* object associated with this pair of regions.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`delete(...)`

The `delete` method allows you to delete existing contact property assignments.

Required arguments

indices
A sequence of Ints specifying the index of each contact property assignment to delete. The *indices* and *surfPair* arguments are mutually exclusive.

surfPair
A sequence of tuples specifying the surface pair of each contact property assignment to delete. Each tuple contains two entries:

- A region or a material object or the SymbolicConstant GLOBAL.

surfPair and *indices* arguments are mutually exclusive.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The WearSurfacePropertyAssignment object has no members.

Corresponding analysis keywords

- **SURFACE PROPERTY ASSIGNMENT*

XFEMCrackGrowth object

The XFEMCrackGrowth object defines the enrichment activation state for an *XFEMCrack*.

The XFEMCrackGrowth object is derived from the *Interaction* object.

Access

```
import interaction
mdb.models[name].interactions[name]
```

XFEMCrackGrowth(...)

This method creates an XFEMCrackGrowth object.

Path

```
mdb.models[name].XFEMCrackGrowth
```

Required arguments

name A String specifying the repository key.

createStepName A String specifying the name of the step in which the XFEMCrackGrowth object is created.

crackName A String specifying the *XFEMCrack* object associated with this interaction.

Optional arguments

allowGrowth A Boolean specifying whether the crack is allowed to grow (propagate) during this analysis step. The default value is ON.

Return value

A XFEMCrackGrowth object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing XFEMCrackGrowth object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *XFEMCrackGrowth* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing XFEMCrackGrowth object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the interaction is modified.

Optional arguments

allowGrowth A Boolean specifying whether the crack is allowed to grow (propagate) during this analysis step. The default value is ON.

Return value

`None.`

Exceptions

None.

Members

The XFEMCrackGrowth object has members with the same names and descriptions as the arguments to the *XFEMCrackGrowth* method.

Corresponding analysis keywords

- **ENRICHMENT ACTIVATION*

XFEMCrackGrowthState object

The XFEMCrackGrowthState object stores the propagating data of an *XFEMCrackGrowth* object in a step. One instance of this object is created internally by the *XFEMCrackGrowth* object for each step. The instance is also deleted internally by the *XFEMCrackGrowth* object.

The XFEMCrackGrowthState object has no constructor or methods.

The XFEMCrackGrowthState object is derived from the *InteractionState* object.

Access

```
import interaction
mdb.models[name].steps[name].interactionStates[name]
```

Members

The XFEMCrackGrowthState object has the following members:

allowGrowth

A Boolean specifying whether the crack is allowed to grow (propagate) during this analysis step. The default value is ON.

allowGrowthState

A SymbolicConstant specifying the propagation state of the *allowGrowth* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *InteractionState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **ENRICHMENT ACTIVATION*

Job commands

The Job commands provide methods to create, modify, submit, and control jobs.

In this section:

- *Job object*
- *JobFromInputFile object*
- *Message object*
- *ModelJob object*
- *Queue object*
- *Coexecution object*
- *OptimizationProcess object*

Job object

The Job object is the abstract base type for other Job objects. The Job object has no explicit constructor. The methods and members of the Job object are common to all objects derived from Job.

Access

```
import job
mdb.coexecutions[name].jobs[name]
mdb.jobs[name]
```

kill()

This method ends the analysis of a job.

Arguments

None.

Return value

None.

Exceptions

None.

submit(...)

This method submits a job for analysis.

Required arguments

None.

Optional arguments

consistencyChecking A Boolean specifying whether to perform consistency checking for the job. The default value is ON. It is not recommended that you turn off the consistency checking unless you are absolutely sure the model is consistent.

datacheckJob A Boolean specifying whether to run the job as a datacheck analysis. The default value is False. The datacheckJob and continueJob arguments cannot both be True.

continueJob A Boolean specifying whether to run the job as a continuation analysis. The default value is False. The datacheckJob and continueJob arguments cannot both be True.

Return value

None.

Exceptions

None.

waitForCompletion()

This method interrupts the execution of the script until the end of the analysis. If you call the `waitForCompletion` method and the *status* member is neither SUBMITTED nor RUNNING, Abaqus assumes that the analysis has either completed or stopped and returns immediately.

Arguments

None.

Return value

`None.`

Exceptions

None.

clearMessages

This method clears *messages* and sets the *status* to NONE.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Job object can have the following members:

name

A String specifying the name of the new job. The name must be a valid Abaqus/CAE object name.

type

A SymbolicConstant specifying the type of job. Possible values are ANALYSIS, SYNTAXCHECK, RECOVER, and RESTART. The default value is ANALYSIS.

If the object has the type *JobFromInputFile*, *type*=RESTART is not available.

waitHours

An Int specifying the number of hours to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitMinutes*. *waitHours* and *atTime* are mutually exclusive.

waitMinutes

An Int specifying the number of minutes to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitHours*. *waitMinutes* and *atTime* are mutually exclusive.

numCpus

An Int specifying the number of CPUs to use for this analysis if parallel processing is available. Possible values are *numCpus*> 0. The default value is 1.

memory

An Int specifying the amount of memory available to an Abaqus analysis. The value should be expressed in the units supplied in *memoryUnits*. The default value is 90.

memoryUnits

A SymbolicConstant specifying the units for the amount of memory used in an Abaqus analysis. Possible values are PERCENTAGE, MEGA_BYTES, and GIGA_BYTES. The default value is PERCENTAGE.

getMemoryFromAnalysis

A Boolean specifying whether to retrieve the recommended memory settings from the last datacheck or analysis run and use those values in subsequent submissions. The default value is ON.

explicitPrecision

A SymbolicConstant specifying whether to use the double precision version of Abaqus/Explicit. Possible values are SINGLE, FORCE_SINGLE, DOUBLE, DOUBLE_CONSTRAINT_ONLY, and DOUBLE_PLUS_PACK. The default value is SINGLE.

nodalOutputPrecision

A SymbolicConstant specifying the precision of the nodal output written to the output database. Possible values are SINGLE and FULL. The default value is SINGLE.

numDomains

An Int specifying the number of domains for parallel execution in Abaqus/Explicit. When using more than 1 *numCpus*, *numDomains* must be a multiple of *numCpus*. The default value is 1.

activateLoadBalancing

A Boolean specifying whether to activate dyanmic load balancing for jobs running on multiple processors with multiple domains in Abaqus/Explicit. The default value is OFF.

multiprocessingMode

A SymbolicConstant specifying whether an analysis is decomposed into threads or into multiple processes that communicate through a message passing interface (MPI). Possible values are DEFAULT, THREADS,MPI, and HYBRID. The default value is DEFAULT.

numThreadsPerMpiProcess

An Int specifying the number of threads per MPI process to use for this analysis if parallel processing is available. Possible values are *numThreadsPerMpiProcess*> 0. The default value is 1.

analysis

A SymbolicConstant specifying whether the job will be analyzed by Abaqus/Standard or Abaqus/Explicit. Possible values are STANDARD, EXPLICIT, and UNKNOWN.

If the object has the type *JobFromInputFile*, *analysis*=UNKNOWN.

status

A SymbolicConstant specifying the status of the analysis. Possible values are SUBMITTED, RUNNING, ABORTED, TERMINATED, COMPLETED, CHECK_RUNNING, and CHECK_COMPLETED.

If the *message* member is empty, *status* is set to NONE.

queue

A String specifying the name of the queue to which to submit the job. The default value is an empty string.



Note:

You can use the *queue* argument when creating a Job object on a Windows workstation; however, remote queues are available only on Linux platforms.

atTime

A String specifying the time at which to submit the job. If *queue* is empty, the string syntax must be valid for the Linux`at` command. If *queue* is set, the syntax must be valid according to the system administrator. The default value is an empty string.



Note:

You can use the *atTime* argument when creating a Job object on a Windows workstation; however, the `at` command is available only on Linux platforms.

scratch

A String specifying the location of the scratch directory. The default value is an empty string.

userSubroutine

A String specifying the file containing the user's subroutine definitions. The default value is an empty string.

messages

A *MessageArray* object specifying the messages received during an analysis.

environment

A tuple of Strings specifying the environment variables and their values.

licenseType

A SymbolicConstant specifying the type of license type being used in the case of the DSLS SimUnit license model. Possible values are DEFAULT, TOKEN, and CREDIT. The default value is DEFAULT.

If the license model is not the DSLS SimUnit, the licenseType is not available.

Corresponding analysis keywords

- **HEADING*
- **PREPRINT*

JobFromInputFile object

The JobFromInputFile object defines a *Job* object that analyzes a model contained in an input file.

The JobFromInputFile object is derived from the *Job* object.

Access

```
import job
mdb.jobs[name]
```

JobFromInputFile(...)

This method creates an analysis job using an input file for the model definition.

Path

```
mdb.JobFromInputFile
```

Required arguments

name

A String specifying the name of the new job. The name must be a valid Abaqus/CAE object name.

inputFileName

A String specifying the input file to read. Possible values are any valid file name. If the `.inp` extension is not included in the value of the argument, the system will append it for the user.

Optional arguments

type

A SymbolicConstant specifying the type of job. Possible values are ANALYSIS, SYNTAXCHECK, and RECOVER. The default value is ANALYSIS.

For theJobFromInputFile object, *type*=RESTART is not currently supported.

queue

A String specifying the name of the queue to which to submit the job. The default value is an empty string.



Note: You can use the *queue* argument when creating a Job object on a Windows workstation; however, remote queues are available only on Linux platforms.

waitHours

An Int specifying the number of hours to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitMinutes*. *waitHours* and *atTime* are mutually exclusive.

waitMinutes

An Int specifying the number of minutes to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitHours*. *waitMinutes* and *atTime* are mutually exclusive.

atTime

A String specifying the time at which to submit the job. If *queue* is empty, the string syntax must be valid for the Linux`at` command. If *queue* is set, the syntax must be valid according to the system administrator. The default value is an empty string.



Note: You can use the *atTime* argument when creating a Job object on a Windows workstation; however, the `at` command is available only on Linux platforms.

scratch

A String specifying the location of the scratch directory. The default value is an empty string.

userSubroutine

A String specifying the file containing the user's subroutine definitions. The default value is an empty string.

numCpus

An Int specifying the number of CPUs to use for this analysis if parallel processing is available. Possible values are *numCpus*> 0. The default value is 1.

memory

An Int specifying the amount of memory available to an Abaqus analysis. The value should be expressed in the units supplied in *memoryUnits*. The default value is 90.

memoryUnits

A SymbolicConstant specifying the units for the amount of memory used in an Abaqus analysis. Possible values are PERCENTAGE, MEGA_BYTES, and GIGA_BYTES. The default value is PERCENTAGE.

explicitPrecision

A SymbolicConstant specifying whether to use the double precision version of Abaqus/Explicit. Possible values are SINGLE, FORCE_SINGLE, DOUBLE, DOUBLE_CONSTRAINT_ONLY, and DOUBLE_PLUS_PACK. The default value is SINGLE.

nodalOutputPrecision

A SymbolicConstant specifying the precision of the nodal output written to the output database. Possible values are SINGLE and FULL. The default value is SINGLE.

numDomains

An Int specifying the number of domains for parallel execution in Abaqus/Explicit. When using more than 1 *numCpus*, *numDomains* must be a multiple of *numCpus*. The default value is 1.

activateLoadBalancing

A Boolean specifying whether to activate dyanmic load balancing for jobs running on multiple processors with multiple domains in Abaqus/Explicit. The default value is OFF.

multiprocessingMode

A SymbolicConstant specifying whether an analysis is decomposed into threads or into multiple processes that communicate through a message passing interface (MPI). Possible values are DEFAULT, THREADS, and MPI. The default value is DEFAULT.

licenseType

A SymbolicConstant specifying the type of license type being used in the case of the DSLS SimUnit license model. Possible values are DEFAULT, TOKEN, and CREDIT. The default value is DEFAULT.

If the license model is not the DSLS SimUnit, the licenseType is not available.

Return value

A JobFromInputFile object.

Exceptions

AbaqusException

ValueError

If the user attempts to provide RESTART as a value to argument type:	ValueError: RESTART of input file job is not currently supported.

`setValues(...)`

This method modifies the JobFromInputFile object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *JobFromInputFile* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The JobFromInputFile object has members with the same names and descriptions as the arguments to the *JobFromInputFile* method.

In addition, the JobFromInputFile object can have the following members:

getMemoryFromAnalysis	A Boolean specifying whether to retrieve the recommended memory settings from the last datacheck or analysis run and use those values in subsequent submissions. The default value is ON.
analysis	A SymbolicConstant specifying whether the job will be analyzed by Abaqus/Standard or Abaqus/Explicit. Possible values are STANDARD, EXPLICIT, and UNKNOWN. If the object has the type JobFromInputFile, *analysis*=UNKNOWN.
status	A SymbolicConstant specifying the status of the analysis. Possible values are SUBMITTED, RUNNING, ABORTED, TERMINATED, COMPLETED, CHECK_RUNNING, and CHECK_COMPLETED. If the *message* member is empty, *status* is set to NONE.

messages

A *MessageArray* object specifying the messages received during an analysis.

environment

A tuple of Strings specifying the environment variables and their values.

Message object

The Message object contains information about a given phase of the simulation. Job messages are not returned if a script is run without the Abaqus/CAE GUI (using the `noGUI` option).

Access

```
import job
mdb.coexecutions[name].jobs[name].messages[i]
mdb.jobs[name].messages[i]
```

Members

The Message object has the following members:

type

A SymbolicConstant specifying the type of message. Possible values are:

- ABORTED
- ANY_JOB
- ANY_MESSAGE_TYPE
- COMPLETED
- END_STEP
- ERROR
- HEADING
- HEALER_JOB
- HEALER_TYPE
- INTERRUPTED
- ITERATION
- JOB_ABORTED
- JOB_COMPLETED
- JOB_INTERRUPTED
- JOB_SUBMITTED
- MONITOR_DATA
- ODB_FILE
- ODB_FRAME
- STARTED
- STATE_FRAME
- STATUS
- STEP
- WARNING

data

A Dictionary object specifying the data returned by the analysis product. The value depends on the message returned. For a list of the possible entries, see the members of *DataObject*.

ModelJob object

The ModelJob object defines a *Job* object that analyzes a model on a model database (MDB).

The ModelJob object is derived from the *Job* object.

Access

```
import job
mdb.adaptivityProcesses[name].job
mdb.jobs[name]
```

Job(...)

This method creates an analysis job using a model on a model database (MDB) for the model definition.

Path

mdb.Job

Required arguments

name

A String specifying the name of the new job. The name must be a valid Abaqus/CAE object name.

model

A String specifying the name of the model to be analyzed or a Model object specifying the model to be analyzed.

Optional arguments

description

A String specifying a description of the job.

type

A SymbolicConstant specifying the type of job. Possible values are ANALYSIS, SYNTAXCHECK, RECOVER, and RESTART. The default value is ANALYSIS.

If the object has the type *JobFromInputFile*, *type*=RESTART is not available.

queue

A String specifying the name of the queue to which to submit the job. The default value is an empty string.



Note:

You can use the *queue* argument when creating a Job object on a Windows workstation; however, remote queues are available only on Linux platforms.

waitHours

An Int specifying the number of hours to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitMinutes*. *waitHours* and *atTime* are mutually exclusive.

waitMinutes

An Int specifying the number of minutes to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitHours*. *waitMinutes* and *atTime* are mutually exclusive.

atTime

A String specifying the time at which to submit the job. If *queue* is empty, the string syntax must be valid for the Linux`at` command. If *queue* is set, the syntax must be valid according to the system administrator. The default value is an empty string.



Note:

You can use the *atTime* argument when creating a Job object on a Windows workstation; however, the `at` command is available only on Linux platforms.

echoPrint

A Boolean specifying whether an echo of the input data is printed. The default value is OFF.

contactPrint

A Boolean specifying whether contact constraint data are printed. The default value is OFF.

modelPrint

A Boolean specifying whether model definition data are printed. The default value is OFF.

historyPrint

A Boolean specifying whether history data are printed. The default value is OFF.

scratch

A String specifying the location of the scratch directory. The default value is an empty string.

userSubroutine

A String specifying the file containing the user's subroutine definitions. The default value is an empty string.

numCpus

An Int specifying the number of CPUs to use for this analysis if parallel processing is available. Possible values are *numCpus*> 0. The default value is 1.

memory

An Int specifying the amount of memory available to an Abaqus analysis. The value should be expressed in the units supplied in *memoryUnits*. The default value is 90.

memoryUnits

A SymbolicConstant specifying the units for the amount of memory used in an Abaqus analysis. Possible values are PERCENTAGE, MEGA_BYTES, and GIGA_BYTES. The default value is PERCENTAGE.

explicitPrecision

A SymbolicConstant specifying whether to use the double precision version of Abaqus/Explicit. Possible values are SINGLE, FORCE_SINGLE, DOUBLE, DOUBLE_CONSTRAINT_ONLY, and DOUBLE_PLUS_PACK. The default value is SINGLE.

nodalOutputPrecision

A SymbolicConstant specifying the precision of the nodal output written to the output database. Possible values are SINGLE and FULL. The default value is SINGLE.

numDomains

An Int specifying the number of domains for parallel execution in Abaqus/Explicit. When using more than 1 *numCpus*, *numDomains* must be a multiple of *numCpus*. The default value is 1.

activateLoadBalancing A Boolean specifying whether to activate dyanmic load balancing for jobs running on multiple processors with multiple domains in Abaqus/Explicit. The default value is OFF.

multiprocessingMode A SymbolicConstant specifying whether an analysis is decomposed into threads or into multiple processes that communicate through a message passing interface (MPI). Possible values are DEFAULT, THREADS, and MPI. The default value is DEFAULT.

licenseType A SymbolicConstant specifying the type of license type being used in the case of the DSLS SimUnit license model. Possible values are DEFAULT, TOKEN, and CREDIT. The default value is DEFAULT.

If the license model is not the DSLS SimUnit, the licenseType is not available.

Return value

A ModelJob object.

Exceptions

AbaqusException.

`writeInput(...)`

This method writes an input file.

Required arguments

None.

Optional arguments

consistencyChecking A Boolean specifying whether to perform consistency checking for the job. The default value is ON. It is not recommended that you turn off the consistency checking unless you are absolutely sure that the model is consistent.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the ModelJob object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ModelJob* method.

Return value

`None.`

Exceptions

None.

Members

The ModelJob object can have the following members:

name

A String specifying the name of the new job. The name must be a valid Abaqus/CAE object name.

echoPrint

A Boolean specifying whether an echo of the input data is printed. The default value is OFF.

contactPrint

A Boolean specifying whether contact constraint data are printed. The default value is OFF.

modelPrint

A Boolean specifying whether model definition data are printed. The default value is OFF.

historyPrint

A Boolean specifying whether history data are printed. The default value is OFF.

model

A String specifying the name of the model to be analyzed or a Model object specifying the model to be analyzed.

description

A String specifying a description of the job.

type

A SymbolicConstant specifying the type of job. Possible values are ANALYSIS, SYNTAXCHECK, RECOVER, and RESTART. The default value is ANALYSIS.

If the object has the type *JobFromInputFile*, *type*=RESTART is not available.

waitHours

An Int specifying the number of hours to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitMinutes*. *waitHours* and *atTime* are mutually exclusive.

waitMinutes

An Int specifying the number of minutes to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitHours*. *waitMinutes* and *atTime* are mutually exclusive.

numCpus

An Int specifying the number of CPUs to use for this analysis if parallel processing is available. Possible values are *numCpus*> 0. The default value is 1.

memory

An Int specifying the amount of memory available to an Abaqus analysis. The value should be expressed in the units supplied in *memoryUnits*. The default value is 90.

memoryUnits

A SymbolicConstant specifying the units for the amount of memory used in an Abaqus analysis. Possible values are PERCENTAGE, MEGA_BYTES, and GIGA_BYTES. The default value is PERCENTAGE.

getMemoryFromAnalysis

A Boolean specifying whether to retrieve the recommended memory settings from the last datacheck or analysis run and use those values in subsequent submissions. The default value is ON.

explicitPrecision

A SymbolicConstant specifying whether to use the double precision version of Abaqus/Explicit. Possible values are SINGLE, FORCE_SINGLE, DOUBLE, DOUBLE_CONSTRAINT_ONLY, and DOUBLE_PLUS_PACK. The default value is SINGLE.

nodalOutputPrecision

A SymbolicConstant specifying the precision of the nodal output written to the output database. Possible values are SINGLE and FULL. The default value is SINGLE.

parallelizationMethodExplicit

A SymbolicConstant specifying the parallelization method for Abaqus/Explicit. This value is ignored for Abaqus/Standard. Possible values are DOMAIN and LOOP. The default value is DOMAIN.

numDomains

An Int specifying the number of domains for parallel execution in Abaqus/Explicit. When *parallelizationMethodExplicit*=DOMAIN, *numDomains* must be a multiple of *numCpus*. The default value is 1.

activateLoadBalancing

A Boolean specifying whether to activate dyanmic load balancing for jobs running on multiple processors with multiple domains in Abaqus/Explicit. The default value is OFF.

multiprocessingMode

A SymbolicConstant specifying whether an analysis is decomposed into threads or into multiple processes that communicate through a message passing interface (MPI). Possible values are DEFAULT, THREADS, and MPI. The default value is DEFAULT.

analysis

A SymbolicConstant specifying whether the job will be analyzed by Abaqus/Standard or Abaqus/Explicit. Possible values are STANDARD, EXPLICIT, and UNKNOWN.

If the object has the type *JobFromInputFile*, *analysis*=UNKNOWN.

status

A SymbolicConstant specifying the status of the analysis. Possible values are SUBMITTED, RUNNING, ABORTED, TERMINATED, COMPLETED, CHECK_RUNNING, and CHECK_COMPLETED.

If the *message* member is empty, *status* is set to NONE.

queue

A String specifying the name of the queue to which to submit the job. The default value is an empty string.



Note:

You can use the *queue* argument when creating a Job object on a Windows workstation; however, remote queues are available only on Linux platforms.

atTime

A String specifying the time at which to submit the job. If *queue* is empty, the string syntax must be valid for the Linux`at` command. If *queue* is set, the syntax must be valid according to the system administrator. The default value is an empty string.



Note:

You can use the *atTime* argument when creating a Job object on a Windows workstation; however, the `at` command is available only on Linux platforms.

scratch

A String specifying the location of the scratch directory. The default value is an empty string.

userSubroutine

A String specifying the file containing the user's subroutine definitions. The default value is an empty string.

messages

A *MessageArray* object specifying the messages received during an analysis.

environment

A tuple of Strings specifying the environment variables and their values.

Queue object

A Queue object tells the job where and how to submit a job remotely. A Queue object can be used as the *queue* argument to the `Job` method.

Access

```
import job
session.queues[name]
```

Queue(...)

This method creates a Queue object.



Note:

Remote queues are available only on Linux platforms.

Path

```
session.Queue
```

Required arguments

name	A String specifying the name of the new Queue object.
queueName	A String specifying the name of the remote analysis queue.

Optional arguments

hostName	A String specifying the name of the remote host. The default value is an empty string.
fileCopy	A Boolean specifying if the results files are to be copied from the remote machine to the local machine. The default value is ON.
directory	A String specifying the remote location for the execution of the simulation. The default value is an empty string.
driver	A String specifying the designation of the remote driver. The default value is "abaqus".
remotePlatform	A SymbolicConstant specifying the type of operating system on the remote machine. The default value is Linux.
filesToCopy	A list of Strings specifying the files to be copied from the remote location to the local machine, or ALL. Strings specified in a list are the extensions of the job files that will be copied, such as ('log', 'dat', 'msg', 'sta', 'odb'). The default value is ALL.
deleteAfterCopy	A Boolean specifying whether remote files are to be deleted after they are copied to the local machine. The default value is OFF.

description A String specifying a description of the queue. The default value is an empty string.

Return value

A Queue object.

Exceptions

If *fileCopy*=ON and *hostName* is empty: Remote queue host name is not set.

If *fileCopy*=ON and *directory* is empty: Directory in which to run the job on the remote computer is not set.

Members

The Queue object has members with the same names and descriptions as the arguments to the *Queue* method.

Coexecution object

The Coexecution object contains a set of jobs as associated parameters to define a co-simulation analysis.

Access

```
import job
mdb.coexecutions[name]
```

kill()

This method ends the analysis of a co-execution.

Arguments

None.

Return value

None.

Exceptions

None.

submit(...)

This method submits a co-execution for analysis.

Required arguments

None.

Optional arguments

consistencyChecking A Boolean specifying whether to perform consistency checking for the individual jobs. The default value is ON. It is not recommended to turn the consistency checking off unless you are absolutely sure the models are all consistent.

datacheckJob A Boolean specifying whether to run the co-execution as a datacheck analysis. The default value is False. The *datacheckJob* and *continueJob* arguments cannot both be True.

continueJob A Boolean specifying whether to run the co-execution as a continuation analysis. The default value is False. The *datacheckJob* and *continueJob* arguments cannot both be True.

Return value

None.

Exceptions

None.

writeInput(...)

This method writes an input file for each analysis in the co-execution.

Required arguments

None.

Optional arguments

consistencyChecking A Boolean specifying whether to perform consistency checking for the individual jobs. The default value is ON. It is not recommended to turn the consistency checking off unless you are absolutely sure the models are all consistent.

Return value

`None.`

Exceptions

None.

waitForCompletion()

This method interrupts the execution of the script until the end of all the analyses. If you call the `waitForCompletion` method and the *status* member is neither SUBMITTED nor RUNNING, Abaqus assumes the analysis has either completed or aborted and returns immediately.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Coexecution object can have the following members:

name A String specifying the name of the new job. The name must be a valid Abaqus/CAE object name.

type A SymbolicConstant specifying the type of analysis to execute for the co-execution. Possible values are ANALYSIS, SYNTAXCHECK, RECOVER, and RESTART. The default value is ANALYSIS.

mainAnalysisProduct A SymbolicConstant specifying the analysis product type of the main model for the co-execution. The default value is ABAQUS.

waitHours

An Int specifying the number of hours to wait before submitting the co-execution. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitMinutes*. *waitHours* and *atTime* are mutually exclusive.

waitMinutes

An Int specifying the number of minutes to wait before submitting the job. This argument is ignored if *queue* is set. The default value is 0.

This argument works in conjunction with *waitHours*. *waitMinutes* and *atTime* are mutually exclusive.

status

A SymbolicConstant specifying the status of the co-execution. Possible values are SUBMITTED, RUNNING, ABORTED, TERMINATED, COMPLETED, CHECK_SUBMITTED, CHECK_RUNNING, and CHECK_COMPLETED.

If the *message* member of all the jobs are empty, *status* is set to NONE.

queue

A String specifying the name of the queue to which to submit the co-execution. The default value is an empty string.



Note:

You can use the *queue* argument when creating a Coexecution object on a Windows workstation; however, remote queues are available only on Linux platforms.

atTime

A String specifying the time at which to submit the co-execution. If *queue* is empty, the string syntax must be valid for the Linux`at` command. If *queue* is set, the syntax must be valid according to the system administrator. The default value is an empty string.



Note:

You can use the *atTime* argument when creating a Coexecution object on a Windows workstation; however, the `at` command is available only on Linux platforms.

jobs

A repository of *Job* objects specifying the jobs that comprise this co-execution.

secondaryModels

A tuple of Strings specifying the names of the secondary models for the co-execution.

secondaryAnalysisProducts

A tuple of SymbolicConstants specifying the analysis product types of the secondary models for the co-execution. The default value is an empty sequence.

mainModel

A String specifying the name of the main model for the co-execution.

licenseType

A SymbolicConstant specifying the type of license type being used in case of DSLS SimUnit license model. Possible values are DEFAULT, TOKEN, and CREDIT. The default value is DEFAULT.

If the license model is not DSLS SimUnit then the licenseType is not available.

Corresponding analysis keywords

- *HEADING
- *PREPRINT

OptimizationProcess object

The OptimizationProcess object defines a process to perform an optimization of a model defined using an optimization task.

Access

```
import job
mdb.optimizationProcesses[name]
```

OptimizationProcess(...)

This method creates an OptimizationProcess object.

Path

```
mdb.OptimizationProcess
```

Required arguments

name A String specifying name of the optimization process.

model A String specifying name of the model to be used for the optimization process.

task A String specifying name of the optimization task to be used for the optimization process.

prototypeJob A String specifying name of the job to be used as the prototype for all analysis jobs run by the optimization process.

Optional arguments

description A String specifying a description of the optimization process.

maxDesignCycle An Int specifying the maximum number of allowed design cycles for the optimization process. The default value is 15.

dataSaveFrequency An Enum specifying whether Abaqus should save every iteration file in the optimization process or a selection of iteration files saved at a user-specified frequency. If you set *dataSaveFrequency*=OPT_DATASAVE_EVERY_CYCLE, Abaqus saves every iteration file; if you set *dataSaveFrequency*=OPT_DATASAVE_SPECIFY_CYCLE, Abaqus saves iteration files according to the frequency defined by the *saveEvery* parameter. The default value is OPT_DATASAVE_SPECIFY_CYCLE.

saveInitial A Boolean specifying whether the initial cycle should be saved when *dataSaveFrequency* is OPT_DATASAVE_SPECIFY_CYCLE. The default value is True.

saveFirst A Boolean specifying whether the first cycle should be saved when *dataSaveFrequency* is OPT_DATASAVE_SPECIFY_CYCLE. The default value is True.

saveLast
A Boolean specifying whether the last cycle should be saved when *dataSaveFrequency* is OPT_DATASAVE_SPECIFY_CYCLE. The default value is True.

saveEvery
An Int specifying every nth cycle iterations to be saved when *dataSaveFrequency* is OPT_DATASAVE_SPECIFY_CYCLE. Abaqus saves file iterations for every nth iteration after iteration 1; if you set *saveEvery*=3, Abaqus saves file iterations for cycles 1, 4, 7, and so on. The default value is None.

licenseType
A SymbolicConstant specifying the type of license type being used in the case of the DSLS SimUnit license model. Possible values are DEFAULT, TOKEN, and CREDIT. The default value is DEFAULT.

For optimization job submission, the licenseType options are available regardless of the license model.

Return value

An OptimizationProcess object.

Exceptions

AbaqusException.

writeParAndInputFiles()

This method allows you to write par and input files for an optimization task.

Arguments

None.

Return value

`None.`

Exceptions

None.

submit(...)

This method submits an optimization process.

Required arguments

None.

Optional arguments

validate
A Boolean specifying whether Abaqus should perform the validation of the optimization process only. The default value is False.

Return value
`None.`

Exceptions
None.

waitForCompletion()

This method interrupts the execution of the script until the end of all the analyses. If you call the `waitForCompletion` method and the *status* member is neither SUBMITTED nor RUNNING, Abaqus assumes the analysis has either completed or aborted and returns immediately.

Arguments
None.

Return value
`None.`

Exceptions
None.

extract(...)

This method extracts a surface mesh from the optimized model.

Required arguments

outputFileName	Name of the output file for the extracted surface mesh.
designCycle	The design cycle number for which the surface mesh should be extracted.

Optional arguments

isoValue	Value used to determine the positions on the element edges where the new nodes are created. Value between 0 and 1. The default value is 0.3.
smoothCycles	Number of smoothing cycles; if set to 0, no smoothing is performed. The default value is 5.
reductionPercent	Defines the percent of faces that should be removed during the data reduction. If set to 0, no data reduction occurs. If set to 100, the data reduction stops when no faces can be removed (that is checked using reductionAngle parameter). Value between 0 and 100. The default value is 0.
reductionAngle	Defines the maximal angle between adjacent faces at a node such that the node may be removed during the data reduction. Value in degrees between 0 and 90. The default value is15.

targetVolume Defines the target volume that is to be achieved iteratively by varying the isovalue. Value between 0 and 1. The default value is 0.

extractFormat A tuple for the types of format of the output. Possible values are OPT_EXTRACT_SMOOTH_ABAQUS_INPUT_FILE, OPT_EXTRACT_SMOOTH_STL, OPT_EXTRACT_SMOOTH_IGES1, OPT_EXTRACT_SMOOTH_IGES2, or OPT_EXTRACT_SMOOTH_IGES3. The default value is OPT_EXTRACT_SMOOTH_ABAQUS_INPUT_FILE.

resultFiltering Possible string values are OFF, MODERATE, or FULL. Defines if the element material values are to be filtered (averaged locally) before the isocut, and to what extent. The default value is OFF.

instances Defines a list of names of part instances to be used for surface extraction. One file (according to extractFormat) is created for each part instance. If the argument is not specified, no part instance is selected and surface is generated for the whole model.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the OptimizationProcess object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *OptimizationProcess* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The OptimizationProcess object has members with the same names and descriptions as the arguments to the *OptimizationProcess* method.

Load and Load Case commands

A specific type of load object and a specific type of load state object are designed for each type of load. A load object stores the nonpropagating data of a load as well as a number of instances of the corresponding load state object, each of which stores the propagating data of the load in a single step. Instances of the load state object are created and deleted internally by its corresponding load object.

Load Case commands are used for configuring load cases in specific types of steps that may use them.

In this section:

- *Load object*
- *LoadCase object*
- *LoadState object*
- *BodyCharge object*
- *BodyChargeState object*
- *BodyConcentrationFlux object*
- *BodyConcentrationFluxState object*
- *BodyCurrent object*
- *BodyCurrentDensity object*
- *BodyCurrentState object*
- *BodyForce object*
- *BodyForceState object*
- *BodyHeatFlux object*
- *BodyHeatFluxState object*
- *BoltLoad object*
- *BoltLoadState object*
- *ConcCharge object*
- *ConcConcFlux object*
- *ConcCurrent object*
- *ConcCurrentState object*
- *ConcentratedChargeState object*
- *ConcentratedConcentrationFluxState object*
- *ConcentratedForce object*
- *ConcentratedForceState object*
- *ConcentratedHeatFlux object*
- *ConcentratedHeatFluxState object*
- *ConcentratedPoreFluidState object*
- *ConcPoreFluid object*
- *ConnectorForce object*
- *ConnectorForceState object*
- *ConnectorMoment object*
- *ConnectorMomentState object*
- *CoriolisForce object*
- *CoriolisForceState object*

- *DistIonConcBodyFlux object*
- *DistIonConcBodyFluxState object*
- *FluidPressurePenetration object*
- *FluidPressurePenetrationState object*
- *FluidSurfaceCurrent object*
- *FluidSurfaceCurrentState object*
- *Gravity object*
- *GravityState object*
- *HydrostaticFluidFlowState object*
- *InertiaRelief object*
- *InertiaReliefState object*
- *InwardVolAccel object*
- *InwardVolAccelState object*
- *IonConcentrationFlux object*
- *IonConcentrationFluxState object*
- *LineLoad object*
- *LineLoadState object*
- *Moment object*
- *MomentState object*
- *PEGLoad object*
- *PEGLoadState object*
- *PipePressure object*
- *PipePressureState object*
- *Pressure object*
- *PressureState object*
- *RotationalBodyForce object*
- *RotationalBodyForceState object*
- *ShellEdgeLoad object*
- *ShellEdgeLoadState object*
- *SubmodelSB object*
- *SubmodelSBState object*
- *SubstructureLoad object*
- *SubstructureLoadState object*
- *SurfaceCharge object*
- *SurfaceChargeState object*
- *SurfaceConcentrationFlux object*
- *SurfaceConcentrationFluxState object*
- *SurfaceCurrent object*
- *SurfaceCurrentDensity object*
- *SurfaceCurrentState object*
- *SurfaceHeatFlux object*
- *SurfaceHeatFluxState object*
- *SurfacePoreFluid object*
- *SurfacePoreFluidState object*
- *SurfaceTraction object*
- *SurfaceTractionState object*

Load object

The Load object is the abstract base type for other Load objects. The Load object has no explicit constructor. The methods and members of the Load object are common to all objects derived from Load.

Access

```
import load
mdb.models[name].loads[name]
```

deactivate(...)

This method deactivates the load in the specified step and all its subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the load is deactivated.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

move(...)

This method moves the load state object from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the load state is moved.

toStepName A String specifying the name of the step to which the load state is moved.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

reset(...)

This method resets the load state of the specified step to the state of the previous general analysis step.

Required arguments

stepName	A String specifying the name of the step in which the load state is reset.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

resume()

This method resumes the load that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

suppress()

This method suppresses the load.

Arguments

None.

Return value

`None.`

Exceptions

None.

delete(...)

This method allows you to delete existing loads.

Required arguments

indices A sequence of Ints specifying the index of each load to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Load object can have the following members:

name A String specifying the load repository key.

region A *Region* object specifying the region to which the load is applied.

LoadCase object

The LoadCase object is used to define the loads and constraints comprising a particular loading condition and the linear response of a structure subjected to that loading condition.

Access

```
import step
mdb.models[name].steps[name].loadCases[name]
```

LoadCase(...)

This method creates a load case in a step.

Path

```
mdb.models[name].steps[name].LoadCase
```

Required arguments

name A String specifying the name of the object.

Optional arguments

boundaryConditions A sequence of (String, Float) sequences specifying the name of a BoundaryCondition followed by a nonzero Float scaling factor. The default value is an empty sequence.

loads A sequence of (String, Float) sequences specifying the name of a Load followed by a nonzero Float specifying a scale factor. The default value is an empty sequence.

includeActiveBaseStateBC A Boolean specifying whether to include all active boundary conditions propagated or modified from the base state. The default value is ON.

Return value

A LoadCase object.

Exceptions

RangeError.

resume()

This method resumes the load case that was previously suppressed.

Arguments

None.

Return value

```
None.
```

Exceptions

None.

suppress()

This method suppresses the load case.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the LoadCase object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LoadCase* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The LoadCase object has members with the same names and descriptions as the arguments to the *LoadCase* method.

In addition, the LoadCase object has the following member:

suppressed A Boolean specifying whether the load case is suppressed or not. The default value is OFF.

LoadState object

The LoadState object is the abstract base type for other LoadState objects. The LoadState object has no explicit constructor or methods. The members of the LoadState object are common to all objects derived from LoadState.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The LoadState object has the following members:

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the LoadState object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

BodyCharge object

The BodyCharge object stores the data for a body charge.

The BodyCharge object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyCharge(...)

This method creates a BodyCharge object.

Path

```
mdb.models[name].BodyCharge
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

Return value

A BodyCharge object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyCharge object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyCharge* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyCharge object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BodyCharge object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

BodyChargeState object

The BodyChargeState object stores the propagating data of a body charge in a step. One instance of this object is created internally by the *BodyCharge* object for each step. The instance is also deleted internally by the *BodyCharge* object.

The BodyChargeState object has no constructor or methods.

The BodyChargeState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BodyChargeState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DECHARGE* (load type label: EBF)

BodyConcentrationFlux object

The BodyConcentrationFlux object defines body concentration flux from a region or into a region.

The BodyConcentrationFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyConcentrationFlux(...)

This method creates a BodyConcentrationFlux object.

Path

```
mdb.models[name].BodyConcentrationFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the body concentration flux magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
distributionType	A SymbolicConstant specifying how the body concentration flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A BodyConcentrationFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyConcentrationFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyConcentrationFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyConcentrationFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the Body heat flux is modified.

Optional arguments

magnitude A Float specifying the Body concentration flux magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BodyConcentrationFlux object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the body concentration flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

BodyConcentrationFluxState object

The BodyConcentrationFluxState object stores the propagating data for a *BodyConcentrationFlux* object in a step. One instance of this object is created internally by the *BodyConcentrationFlux* object for each step. The instance is also deleted internally by the *BodyConcentrationFlux* object.

The BodyConcentrationFluxState object has no constructor or methods.

The BodyConcentrationFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BodyConcentrationFluxState object has the following members:

magnitude

A Float specifying the body concentration flux magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the body concentration flux magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CFLUX*

BodyCurrent object

The BodyCurrent object stores the data for a body current.

The BodyCurrent object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyCurrent(...)

This method creates a BodyCurrent object.

Path

```
mdb.models[name].BodyCurrent
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

Return value

A BodyCurrent object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyCurrent object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyCurrent* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyCurrent object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BodyCurrent object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

BodyCurrentDensity object

The BodyCurrentDensity object stores the data for a body current.

The BodyCurrentDensity object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyCurrentDensity(...)

This method creates a BodyCurrentDensity object.

Path

```
mdb.models[name].BodyCurrentDensity
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
comp1	A Complex specifying the first component of the load.
comp2	A Complex specifying the second component of the load.
comp3	A Complex specifying the third component of the load.

Optional arguments

amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

Return value

A BodyCurrentDensity object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyCurrentDensity object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyCurrentDensity* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyCurrentDensity object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Complex specifying the first component of the load.

comp2 A Complex specifying the second component of the load.

comp3 A Complex specifying the third component of the load.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BodyCurrentDensity object can have the following members:

name

A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

region

A *Region* object specifying the region to which the load is applied.

BodyCurrentState object

The BodyCurrentState object stores the propagating data of a body current in a step. One instance of this object is created internally by the *BodyCurrent* object for each step. The instance is also deleted internally by the *BodyCurrent* object.

The BodyCurrentState object has no constructor or methods.

The BodyCurrentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BodyCurrentState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DECURRENT* (load type label: CBF)

BodyForce object

The BodyForce object defines a distributed load.

The BodyForce object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyForce(...)

This method creates a BodyForce object.

Path

```
mdb.models[name].BodyForce
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

region
A *Region* object specifying the region to which the load is applied.

Optional arguments

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

comp1
A Float or a Complex specifying the body force component in the 1-direction.



Note:

Although *comp1*, *comp2*, and *comp3* are optional arguments, at least one of them must be nonzero unless *distributionType*=USER_DEFINED.

comp2
A Float or a Complex specifying the body force component in the 2-direction.

comp3
A Float or a Complex specifying the body force component in the 3-direction.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A BodyForce object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyForce object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyForce* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyForce object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the body force component in the 1-direction. UNCHANGED should be used if the body force component is propagated from the previous analysis step.

comp2 A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the body force component in the 2-direction. UNCHANGED should be used if the body force component is propagated from the previous analysis step.

comp3 A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the body force component in the 3-direction. UNCHANGED should be used if the body force component is propagated from the previous analysis step.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the

amplitude is propagated from the previous analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

```
None.
```

Exceptions

None.

Members

The BodyForce object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

Corresponding analysis keywords

- **DLOAD* (load type label: BX, BY, BZ, BR, BXNU, BYNU, BZNU, or BRNU)

BodyForceState object

The BodyForceState object stores the propagating data of a body force in a step. One instance of this object is created internally by the *BodyForce* object for each step. The instance is also deleted internally by the *BodyForce* object.

The BodyForceState object has no constructor or methods.

The BodyForceState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BodyForceState object has the following members:

comp1

A Float or a Complex specifying the body force component in the 1-direction.

comp2

A Float or a Complex specifying the body force component in the 2-direction.

comp3

A Float or a Complex specifying the body force component in the 3-direction.

comp1State

A SymbolicConstant specifying the propagation state of the body force component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

comp2State

A SymbolicConstant specifying the propagation state of the body force component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

comp3State

A SymbolicConstant specifying the propagation state of the body force component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DLOAD* (load type label: BX, BY, BZ, BR, BXNU, BYNU, BZNU, or BRNU)

BodyHeatFlux object

The BodyHeatFlux object defines body heat flux from a region or into a region.

The BodyHeatFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BodyHeatFlux(...)

This method creates a BodyHeatFlux object.

Path

```
mdb.models[name].BodyHeatFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the body heat flux magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
distributionType	A SymbolicConstant specifying how the body heat flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A BodyHeatFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing BodyHeatFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BodyHeatFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BodyHeatFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the body heat flux is modified.

Optional arguments

magnitude A Float specifying the body heat flux magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BodyHeatFlux object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the body heat flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

BodyHeatFluxState object

The BodyHeatFluxState object stores the propagating data for a Body *BodyHeatFlux* object in a step. One instance of this object is created internally by the *BodyHeatFlux* object for each step. The instance is also deleted internally by the *BodyHeatFlux* object.

The BodyHeatFluxState object has no constructor or methods.

The BodyHeatFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BodyHeatFluxState object has the following members:

magnitude

A Float specifying the Body heat flux magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the Body heat flux magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DFLUX*

BoltLoad object

The BoltLoad object defines a bolt load.

The BoltLoad object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

BoltLoad(...)

This method creates a BoltLoad object.

Path

```
mdb.models[name].BoltLoad
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created. This must be the first analysis step name.

region
A *Region* object specifying the region to which the load is applied.

magnitude
A Float specifying the bolt load magnitude.

Optional arguments

boltMethod
A SymbolicConstant specifying the method of applying the bolt load. Possible values are APPLY_FORCE and ADJUST_LENGTH. The default value is APPLY_FORCE.

datumAxis
A *DatumAxis* object specifying the orientation of the pre-tension section normal.



Note: *datumAxis* is applicable only for Solid and Shell regions; it has no meaning for Wire regions.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

preTenSecPartLevel
A Boolean specifying whether the pre-tension section is to be defined at the part level. The default value is False. You should provide the *preTenSecPartLevel* argument only if the selected region belongs to a dependent part instance. A pre-tension section cannot be defined at the part level for independent and model instances.

Return value

A BoltLoad object.

Exceptions

TextError.

setValues(...)

This method modifies the data for an existing BoltLoad object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BoltLoad* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing BoltLoad object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

boltMethod A SymbolicConstant specifying the type of bolt load. Possible values are APPLY_FORCE, ADJUST_LENGTH, and FIX_LENGTH. The default is APPLY_FORCE.

magnitude A Float specifying the bolt load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The BoltLoad object can have the following members:

name

A String specifying the load repository key.

datumAxis

A *DatumAxis* object specifying the orientation of the pre-tension section normal.



Note:

datumAxis is required only for Solid and Shell regions; it has no meaning for Wire regions.

region

A *Region* object specifying the region to which the load is applied.

Corresponding analysis keywords

- **PRE-TENSION SECTION*
- **NODE* (for the reference node)
- **NSET* (for the reference node)

BoltLoadState object

The BoltLoadState object stores the propagating data of a bolt load in a step. One instance of this object is created internally by the *BoltLoad* object for each step. The instance is also deleted internally by the *BoltLoad* object.

The BoltLoadState object has no constructor or methods.

The BoltLoadState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The BoltLoadState object has the following members:

boltMethod

A SymbolicConstant specifying the type of bolt load. Possible values are APPLY_FORCE, ADJUST_LENGTH, and FIX_LENGTH.

boltMethodState

A SymbolicConstant specifying the propagation state of the bolt load type. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

magnitude

A Float specifying the bolt load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the bolt load magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (when *boltMethod*=APPLY_FORCE)
- **BOUNDARY* (when *boltMethod*=ADJUST_LENGTH or FIX_LENGTH)

ConcCharge object

The ConcCharge object stores the data for a concentrated charge.

The ConcCharge object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcCharge(...)

This method creates a ConcCharge object.

Path

```
mdb.models[name].ConcCharge
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConcCharge object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConcCharge object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcCharge* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConcCharge object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcCharge object can have the following members:

name A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

ConcConcFlux object

The ConcConcFlux object stores the data for a concentrated concentration flux.

The ConcConcFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcConcFlux(...)

This method creates a ConcConcFlux object.

Path

```
mdb.models[name].ConcConcFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConcConcFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConcConcFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcConcFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConcConcFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcConcFlux object can have the following members:

name A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

ConcCurrent object

The ConcCurrent object stores the data for a concentrated current.

The ConcCurrent object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcCurrent(...)

This method creates a ConcCurrent object.

Path

```
mdb.models[name].ConcCurrent
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConcCurrent object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConcCurrent object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcCurrent* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConcCurrent object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcCurrent object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

ConcCurrentState object

The ConcCurrentState object stores the propagating data of a concentrated current in a step. One instance of this object is created internally by the *ConcCurrent* object for each step. The instance is also deleted internally by the *ConcCurrent* object.

The ConcCurrentState object has no constructor or methods.

The ConcCurrentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcCurrentState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CECURRENT*

ConcentratedChargeState object

The ConcentratedChargeState object stores the propagating data of a concentrated charge in a step. One instance of this object is created internally by the *ConcCharge* object for each step. The instance is also deleted internally by the *ConcCharge* object.

The ConcentratedChargeState object has no constructor or methods.

The ConcentratedChargeState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcentratedChargeState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CECHARGE*

ConcentratedConcentrationFluxState object

The ConcentratedConcentrationFluxState object stores the propagating data of a concentrated concentration flux in a step. One instance of this object is created internally by the *ConcConcFlux* object for each step. The instance is also deleted internally by the *ConcConcFlux* object.

The ConcentratedConcentrationFluxState object has no constructor or methods.

The ConcentratedConcentrationFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcentratedConcentrationFluxState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CFLUX* (degree of freedom: 11, which is assumed)

ConcentratedForce object

The ConcentratedForce object defines a concentrated force.

The ConcentratedForce object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcentratedForce(...)

This method creates a ConcentratedForce object.

Path

```
mdb.models[name].ConcentratedForce
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

Optional arguments

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

cf1 A Float or a Complex specifying the concentrated force component in the 1-direction. Although *cf1*, *cf2*, and *cf3* are optional arguments, at least one of them must be nonzero.

cf2 A Float or a Complex specifying the concentrated force component in the 2-direction.

cf3 A Float or a Complex specifying the concentrated force component in the 3-direction.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

follower A Boolean specifying whether the direction of the force rotates with the rotation at each node of the region. You should provide the *follower* argument only if it is valid for the specified step. The default value is OFF.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the load's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

Return value

A ConcentratedForce object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConcentratedForce object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcentratedForce* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing ConcentratedForce object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

cf1 A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the concentrated force component in the 1-direction. UNCHANGED should be used if the concentrated force component is propagated from the previous analysis step.

cf2 A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the concentrated force component in the 2-direction. UNCHANGED should be used if the concentrated force component is propagated from the previous analysis step.

cf3
A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the concentrated force component in the 3-direction. UNCHANGED should be used if the concentrated force component is propagated from the previous analysis step.

amplitude
A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value
`None`.

Exceptions
None.

Members
The ConcentratedForce object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

follower
A Boolean specifying whether the direction of the force rotates with the rotation at each node of the region. You should provide the *follower* argument only if it is valid for the specified step. The default value is OFF.

localCsys
`None` or a *DatumCsys* object specifying the local coordinate system of the load's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

ConcentratedForceState object

The ConcentratedForceState object stores the propagating data for a concentrated force in a step. One instance of this object is created internally by the *ConcentratedForce* object for each step. The instance is also deleted internally by the *ConcentratedForce* object.

The ConcentratedForceState object has no constructor or methods.

The ConcentratedForceState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcentratedForceState object has the following members:

cf1

A Float or a Complex specifying the concentrated force component in the 1-direction. Although *cf1*, *cf2*, and *cf3* are optional arguments, at least one of them must be nonzero.

cf2

A Float or a Complex specifying the concentrated force component in the 2-direction.

cf3

A Float or a Complex specifying the concentrated force component in the 3-direction.

cf1State

A SymbolicConstant specifying the propagation state of the concentrated force component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

cf2State

A SymbolicConstant specifying the propagation state of the concentrated force component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

cf3State

A SymbolicConstant specifying the propagation state of the concentrated force component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (degree of freedom: 1, 2, or 3)

ConcentratedHeatFlux object

The ConcentratedHeatFlux object stores the data for a concentrated heat flux load.

The ConcentratedHeatFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcentratedHeatFlux(...)

This method creates a ConcentratedHeatFlux object.

Path

```
mdb.models[name].ConcentratedHeatFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
dof	An Int specifying the degree of freedom of the node, to which the concentrated heat flux should be applied. The default value is 11.

Return value

A ConcentratedHeatFlux object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ConcentratedHeatFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcentratedHeatFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing ConcentratedHeatFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcentratedHeatFlux object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

dof
An Int specifying the degree of freedom of the node, to which the concentrated heat flux should be applied. The default value is 11.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

ConcentratedHeatFluxState object

The ConcentratedHeatFluxState object stores the propagating data of a concentrated heat flux load in a step. One instance of this object is created internally by the *ConcentratedHeatFlux* object for each step. The instance is also deleted internally by the *ConcentratedHeatFlux* object.

The ConcentratedHeatFluxState object has no constructor or methods.

The ConcentratedHeatFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcentratedHeatFluxState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CFLUX* (degree of freedom: one out of integers 11 to 31)

ConcentratedPoreFluidState object

The ConcentratedPoreFluidState object stores the propagating data of a concentrated pore fluid flow load in a step. One instance of this object is created internally by the *ConcPoreFluid* object for each step. The instance is also deleted internally by the *ConcPoreFluid* object.

The ConcentratedPoreFluidState object has no constructor or methods.

The ConcentratedPoreFluidState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConcentratedPoreFluidState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (degree of freedom: 8)

ConcPoreFluid object

The ConcPoreFluid object stores the data for a concentrated pore fluid flow load.

The ConcPoreFluid object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConcPoreFluid(...)

This method creates a ConcPoreFluid object.

Path

```
mdb.models[name].ConcPoreFluid
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created. This must be the first analysis step name.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float specifying the load magnitude.

Optional arguments

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConcPoreFluid object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConcPoreFluid object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcPoreFluid* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConcPoreFluid object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConcPoreFluid object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

ConnectorForce object

The ConnectorForce object defines a connector force.

The ConnectorForce object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConnectorForce(...)

This method creates a ConnectorForce object on a wire region. Alternatively, the load may also be applied to a wire set referenced from an assembled fastener template model.

Path

```
mdb.models[name].ConnectorForce
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

Optional arguments

region
The wire region to which the load is applied. This argument is not valid when *fastenerName* and *fastenerSetName* are specified.

fastenerName
A String specifying the name of the assembled fastener to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName
A String specifying the assembled fastener template model set to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

f1
A Float or a Complex specifying the connector force component in the connector's local 1-direction.



Note:

Although *f1*, *f2*, and *f3* are optional arguments, at least one of them must be nonzero.

f2
A Float or a Complex specifying the connector force component in the connector's local 2-direction.

f3
A Float or a Complex specifying the connector force component in the connector's local 3-direction.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConnectorForce object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConnectorForce object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorForce* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConnectorForce object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the load is modified.

Optional arguments

f1
A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the connector force component in the connector's local 1-direction. UNCHANGED should be used if the connector force component is propagated from the previous analysis step.

f2
A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the connector force component in the connector's local 2-direction. UNCHANGED should be used if the connector force component is propagated from the previous analysis step.

f3
A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the connector force component in the connector's local 3-direction. UNCHANGED should be used if the connector force component is propagated from the previous analysis step.

amplitude
A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value
`None.`

Exceptions
None.

Members
The ConnectorForce object can have the following members:

name
A String specifying the load repository key.

fastenerName
A String specifying the name of the assembled fastener to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName
A String specifying the assembled fastener template model set to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

ConnectorForceState object

The ConnectorForceState object stores the propagating data for a connector force in a step. One instance of this object is created internally by the *ConnectorForce* object for each step. The instance is also deleted internally by the *ConnectorForce* object.

The ConnectorForceState object has no constructor or methods.

The ConnectorForceState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConnectorForceState object has the following members:

f1

A Float or a Complex specifying the connector force component in the connector's local 1-direction.

f2

A Float or a Complex specifying the connector force component in the connector's local 2-direction.

f3

A Float or a Complex specifying the connector force component in the connector's local 3-direction.

f1State

A SymbolicConstant specifying the propagation state of the connector force component in the connector's local 1-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

f2State

A SymbolicConstant specifying the propagation state of the connector force component in the connector's local 2-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

f3State

A SymbolicConstant specifying the propagation state of the connector force component in the connector's local 3-direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CONNECTOR LOAD* (degree of freedom: 1, 2, or 3)

ConnectorMoment object

The ConnectorMoment object stores the data for a connector moment.

The ConnectorMoment object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ConnectorMoment(...)

This method creates a ConnectorMoment object on a wire region. Alternatively, the load may also be applied to a wire set referenced from an assembled fastener template model.

Path

```
mdb.models[name].ConnectorMoment
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

Optional arguments

region The wire region to which the load is applied. This argument is not valid when *fastenerName* and *fastenerSetName* are specified.

fastenerName A String specifying the name of the assembled fastener to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName A String specifying the assembled fastener template model set to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

m1 A Float or a Complex specifying the moment component in the connector's local 4-direction.

m2 A Float or a Complex specifying the moment component in the connector's local 5-direction.

m3 A Float or a Complex specifying the moment component in the connector's local 6-direction.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A ConnectorMoment object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing ConnectorMoment object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorMoment* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ConnectorMoment object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the load is modified.

Optional arguments

m1
A Float, a Complex, or a SymbolicConstant specifying the moment component in the connector's local 4-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

m2
A Float, a Complex, or a SymbolicConstant specifying the moment component in the connector's local 5-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

m3
A Float, a Complex, or a SymbolicConstant specifying the moment component in the connector's local 6-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ConnectorMoment object can have the following members:

name A String specifying the load repository key.

fastenerName A String specifying the name of the assembled fastener to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerSetName* must also be specified. The default value is an empty string.

fastenerSetName A String specifying the assembled fastener template model set to which the load will be applied. This argument is not valid when *region* is specified. When this argument is specified, *fastenerName* must also be specified. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

ConnectorMomentState object

The ConnectorMomentState object stores the propagating data for a connector moment in a step. One instance of this object is created internally by the *ConnectorMoment* object for each step. The instance is also deleted internally by the *ConnectorMoment* object.

The ConnectorMomentState object has no constructor or methods.

The ConnectorMomentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ConnectorMomentState object has the following members:

m1

A Float or a Complex specifying the connector moment component in the connector's local 4-direction. Although *m1*, *m2*, and *m3* are optional arguments, at least one of them must be nonzero.

m2

A Float or a Complex specifying the connector moment component in the connector's local 5direction.

m3

A Float or a Complex specifying the connector moment component in the connector's local 6-direction.

m1State

A SymbolicConstant specifying the propagation state of the load component in the connector's local 4-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

m2State

A SymbolicConstant specifying the propagation state of the load component in the connector's local 5-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

m3State

A SymbolicConstant specifying the propagation state of the load component in the connector's local 6-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE

- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CONNECTOR LOAD* (degree of freedom: 4, 5, or 6)

CoriolisForce object

The CoriolisForce object stores the data for a coriolis force.

The CoriolisForce object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

CoriolisForce(...)

This method creates a CoriolisForce object.

Path

```
mdb.models[name].CoriolisForce
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.
point1	A sequence of Floats specifying the first point on the axis of rotation for the load.
point2	A sequence of Floats specifying the second point on the axis of rotation for the load.

Optional arguments

amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.
distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

Return value

A CoriolisForce object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing CoriolisForce object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CoriolisForce* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing CoriolisForce object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The CoriolisForce object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

point1
A tuple of Floats specifying the first point on the axis of rotation for the load.

point2
A tuple of Floats specifying the second point on the axis of rotation for the load.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

CoriolisForceState object

The CoriolisForceState object stores the propagating data of a coriolis force in a step. One instance of this object is created internally by the *CoriolisForce* object for each step. The instance is also deleted internally by the *CoriolisForce* object.

The CoriolisForceState object has no constructor or methods.

The CoriolisForceState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The CoriolisForceState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DLOAD* (load type label: CORIO)

DistIonConcBodyFlux object

The DistIonConcBodyFlux object defines distributed ion concentration body flux from a region or into a region.

The DistIonConcBodyFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

DistIonConcBodyFlux(...)

This method creates a DistIonConcBodyFlux object.

Path

```
mdb.models[name].DistIonConcBodyFlux
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

region
A *Region* object specifying the region to which the load is applied.

magnitude
A Float specifying the body heat flux magnitude. *magnitude* is optional if *distributionType* is USER_DEFINED.

Optional arguments

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* is FIELD. The default value is an empty string.

distributionType
A SymbolicConstant specifying how the distributed ion concentration body flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A DistIonConcBodyFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing DistIonConcBodyFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DistIonConcBodyFlux* method, except for the *name* and *createStepName* arguments.

Return value

```
None.
```

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing DistIonConcBodyFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the body heat flux is modified.

Optional arguments

magnitude A Float specifying the body heat flux magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

```
None.
```

Exceptions

None.

Members

The DistIonConcBodyFlux object can have the following members:

name A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the distributed ion concentration body flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* = FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

DistIonConcBodyFluxState object

The DistIonConcBodyFluxState object stores the propagating data for a DistIonConcBodyFlux object in a step. One instance of this object is created internally by the *DistIonConcBodyFlux* object for each step. The instance is also deleted internally by the *DistIonConcBodyFlux* object.

The DistIonConcBodyFluxState object has no constructor or methods.

The DistIonConcBodyFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The DistIonConcBodyFluxState object has the following members:

magnitude

A Float specifying the distributed ion concentration body flux magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the distributed ion concentration body flux magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DFLUX*

FluidPressurePenetration object

The FluidPressurePenetration object defines a fluid pressure penetration load.

The FluidPressurePenetration object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

FluidPressurePenetration(...)

This method creates a FluidPressurePenetration object.

Path

```
mdb.models[name].FluidPressurePenetration
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the pressure is created.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float specifying the fluid pressure magnitude.

Optional arguments

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

criticalPressure A Float specifying the critical contact interface pressure threshold at which fluid penetration starts to occur. The default value is 0.0.

timePeriod A Float specifying the time period over which the fluid pressure on the newly penetrated contact surface segments is ramped up to the current magnitude. The *timePeriod* argument applies only in Abaqus/Standard and is neglected in the linear perturbation analysis. The default value is 0.001.

evolutionRegionMethod A SymbolicConstant LOCAL or WETTING_ADVANCE used for specifying the algorithm to control the evolution of the region exposed to the fluid pressure. The *evolutionRegionMethod* argument applies WETTING_ADVANCE only in an Abaqus/Explicit analysis. The default value is LOCAL.

wettedRegion A *Region* object specifying the node or node set on the surface initially exposed to fluid pressure. The *wettedRegion* argument applies only when the *evolutionRegionMethod* argument has the value WETTING_ADVANCE.

Return value

A FluidPressurePenetration object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing FluidPressurePenetration object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Pressure* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing Pressure object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the fluid pressure magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

criticalPressure A Float specifying the critical contact interface pressure threshold at which fluid penetration starts to occur. The default value is 0.0.

timePeriod A Float specifying the time period over which the fluid pressure on the newly penetrated contact surface segments is ramped up to the current magnitude. The *timePeriod* argument applies only in Abaqus/Standard and is neglected in the linear perturbation analysis. The default value is 0.001.

evolutionRegionMethod A SymbolicConstant LOCAL or WETTING_ADVANCE used for specifying the algorithm to control the evolution of the region exposed to the fluid pressure. The *evolutionRegionMethod* argument applies WETTING_ADVANCE only in an Abaqus/Explicit analysis. The default value is LOCAL.

wettedRegion A *Region* object specifying the node or node set on the surface initially exposed to fluid pressure. The *wettedRegion* argument applies only when the *evolutionRegionMethod* argument has the value WETTING_ADVANCE.

Return value

`None.`

Exceptions

None.

Members

The Pressure object can have the following members:

name A String specifying the load repository key.

region A *Region* object specifying the region to which the load is applied.

FluidPressurePenetrationState object

The FluidPressurePenetrationState object stores the propagating data for a pressure in a step.
One instance of this object is created internally by the *FluidPressurePenetration* object for each step. The instance is also deleted internally by the *FluidPressurePenetration* object.

The FluidPressurePenetrationState object has no constructor or methods.

The FluidPressurePenetrationState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The FluidPressurePenetrationState object has the following members:

magnitude
A Float specifying the pressure magnitude.

magnitudeState
A SymbolicConstant specifying the propagation state of the pressure magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitude
A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

amplitudeState
A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

criticalPressure
A Float specifying the critical contact interface pressure threshold at which fluid penetration starts to occur. The default value is 0.0.

criticalPressureState
A SymbolicConstant specifying the propagation state of the criticalPressure member. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

timePeriod
A Float specifying the time period over which the fluid pressure on the newly penetrated contact surface segments is ramped up to the current magnitude. The *timePeriod* argument applies only in *Abaqus/Standard* and is neglected in the linear perturbation analysis. The default value is 0.001.

timePeriodState
A SymbolicConstant specifying the propagation state of the timePeriod member. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

evolutionRegionMethod
A SymbolicConstant LOCAL or WETTING_ADVANCE used for specifying the algorithm to control the evolution of the region exposed to the fluid pressure. The *evolutionRegionMethod* argument applies WETTING_ADVANCE only in an *Abaqus/Explicit* analysis. The default value is LOCAL.

evolutionRegionMethodState A SymbolicConstant specifying the propagation state of the evolutionRegionMethod member. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

wettedRegion A *Region* object specifying the node or node set on the surface initially exposed to fluid pressure. The *wettedRegion* argument applies only when the *evolutionRegionMethod* argument has the value WETTING_ADVANCE.

wettedRegionState A SymbolicConstant specifying the propagation state of the wettedRegion member. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

status A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

Corresponding analysis keywords

- **DSLOAD*
- **DLOAD*

FluidSurfaceCurrent object

The FluidSurfaceCurrent object stores the data for a fluid surface current.

The FluidSurfaceCurrent object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

FluidSurfaceCurrent(...)

This method creates a FluidSurfaceCurrent object.

Path

```
mdb.models[name].FluidSurfaceCurrent
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* is FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A FluidSurfaceCurrent object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing FluidSurfaceCurrent object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidSurfaceCurrent* method, except for the *name* and *createStepName* arguments.

Return value

```
None.
```

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing FluidSurfaceCurrent object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

```
None.
```

Exceptions

None.

Members

The FluidSurfaceCurrent object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* is FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

FluidSurfaceCurrentState object

The FluidSurfaceCurrentState object stores the propagating data of a surface current in a step.
One instance of this object is created internally by the *FluidSurfaceCurrent* object for each step. The instance is also deleted internally by the *FluidSurfaceCurrent* object.

The FluidSurfaceCurrentState object has no constructor or methods.

The FluidSurfaceCurrentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The FluidSurfaceCurrentState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSECURRENT* (load type label: ECS)

Gravity object

The Gravity object stores the data of a gravity load.

The Gravity object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

Gravity(...)

This method creates a Gravity object.

Path

```
mdb.models[name].Gravity
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

Optional arguments

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

comp1 A Float or a Complex specifying the component of the load in the 1-direction.



Note:

Although *comp1*, *comp2*, and *comp3* are optional arguments, at least one of them must be nonzero.

comp2 A Float or a Complex specifying the component of the load in the 2-direction.

comp3 A Float or a Complex specifying the component of the load in the 3-direction.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A Gravity object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing Gravity object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Gravity* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing Gravity object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Float, a Complex, or a SymbolicConstant specifying the load component in the 1-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

comp2 A Float, a Complex, or a SymbolicConstant specifying the load component in the 2-direction. For details see *comp1*.

comp3 A Float, a Complex, or a SymbolicConstant specifying the load component in the 3-direction. For details see *comp1*.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load

is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The Gravity object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

GravityState object

The GravityState object stores the propagating data for a gravity load in a step. One instance of this object is created internally by the *Gravity* object for each step. The instance is also deleted internally by the *Gravity* object.

The GravityState object has no constructor or methods.

The GravityState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The GravityState object has the following members:

comp1

A Float or a Complex specifying the load component in the 1-direction.

comp2

A Float or a Complex specifying the load component in the 2-direction.

comp3

A Float or a Complex specifying the load component in the 3-direction.

comp1State

A SymbolicConstant specifying the propagation state of the load component in the 1-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp2State

A SymbolicConstant specifying the propagation state of the load component in the 2-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp3State

A SymbolicConstant specifying the propagation state of the load component in the 3-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DLOAD* (load type label:GRAV)

HydrostaticFluidFlowState object

The HydrostaticFluidFlowState object stores the propagating data for a concentrated HydrostaticFluidFlow object in a step. One instance of this object is created internally by the HydrostaticFluidFlow object for each step. The instance is also deleted internally by the HydrostaticFluidFlow object.

The HydrostaticFluidFlowState object has no constructor or methods.

The HydrostaticFluidFlowState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The HydrostaticFluidFlowState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **FLUID FLUX*

InertiaRelief object

The InertiaRelief object defines an inertia relief load.

The InertiaRelief object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

InertiaRelief(...)

This method creates an InertiaRelief object.

Path

```
mdb.models[name].InertiaRelief
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

Optional arguments

u1 A Boolean specifying the 1-direction as a free direction.



Note:

Although *u1*, *u2*, *u3*, *ur1*, *ur2*, and *ur3* are optional arguments, at least one of them must be specified. Further, any specified set of free directions cannot include only two rotational degrees of freedom.

u2 A Boolean specifying the 2-direction as a free direction.

u3 A Boolean specifying the 3-direction as a free direction.

ur1 A Boolean specifying the rotation about the 1–direction as a free direction.

ur2 A Boolean specifying the rotation about the 2–direction as a free direction.

ur3 A Boolean specifying the rotation about the 3–direction as a free direction.

referencePoint A sequence of Floats specifying the *X*, *Y* and *Z*-coordinates of a fixed rotation point or a point on the rotation axis or a point on the symmetry line, about which rotations are defined. Such a point must be specified only for certain combinations of free directions.

localCoordinates `None` or a *DatumCsys* object specifying the local coordinate system of the rigid body degrees of freedom for the inertia relief load. If *localCoordinates*=`None`, the free directions are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

Return value

An InertiaRelief object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing InertiaRelief object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *InertiaRelief* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing InertiaRelief object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

u1 A Boolean specifying the 1-direction as a free direction.

u2 A Boolean specifying the 2-direction as a free direction.

u3 A Boolean specifying the 3-direction as a free direction.

ur1 A Boolean specifying the rotation about the 1–direction as a free direction.

ur2
A Boolean specifying the rotation about the 2–direction as a free direction.

ur3
A Boolean specifying the rotation about the 3–direction as a free direction.

referencePoint
A sequence of Floats specifying the point about which rotations are defined. The point can be specified only for certain combinations of free directions. The *referencePoint* argument can be one of the following:

- The *X*, *Y* and *Z*-coordinates of a fixed rotation point.
- A point on the rotation axis.
- A point on the symmetry line.

fixed
A Boolean specifying whether the inertia relief loading should remain fixed at the current loading at the start of the step. The default value is OFF.

Return value
`None`.

Exceptions
None.

Members
The InertiaRelief object can have the following members:

name
A String specifying the load repository key.

localCoordinates
`None` or a *DatumCsys* object specifying the local coordinate system of the rigid body degrees of freedom for the inertia relief load. If *localCoordinates*=`None`, the free directions are defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

region
A *Region* object specifying the region to which the load is applied.

InertiaReliefState object

The InertiaReliefState object stores the propagating data for an inertia relief load in a step. One instance of this object is created internally by the *InertiaRelief* object for each step. The instance is also deleted internally by the *InertiaRelief* object.

The InertiaReliefState object has no constructor or methods.

The InertiaReliefState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The InertiaReliefState object has the following members:

u1

A Boolean specifying the 1-direction as a free direction.

u2

A Boolean specifying the 2-direction as a free direction.

u3

A Boolean specifying the 3-direction as a free direction.

ur1

A Boolean specifying the rotation about the 1–direction as a free direction.

ur2

A Boolean specifying the rotation about the 2–direction as a free direction.

ur3

A Boolean specifying the rotation about the 3–direction as a free direction.

u1State

A SymbolicConstant specifying the propagation state of the Boolean that identifies the local 1-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

u2State

A SymbolicConstant specifying the propagation state of the Boolean that identifies the local 2-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

u3State

A SymbolicConstant specifying the propagation state of the Boolean that identifies the local the 3-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

ur1State

A SymbolicConstant specifying the propagation state of the Boolean that identifies rotation about the local 1-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

ur2State

A SymbolicConstant specifying the propagation state of the Boolean that identifies the rotation about the local the 2-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

ur3State
A SymbolicConstant specifying the propagation state of the Boolean that identifies the rotation about the local the 3-direction as a free direction. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

fixed
A Boolean specifying whether the inertia relief loading should remain fixed at the current loading at the start of the step. The default value is OFF.

fixedState
A SymbolicConstant specifying the propagation state of the Boolean that identifies whether the inertia relief load should remain fixed at current level at the start of the step. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

referencePointState
A SymbolicConstant specifying the propagation state of the reference point of the inertia relief load. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

referencePoint
A tuple of Floats specifying the point about which rotations are defined. The point can be specified only for certain combinations of free directions. The *referencePoint* argument can be one of the following:

- The *X*, *Y* and *Z*-coordinates of a fixed rotation point.
- A point on the rotation axis.
- A point on the symmetry line.

amplitudeState
A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status
A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude
A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **INERTIA RELIEF*

InwardVolAccel object

The InwardVolAccel object stores the data for an inward volume acceleration acoustic load.

The InwardVolAccel object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

InwardVolAccel(...)

This method creates a InwardVolAccel object.

Path

```
mdb.models[name].InwardVolAccel
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the name of the first analysis step.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

An InwardVolAccel object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing InwardVolAccel object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *InwardVolAccel* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing InwardVolAccel object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The InwardVolAccel object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

InwardVolAccelState object

The InwardVolAccelState object stores the propagating data of an inward volume acceleration acoustic load in a step. One instance of this object is created internally by the *InwardVolAccel* object for each step. The instance is also deleted internally by the *InwardVolAccel* object.

The InwardVolAccelState object has no constructor or methods.

The InwardVolAccelState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The InwardVolAccelState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (degree of freedom: 8)

IonConcentrationFlux object

The IonConcentrationFlux object stores the data for an ion concentration flux load.

The IonConcentrationFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

IonConcentrationFlux(...)

This method creates an IonConcentrationFlux object.

Path

```
mdb.models[name].IonConcentrationFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* = FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A IonConcentrationFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing IonConcentrationFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IonConcentrationFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing IonConcentrationFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The IonConcentrationFlux object can have the following members:

name A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType* = FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

IonConcentrationFluxState object

The IonConcentrationFluxState object stores the propagating data of an ion concentration flux load in a step. One instance of this object is created internally by the *IonConcentrationFlux* object for each step. The instance is also deleted internally by the *IonConcentrationFlux* object.

The IonConcentrationFluxState object has no constructor or methods.

The IonConcentrationFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The IonConcentrationFluxState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CFLUX* (degree of freedom: integer 33)

LineLoad object

The LineLoad object stores the data of an applied line load.

The LineLoad object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

LineLoad(...)

This method creates a LineLoad object.

Path

```
mdb.models[name].LineLoad
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

region
A *Region* object specifying the region to which the load is applied.

Optional arguments

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

comp1
A Float or a Complex specifying the component of the load in the global or the beam local 1-direction.



Note:

Although *comp1*, *comp2*, and *comp3* are optional arguments, at least one of them must be nonzero unless *distributionType*=USER_DEFINED.

comp2
A Float or a Complex specifying the component of the load in the global or the beam local 2-direction.

comp3
A Float or a Complex specifying the component of the load in the global 3-direction.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

system A SymbolicConstant specifying whether the load is applied in a global or the beam local frame of reference. Possible values are GLOBAL and LOCAL. The default value is GLOBAL.

Return value

A LineLoad object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing LineLoad object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LineLoad* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing LineLoad object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Float, a Complex, or a SymbolicConstant specifying the load component in the global or the beam local 1-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

comp2
A Float, a Complex, or a SymbolicConstant specifying the load component in the global or the beam local 2-direction. For details see *comp1*.

comp3
A Float, a Complex, or a SymbolicConstant specifying the load component in the global 3-direction. For details see *comp1*.

amplitude
A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value
`None.`

Exceptions
None.

Members
The LineLoad object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

system
A SymbolicConstant specifying whether the load is applied in a global or the beam local frame of reference. Possible values are GLOBAL and LOCAL. The default value is GLOBAL.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

LineLoadState object

The LineLoadState object stores the propagating data of a line load in a step. One instance of this object is created internally by the *LineLoad* object for each step. The instance is also deleted internally by the *LineLoad* object.

The LineLoadState object has no constructor or methods.

The LineLoadState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The LineLoadState object has the following members:

comp1

A Float or a Complex specifying the load component in the global or the beam local 1-direction.

comp2

A Float or a Complex specifying the load component in the global or the beam local 2-direction.

comp3

A Float or a Complex specifying the load component in the global 3-direction.

comp1State

A SymbolicConstant specifying the propagation state of the load component in the global or the beam local 1-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp2State

A SymbolicConstant specifying the propagation state of the load component in the global or the beam local 2-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp3State

A SymbolicConstant specifying the propagation state of the load component in the global 3-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DLOAD* (load type label: PX, PY, PZ for a global reference frame, and P1, P2 for a local reference frame)

Moment object

The Moment object stores the data for a moment.

The Moment object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

Moment(...)

This method creates a Moment object.

Path

```
mdb.models[name].Moment
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

Optional arguments

cm1 A Float or a Complex specifying the load component in the 4-direction.



Note:

Although *comp1*, *comp2*, and *comp3* are optional arguments, at least one of them must be nonzero.

cm2 A Float or a Complex specifying the load component in the 5- direction.

cm3 A Float or a Complex specifying the load component in the 6-direction.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

follower A Boolean specifying whether the direction of the force rotates with the rotation of the node. You should provide the *follower* argument only if it is valid for the specified step. The default value is OFF.

localCsys `None` or a *DatumCsys* object specifying the ID of the Datum coordinate system used as the local coordinate system of the load. If *localCsys*=`None`, the load is defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

Return value

A Moment object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing Moment object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Moment* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing Moment object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Float, a Complex, or a SymbolicConstant specifying the load component in the 4-direction. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should

be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

comp2

A Float, a Complex, or a SymbolicConstant specifying the load component in the 5-direction. For details see *comp1*.

comp3

A Float, a Complex, or a SymbolicConstant specifying the load component in the 6-direction. For details see *comp1*.

amplitude

A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None`.

Exceptions

None.

Members

The Moment object can have the following members:

name

A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

follower

A Boolean specifying whether the direction of the force rotates with the rotation of the node. You should provide the *follower* argument only if it is valid for the specified step. The default value is OFF.

localCsys

`None` or a *DatumCsys* object specifying the ID of the Datum coordinate system used as the local coordinate system of the load. If *localCsys*=`None`, the load is defined in the global coordinate system. When this member is queried, it returns an Int. The default value is `None`.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

MomentState object

The MomentState object stores the propagating data for a moment in a step. One instance of this object is created internally by the *Moment* object for each step. The instance is also deleted internally by the *Moment* object.

The MomentState object has no constructor or methods.

The MomentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The MomentState object has the following members:

cm1

A Float or a Complex specifying the load component in the 4-direction.

cm2

A Float or a Complex specifying the load component in the 5-direction.

cm3

A Float or a Complex specifying the load component in the 6-direction.

cm1State

A SymbolicConstant specifying the propagation state of the load component in the 4-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

cm2State

A SymbolicConstant specifying the propagation state of the load component in the 5-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

cm3State

A SymbolicConstant specifying the propagation state of the load component in the 6-direction. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (degree of freedom: 4, 5, or 6)

PEGLoad object

The PEGLoad object stores the data for a PEG load.

The PEGLoad object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

PEGLoad(...)

This method creates a PEGLoad object.

Path

```
mdb.models[name].PEGLoad
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

region
A *Region* object specifying the region to which the load is applied.

Optional arguments

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

comp1
A Float or a Complex specifying the load component at dof 1 of reference node 1.



Note:

Although *comp1*, *comp2*, and *comp3* are optional arguments, at least one of them must be nonzero.

comp2
A Float or a Complex specifying the load component at dof 1 of reference node 2.

comp3
A Float or a Complex specifying the load component at dof 2 of reference node 2.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A PEGLoad object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing PEGLoad object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PEGLoad* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing PEGLoad object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Float, a Complex, or a SymbolicConstant specifying the load component at dof 1 of reference node 1. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the load component is propagated from the previous static analysis step. Use FREED to remove a previously defined load component.

comp2 A Float, a Complex, or a SymbolicConstant specifying the load component at dof 1 of reference node 2. For details see *comp1*.

comp3 A Float, a Complex, or a SymbolicConstant specifying the load component at dof 2 of reference node 2. For details see *comp1*.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load

is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The PEGLoad object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

PEGLoadState object

The PEGLoadState object stores the propagating data for a concentrated force in a step. One instance of this object is created internally by the *PEGLoad* object for each step. The instance is also deleted internally by the *PEGLoad* object.

The PEGLoadState object has no constructor or methods.

The PEGLoadState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The PEGLoadState object has the following members:

comp1

A Float or a Complex specifying the load component at dof 1 of reference node 1.

comp2

A Float or a Complex specifying the load component at dof 1 of reference node 2.

comp3

A Float or a Complex specifying the load component at dof 2 of reference node 2.

comp1State

A SymbolicConstant specifying the propagation state of the load component at dof 1 of reference node 1. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp2State

A SymbolicConstant specifying the propagation state of the load component at dof 1 of reference node 2. Possible values are UNSET, SET, UNCHANGED, and FREED.

comp3State

A SymbolicConstant specifying the propagation state of the load component at dof 2 of reference node 2. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **CLOAD* (degree of freedom: 1 or 2)

PipePressure object

The PipePressure object stores the data for a pressure applied to pipe or elbow elements.

The PipePressure object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

PipePressure(...)

This method creates a *Pressure* object.

Path

```
mdb.models[name].PipePressure
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the pressure is created.

region
A *Region* object specifying the region to which the load is applied.

magnitude
A Float specifying the pressure magnitude.



Note:

magnitude is optional if *distributionType*=USER_DEFINED.

diameter
A Float specifying the effective inner or outer diameter.

hZero
A Float specifying the height of the zero pressure level when *distributionType*=HYDROSTATIC.

hReference
A Float specifying the height of the reference pressure level when *distributionType*=HYDROSTATIC.

Optional arguments

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are UNIFORM, HYDROSTATIC, USER_DEFINED, and FIELD. The default value is UNIFORM.

side A SymbolicConstant specifying whether the pressure is applied internally or externally. Possible values are INTERNAL and EXTERNAL. The default value is INTERNAL.

Return value

A PipePressure object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing PipePressure object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PipePressure* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing PipePressure object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the pressure magnitude.

hZero A Float specifying the height of the zero pressure level when *distributionType*=HYDROSTATIC.

hReference A Float specifying the height of the reference pressure level when *distributionType*=HYDROSTATIC.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The PipePressure object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are UNIFORM, HYDROSTATIC, USER_DEFINED, and FIELD. The default value is UNIFORM.

side A SymbolicConstant specifying whether the pressure is applied internally or externally. Possible values are INTERNAL and EXTERNAL. The default value is INTERNAL.

diameter A Float specifying the effective inner or outer diameter.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

PipePressureState object

The PipePressureState object stores the propagating data for a pipe pressure in a step. One instance of this object is created internally by the *PipePressure* object for each step. The instance is also deleted internally by the *PipePressure* object.

The PipePressureState object has no constructor or methods.

The PipePressureState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The PipePressureState object has the following members:

magnitude

A Float or a Complex specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

hZero

A Float specifying the height of the zero pressure level when the pipe pressure *distributionType*=HYDROSTATIC.

hZeroState

A SymbolicConstant specifying the propagation state of *hZero*. Possible values are UNSET, SET, UNCHANGED, and FREED.

hReference

A Float specifying the height of the reference pressure level when the pipe pressure *distributionType*=HYDROSTATIC.

hReferenceState

A SymbolicConstant specifying the propagation state of *hReference*. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE

- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSLOAD*
- **DLOAD*

Pressure object

The Pressure object defines a pressure load.

The Pressure object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

Pressure(...)

This method creates a Pressure object.

Path

```
mdb.models[name].Pressure
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the pressure is created.

region
A *Region* object specifying the region to which the load is applied.

magnitude
A Float or a Complex specifying the pressure magnitude.



Note:

magnitude is optional if *distributionType*=USER_DEFINED.

hZero
A Float specifying the height of the zero pressure level when *distributionType*=HYDROSTATIC.

hReference
A Float specifying the height of the reference pressure level when *distributionType*=HYDROSTATIC.

Optional arguments

field
A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

refPoint
A Region specifying the reference point from which the relative velocity is determined when *distributionType*=STAGNATION or VISCOUS.

distributionType A SymbolicConstant specifying how the pressure is distributed spatially. Possible values are UNIFORM, USER_DEFINED, FIELD, HYDROSTATIC, STAGNATION, VISCOUS, TOTAL_FORCE, and DISCRETE_FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A Pressure object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing Pressure object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Pressure* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing Pressure object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float or a Complex specifying the pressure magnitude.

hZero A Float specifying the height of the zero pressure level when *distributionType*=HYDROSTATIC.

hReference A Float specifying the height of the reference pressure level when *distributionType*=HYDROSTATIC.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The Pressure object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the pressure is distributed spatially. Possible values are UNIFORM, USER_DEFINED, FIELD, HYDROSTATIC, STAGNATION, VISCOUS, TOTAL_FORCE, and DISCRETE_FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

PressureState object

The PressureState object stores the propagating data for a pressure in a step. One instance of this object is created internally by the *Pressure* object for each step. The instance is also deleted internally by the *Pressure* object.

The PressureState object has no constructor or methods.

The PressureState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The PressureState object has the following members:

magnitude

A Float or a Complex specifying the pressure magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the pressure magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

hZero

A Float specifying the height of the zero pressure level when the pressure *distributionType*=HYDROSTATIC.

hZeroState

A SymbolicConstant specifying the propagation state of *hZero*. Possible values are UNSET, SET, UNCHANGED, and FREED.

hReference

A Float specifying the height of the reference pressure level when the pressure *distributionType*=HYDROSTATIC.

hReferenceState

A SymbolicConstant specifying the propagation state of *hReference*. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSLOAD*
- **DLOAD*

RotationalBodyForce object

The RotationalBodyForce object stores the data for a rotational body force.

The RotationalBodyForce object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

RotationalBodyForce(...)

This method creates a RotationalBodyForce object.

Path

```
mdb.models[name].RotationalBodyForce
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created. This must be the first analysis step name.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float specifying the load magnitude.

point1 A sequence of Floats specifying the first point on the axis of rotation for the load.

point2 A sequence of Floats specifying the second point on the axis of rotation for the load.

Optional arguments

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

centrifugal A Boolean specifying whether or not the effect of the load is centrifugal. The default value is OFF.



Note: At least one of *centrifugal* or *rotaryAcceleration* or *rotorDynamicloads*must be specified and only one must have the value ON.

rotaryAcceleration A Boolean specifying whether or not the effect of the load is rotary acceleration. The default value is OFF.



Note: At least one of *centrifugal* or *rotaryAcceleration* or *rotorDynamicloads* must be specified and only one must have the value ON.

rotorDynamicloads A Boolean specifying whether or not the effect of the load is rotordynamic. The default value is OFF.



Note: At least one of *centrifugal* or *rotaryAcceleration* or *rotorDynamicloads* must be specified and only one must have the value ON.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A RotationalBodyForce object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing RotationalBodyForce object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RotationalBodyForce* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing RotationalBodyForce object in the specified step.

Required arguments

stepName
A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude
A Float specifying the load magnitude.

amplitude
A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The RotationalBodyForce object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

centrifugal
A Boolean specifying whether or not the effect of the load is centrifugal. The default value is OFF.



Note:

At least one of *centrifugal* or *rotaryAcceleration* or *rotorDynamicloads* must be specified and only one must have the value ON.

rotaryAcceleration
A Boolean specifying whether or not the effect of the load is rotary acceleration. The default value is OFF.



Note:

At least one of *centrifugal* or *rotaryAcceleration* or *rotarDynamicloads* must be specified and only one must have the value ON.

rotorDynamicloads
A Boolean specifying whether or not the effect of the load is rotordynamic. The default value is OFF.



Note:

At least one of *centrifugal* or *rotaryAcceleration* or *rotorDynamicloads* must be specified and only one must have the value ON.

point1

A tuple of Floats specifying the first point on the axis of rotation for the load.

point2

A tuple of Floats specifying the second point on the axis of rotation for the load.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

RotationalBodyForceState object

The RotationalBodyForceState object stores the propagating data of a rotational body force in a step. One instance of this object is created internally by the *RotationalBodyForce* object for each step. The instance is also deleted internally by the *RotationalBodyForce* object.

The RotationalBodyForceState object has no constructor or methods.

The RotationalBodyForceState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The RotationalBodyForceState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DLOAD* (load type labels: CENTRIF, ROTA)

ShellEdgeLoad object

The ShellEdgeLoad object defines shell edge loads on a region.

The ShellEdgeLoad object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

ShellEdgeLoad(...)

This method creates a ShellEdgeLoad object.

Path

```
mdb.models[name].ShellEdgeLoad
```

Required arguments

name
A String specifying the load repository key.

createStepName
A String specifying the name of the step in which the load is created.

region
A *Region* object specifying the region to which the load is applied.

magnitude
A Float or Complex specifying the load magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED

Optional arguments

distributionType
A SymbolicConstant specifying how the shell edge load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field
A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

amplitude
A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

angle
A Float specifying an additional rotation of *directionVector* about an axis. The default value is 0.This parameter is available only if *traction* is GENERAL.

axis
A SymbolicConstant specifying the axis about which to apply an additional rotation of *directionVector*. Possible values are AXIS_1, AXIS_2, AXIS_3. The default value is AXIS_1.

This parameter is available only if *traction* is GENERAL.

localCsys A *DatumCsys* object specifying the local coordinate system of the load's degrees of freedom. The default value is `None`, indicating that the degrees of freedom are defined in the global coordinate system or by the *userCsys* parameter if defined. This parameter is available only if *traction* is GENERAL. When this member is queried, it returns an Int.

userCsys A String specifying a CSYS defined by a user-subroutine. The default value is `None`, indicating that the degrees of freedom are defined in the global coordinate system or by the *localCsys* parameter if defined. This parameter is available only if *traction* is GENERAL.

directionVector A tuple of two points specifying the direction of the load. Each point is specified as a point region or a tuple of coordinates. If *traction* is SHEAR, then *directionVector* will be projected onto the region surface. This parameter is available only if *traction* is GENERAL.

follower A Boolean specifying whether the direction of the force changes with rotation. The default value is ON. This parameter may be modified only if *traction* is GENERAL. You should provide the *follower* argument only if it is valid for the specified step.

resultant A Boolean specifying whether to maintain a constant resultant force by defining traction per unit undeformed area. If *resultant* is OFF, traction is defined per unit deformed area. The default value is OFF. You should provide the *resultant* argument only if it is valid for the specified step.

traction A SymbolicConstant specifying how to apply surface traction. Possible values are NORMAL, TRANSVERSE, SHEAR, MOMENT and GENERAL. The default value is NORMAL.

Return value

A ShellEdgeLoad object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing ShellEdgeLoad object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShellEdgeLoad* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing ShellEdgeLoad object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the surface pore fluid flow is modified.

Optional arguments

magnitude A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the load magnitude. UNCHANGED should be used if the magnitude is propagated from the previous analysis step.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The ShellEdgeLoad object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the shell edge load is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

ShellEdgeLoadState object

The ShellEdgeLoadState object stores the propagating data for a *ShellEdgeLoad* object in a step. One instance of this object is created internally by the *ShellEdgeLoad* object for each step. The instance is also deleted internally by the *ShellEdgeLoad* object.

The ShellEdgeLoadState object has no constructor or methods.

The ShellEdgeLoadState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The ShellEdgeLoadState object has the following members:

magnitude

A Float or a Complex specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSLOAD*
- **DLOAD*

SubmodelSB object

The SubmodelSB object stores the data for a submodel surface based load.

The SubmodelSB object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SubmodelSB(...)

This method creates a SubmodelSB object.

Path

```
mdb.models[name].SubmodelSB
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

globalStep A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step.

Optional arguments

globalDrivingRegion A String specifying the element set in the global model that will be searched for elements whose responses will be used to drive the submodel. An empty string indicates that the entire global model will be searched. The default value is an empty string.

absoluteExteriorTolerance `None` or a Float specifying the absolute value by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is `None`.

exteriorTolerance `None` or a Float specifying the fraction of the average element size in the global model by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is 0.05.

globalIncrement An Int specifying the increment number in the global model step from which the solution will be used to specify the values of the driven variables. If *globalIncrement*=0, the solution from the last increment will be used. The *globalIncrement* argument is applicable only for linear perturbation steps. The default value is 0.

Return value

A SubmodelSB object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SubmodelSB object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SubmodelSB* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SubmodelSB object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

fixed A Boolean specifying whether the load should remain fixed at the current values at the start of the step. The default value is ON.

globalStep A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step. The *globalStep* argument is applicable only if *fixed*=OFF.

globalIncrement An Int specifying the increment number in the global model step at which the solution will be used to specify the values of the driven variables. If *globalIncrement*=0, the solution from the last increment will be used. The *globalIncrement* argument is applicable only for linear perturbation steps and if *fixed*=OFF. The default value is 0.

Return value
`None`.

Exceptions
None.

Members
The SubmodelSB object can have the following members:

name
A String specifying the load repository key.

absoluteExteriorTolerance
`None` or a Float specifying the absolute value by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is `None`.

exteriorTolerance
`None` or a Float specifying the fraction of the average element size in the global model by which a driven node of the submodel can lie outside the region of the elements of the global model. The default value is 0.05.

globalDrivingRegion
A String specifying the element set in the global model that will be searched for elements whose responses will be used to drive the submodel. An empty string indicates that the entire global model will be searched. The default value is an empty string.

region
A *Region* object specifying the region to which the load is applied.

SubmodelSBState object

The SubmodelSBState object stores the propagating data for a Submodel load in a step. One instance of this object is created internally by the *SubmodelSB* object for each step. The instance is also deleted internally by the *SubmodelSB* object.

The SubmodelSBState object has no constructor or methods.

The SubmodelSBState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SubmodelSBState object has the following members:

globalStepState

A SymbolicConstant specifying the propagation state of the *globalStep* member. Possible values are SET and UNCHANGED.

globalIncrement

An Int specifying the increment number in the global model step at which the solution will be used to specify the values of the driven variables. This argument is applicable only for linear perturbation steps.

globalIncrementState

A SymbolicConstant specifying the propagation state of the *globalIncrement* member. Possible values are SET and UNCHANGED.

globalStep

A String specifying the step in the global model from which Abaqus reads the values of the variables that will drive the submodel analysis. The String indicates the position of the step in the sequence of analysis steps. For example, *globalStep*='1' indicates the first step.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **SUBMODEL*
- **DSLOAD*, SUBMODEL

SubstructureLoad object

The SubstructureLoad object defines a substructure load.

The SubstructureLoad object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SubstructureLoad(...)

This method creates a SubstructureLoad object.

Path

```
mdb.models[name].SubstructureLoad
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the substructure load is created.

region A *Region* object specifying the region to which the load is applied.

loadCaseNames A list of names of the load cases that should be activated by this substructure load.

magnitude A Float specifying the multiplier for the load case magnitude.

Optional arguments

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SubstructureLoad object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SubstructureLoad object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SubstructureLoad* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing SubstructureLoad object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

loadCaseNames A list of names of the load cases that should be activated by this substructure load.

magnitude A Float specifying the multiplier for the load case magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SubstructureLoad object can have the following members:

name A String specifying the load repository key.

region A *Region* object specifying the region to which the load is applied.

SubstructureLoadState object

The SubstructureLoadState object stores the propagating data for a substructure load in a step. One instance of this object is created internally by the *SubstructureLoad* object for each step. The instance is also deleted internally by the *SubstructureLoad* object.

The SubstructureLoadState object has no constructor or methods.

The SubstructureLoadState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SubstructureLoadState object has the following members:

loadCaseNames

A tuple of strings specifying the names of the load cases to be activated.

magnitude

A Float or a Complex specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **SLOAD*

SurfaceCharge object

The SurfaceCharge object stores the data for a surface charge.

The SurfaceCharge object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceCharge(...)

This method creates a SurfaceCharge object.

Path

```
mdb.models[name].SurfaceCharge
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfaceCharge object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceCharge object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceCharge* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceCharge object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceCharge object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

SurfaceChargeState object

The SurfaceChargeState object stores the propagating data of a surface charge in a step. One instance of this object is created internally by the *SurfaceCharge* object for each step. The instance is also deleted internally by the *SurfaceCharge* object.

The SurfaceChargeState object has no constructor or methods.

The SurfaceChargeState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfaceChargeState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSECHARGE* (load type label: ES)

SurfaceConcentrationFlux object

The SurfaceConcentrationFlux object defines surface concentration flux from a region or into a region.

The SurfaceConcentrationFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceConcentrationFlux(...)

This method creates a SurfaceConcentrationFlux object.

Path

```
mdb.models[name].SurfaceConcentrationFlux
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the surface concentration flux magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
distributionType	A SymbolicConstant specifying how the surface concentration flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfaceConcentrationFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceConcentrationFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceConcentrationFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceConcentrationFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the surface concentration flux is modified.

Optional arguments

magnitude A Float specifying the surface concentration flux magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceConcentrationFlux object can have the following members:

name

A String specifying the load repository key.

distributionType

A SymbolicConstant specifying how the surface concentration flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region

A *Region* object specifying the region to which the load is applied.

SurfaceConcentrationFluxState object

The SurfaceConcentrationFluxState object stores the propagating data for a *SurfaceConcentrationFlux* object in a step. One instance of this object is created internally by the *SurfaceConcentrationFlux* object for each step. The instance is also deleted internally by the *SurfaceConcentrationFlux* object.

The SurfaceConcentrationFluxState object has no constructor or methods.

The SurfaceConcentrationFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfaceConcentrationFluxState object has the following members:

magnitude

A Float specifying the surface concentration flux magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the surface concentration flux magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSFLUX*

SurfaceCurrent object

The SurfaceCurrent object stores the data for a surface current.

The SurfaceCurrent object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceCurrent(...)

This method creates a SurfaceCurrent object.

Path

```
mdb.models[name].SurfaceCurrent
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
magnitude	A Float specifying the load magnitude.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.
field	A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfaceCurrent object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceCurrent object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceCurrent* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceCurrent object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float specifying the load magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceCurrent object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

SurfaceCurrentDensity object

The SurfaceCurrentDensity object stores the data for a surface current.

The SurfaceCurrentDensity object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceCurrentDensity(...)

This method creates a SurfaceCurrentDensity object.

Path

```
mdb.models[name].SurfaceCurrentDensity
```

Required arguments

name	A String specifying the load repository key.
createStepName	A String specifying the name of the step in which the load is created. This must be the first analysis step name.
region	A *Region* object specifying the region to which the load is applied.
comp1	A Complex specifying the first component of the load.
comp2	A Complex specifying the second component of the load.
comp3	A Complex specifying the third component of the load.

Optional arguments

distributionType	A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.
amplitude	A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfaceCurrentDensity object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceCurrentDensity object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceCurrentDensity* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceCurrentDensity object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

comp1 A Complex specifying the first component of the load.

comp2 A Complex specifying the second component of the load.

comp3 A Complex specifying the third component of the load.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous static analysis step. FREED should be used if the load is changed to have no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceCurrentDensity object can have the following members:

name
A String specifying the load repository key.

distributionType
A SymbolicConstant specifying how the load is distributed spatially. Possible values are UNIFORM and USER_DEFINED. The default value is UNIFORM.

region
A *Region* object specifying the region to which the load is applied.

SurfaceCurrentState object

The SurfaceCurrentState object stores the propagating data of a surface current in a step. One instance of this object is created internally by the *SurfaceCurrent* object for each step. The instance is also deleted internally by the *SurfaceCurrent* object.

The SurfaceCurrentState object has no constructor or methods.

The SurfaceCurrentState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfaceCurrentState object has the following members:

magnitude

A Float specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and FREED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSECURRENT* (load type label: CS)

SurfaceHeatFlux object

The SurfaceHeatFlux object defines surface heat flux from a region or into a region.

The SurfaceHeatFlux object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceHeatFlux(...)

This method creates a SurfaceHeatFlux object.

Path

```
mdb.models[name].SurfaceHeatFlux
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float specifying the surface heat flux magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

distributionType A SymbolicConstant specifying how the surface heat flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfaceHeatFlux object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfaceHeatFlux object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceHeatFlux* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceHeatFlux object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the surface heat flux is modified.

Optional arguments

magnitude A Float specifying the surface heat flux magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceHeatFlux object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying how the surface heat flux is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

SurfaceHeatFluxState object

The SurfaceHeatFluxState object stores the propagating data for a surface *SurfaceHeatFlux* object in a step. One instance of this object is created internally by the *SurfaceHeatFlux* object for each step. The instance is also deleted internally by the *SurfaceHeatFlux* object.

The SurfaceHeatFluxState object has no constructor or methods.

The SurfaceHeatFluxState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfaceHeatFluxState object has the following members:

magnitude

A Float specifying the surface heat flux magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the surface heat flux magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSFLUX*

SurfacePoreFluid object

The SurfacePoreFluid object defines surface pore fluid flow from a region or into a region.

The SurfacePoreFluid object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfacePoreFluid(...)

This method creates a SurfacePoreFluid object.

Path

```
mdb.models[name].SurfacePoreFluid
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float specifying the surface pore fluid flow magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

A SurfacePoreFluid object.

Exceptions

None.

setValues(...)

This method modifies the data for an existing SurfacePoreFluid object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfacePoreFluid* method, except for the *name* and *createStepName* arguments.

Return value

```
None.
```

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfacePoreFluid object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the surface pore fluid flow is modified.

Optional arguments

magnitude A Float specifying the surface pore fluid flow magnitude.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

```
None.
```

Exceptions

None.

Members

The SurfacePoreFluid object can have the following members:

name A String specifying the load repository key.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the load is applied.

SurfacePoreFluidState object

The SurfacePoreFluidState object stores the propagating data for a *SurfacePoreFluid* object in a step. One instance of this object is created internally by the *SurfacePoreFluid* object for each step. The instance is also deleted internally by the *SurfacePoreFluid* object.

The SurfacePoreFluidState object has no constructor or methods.

The SurfacePoreFluidState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfacePoreFluidState object has the following members:

magnitude

A Float specifying the surface pore fluid flow magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the surface pore fluid flow magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSFLOW*

SurfaceTraction object

The SurfaceTraction object defines surface traction on a region.

The SurfaceTraction object is derived from the *Load* object.

Access

```
import load
mdb.models[name].loads[name]
```

SurfaceTraction(...)

This method creates a SurfaceTraction object.

Path

```
mdb.models[name].SurfaceTraction
```

Required arguments

name A String specifying the load repository key.

createStepName A String specifying the name of the step in which the load is created.

region A *Region* object specifying the region to which the load is applied.

magnitude A Float or Complex specifying the load magnitude. *magnitude* is optional if *distributionType*=USER_DEFINED.

Optional arguments

distributionType A SymbolicConstant specifying how the surface traction is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

amplitude A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the load has no amplitude reference. The default value is UNSET. You should provide the *amplitude* argument only if it is valid for the specified step.

angle A Float specifying an additional rotation of *directionVector* about an axis. The default value is 0.0.

axis A SymbolicConstant specifying the axis about which to apply an additional rotation of *directionVector*. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system of the load's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system

or by the *userCsys* parameter if defined. When this member is queried, it returns an Int. The default value is `None`.

userCsys

A String specifying a CSYS defined by a user-subroutine. If *userCsys*=`None`, the degrees of freedom are defined in the global coordinate system or by the *localCsys* parameter if defined. The default value is "None".

directionVector

A *VertexArray* object of length 2 specifying the direction of the load. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates. If *traction* is SHEAR, then *directionVector* will be projected onto the region surface. This parameter is available only if *traction* is GENERAL or SHEAR.

follower

A Boolean specifying whether the direction of the force changes with rotation. The default value is ON.

This parameter may be modified only if *traction* is GENERAL. You should provide the *follower* argument only if it is valid for the specified step.

resultant

A Boolean specifying whether the to maintain a constant resultant force by defining traction per unit undeformed area. If *resultant* is OFF, traction is defined per unit deformed area. The default value is OFF.

You should provide the *resultant* argument only if it is valid for the specified step.

traction

A SymbolicConstant specifying how to apply surface traction. Possible values are SHEAR and GENERAL. The default value is SHEAR.

Return value

A SurfaceTraction object.

Exceptions

None.

`setValues(...)`

This method modifies the data for an existing SurfaceTraction object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SurfaceTraction* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing SurfaceTraction object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the load is modified.

Optional arguments

magnitude A Float, a Complex, or the SymbolicConstant UNCHANGED specifying the load magnitude. UNCHANGED should be used if the magnitude is propagated from the previous analysis step.

amplitude A String or a SymbolicConstant specifying the name of the amplitude reference. Possible values for the SymbolicConstant are UNCHANGED and FREED. UNCHANGED should be used if the amplitude is propagated from the previous analysis step. FREED should be used if the load has no amplitude reference. You should provide the *amplitude* argument only if it is valid for the specified step.

Return value

`None.`

Exceptions

None.

Members

The SurfaceTraction object can have the following members:

name A String specifying the load repository key.

angle A Float specifying an additional rotation of *directionVector* about an axis. The default value is 0.0.

axis A SymbolicConstant specifying the axis about which to apply an additional rotation of *directionVector*. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

follower A Boolean specifying whether the direction of the force changes with rotation. The default value is ON.

This parameter may be modified only if *traction* is GENERAL. You should provide the *follower* argument only if it is valid for the specified step.

resultant

A Boolean specifying whether the to maintain a constant resultant force by defining traction per unit undeformed area. If *resultant* is OFF, traction is defined per unit deformed area. The default value is OFF.

You should provide the *resultant* argument only if it is valid for the specified step.

traction

A SymbolicConstant specifying how to apply surface traction. Possible values are SHEAR and GENERAL. The default value is SHEAR.

distributionType

A SymbolicConstant specifying how the surface traction is distributed spatially. Possible values are UNIFORM, USER_DEFINED, and FIELD. The default value is UNIFORM.

field

A String specifying the name of the *AnalyticalField* object associated with this load. The *field* argument applies only when *distributionType*=FIELD. The default value is an empty string.

userCsys

A String specifying a CSYS defined by a user-subroutine. If *userCsys*=`None`, the degrees of freedom are defined in the global coordinate system or by the *localCsys* parameter if defined. The default value is "None".

localCsys

`None` or a *DatumCsys* object specifying the local coordinate system of the load's degrees of freedom. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system or by the *userCsys* parameter if defined. When this member is queried, it returns an Int. The default value is `None`.

directionVector

A *VertexArray* object of length 2 specifying the direction of the load. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates. If *traction* is SHEAR, then *directionVector* will be projected onto the region surface. This parameter is available only if *traction* is GENERAL or SHEAR.

region

A *Region* object specifying the region to which the load is applied.

SurfaceTractionState object

The SurfaceTractionState object stores the propagating data for a *SurfaceTraction* object in a step. One instance of this object is created internally by the *SurfaceTraction* object for each step. The instance is also deleted internally by the *SurfaceTraction* object.

The SurfaceTractionState object has no constructor or methods.

The SurfaceTractionState object is derived from the *LoadState* object.

Access

```
import load
mdb.models[name].steps[name].loadStates[name]
```

Members

The SurfaceTractionState object has the following members:

magnitude

A Float or a Complex specifying the load magnitude.

magnitudeState

A SymbolicConstant specifying the propagation state of the load magnitude. Possible values are UNSET, SET, UNCHANGED, and MODIFIED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitude* member. Possible values are UNSET, SET, UNCHANGED, and FREED.

status

A SymbolicConstant specifying the propagation state of the *LoadState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- NO_LONGER_ACTIVE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE
- BUILT_INTO_BASE_STATE

amplitude

A String specifying the name of the amplitude reference. The String is empty if the load has no amplitude reference.

Corresponding analysis keywords

- **DSLOAD*
- **DLOAD*

Material commands

The Material commands are used to define the materials in a model.

In this section:

- *Material object*
- *AcousticMedium object*
- *AnnealTemperature object*
- *BiaxialTestData object*
- *BrittleCracking object*
- *BrittleFailure object*
- *BrittleShear object*
- *CapCreepCohesion object*
- *CapCreepConsolidation object*
- *CapHardening object*
- *CapPlasticity object*
- *CastIronCompressionHardening object*
- *CastIronPlasticity object*
- *CastIronTensionHardening object*
- *ClayHardening object*
- *ClayPlasticity object*
- *CombinedTestData object*
- *Concrete object*
- *ConcreteCompressionDamage object*
- *ConcreteCompressionHardening object*
- *ConcreteDamagedPlasticity object*
- *ConcreteTensionDamage object*
- *ConcreteTensionStiffening object*
- *Conductivity object*
- *ContactArea object*
- *Creep object*
- *CrushableFoam object*
- *CrushableFoamHardening object*
- *CrushStress object*
- *CrushStressVelocityFactor object*
- *CycledPlastic object*
- *CyclicHardening object*
- *DamageEvolution object*
- *DamageInitiation object*
- *DamageStabilization object*
- *DamageStabilizationCohesive object*
- *Damping object*
- *DeformationPlasticity object*

- *Density object*
- *Depvar object*
- *DetonationPoint object*
- *Dielectric object*
- *Diffusivity object*
- *DruckerPrager object*
- *DruckerPragerCreep object*
- *DruckerPragerHardening object*
- *Elastic object*
- *ElectricalConductivity object*
- *Eos object*
- *EosCompaction object*
- *Expansion object*
- *FailStrain object*
- *FailStress object*
- *FailureRatios object*
- *FluidLeakoff object*
- *GapConductance object*
- *GapConvection object*
- *GapFlow object*
- *GapRadiation object*
- *GasketMembraneElastic object*
- *GasketThicknessBehavior object*
- *GasketTransverseShearElastic object*
- *Gel object*
- *HeatGeneration object*
- *Hyperelastic object*
- *Hyperfoam object*
- *Hypoelastic object*
- *Hysteresis object*
- *InelasticHeatFraction object*
- *JouleHeatFraction object*
- *LatentHeat object*
- *LowDensityFoam object*
- *MagneticPermeability object*
- *MeanFieldHomogenization object*
- *MeanFieldInclusion object*
- *MeanFieldMatrix object*
- *MeanFieldVoid object*
- *MohrCoulombHardening object*
- *MohrCoulombPlasticity object*
- *MoistureSwelling object*
- *MullinsEffect object*
- *Ornl object*
- *Permeability object*
- *Piezoelectric object*

- *PlanarTestData object*
- *Plastic object*
- *PlasticityCorrection object*
- *PoreFluidExpansion object*
- *PorousBulkModuli object*
- *PorousElastic object*
- *PorousFailureCriteria object*
- *PorousMetalPlasticity object*
- *Potential object*
- *PressureEffect object*
- *RateDependent object*
- *Ratios object*
- *Regularization object*
- *SaturationDependence object*
- *ShearRetention object*
- *ShearTestData object*
- *SimpleShearTestData object*
- *Solubility object*
- *SoretEffect object*
- *Sorption object*
- *SpecificHeat object*
- *SuperElasticHardening object*
- *SuperElasticHardeningModifications object*
- *SuperElasticity object*
- *Swelling object*
- *TensileFailure object*
- *TensionCutOff object*
- *TensionStiffening object*
- *TriaxialTestData object*
- *Trs object*
- *UniaxialTestData object*
- *UserDefinedField object*
- *UserMaterial object*
- *UserOutputVariables object*
- *VelocityDependence object*
- *Viscoelastic object*
- *Viscosity object*
- *Viscous object*
- *VoidNucleation object*
- *VolumetricTestData object*
- *evaluateMaterial*

Material object

A Material object is the object used to specify a material. The Material object stores the various settings that determine how a material behaves.

A material is created by combining one or more individual material options and sub options. A particular material option is associated with the Material object through a member. For example: the *acousticMedium* member may contain an AcousticMedium object. The alternative of having a MaterialOption abstract base class and a container of MaterialOptions was rejected because it would make it more difficult to enforce the fact that one Material object cannot contain two AcousticMedium objects, for example.

Access

```
import material
mdb.models[name].materials[name]
import odbMaterial
session.odbs[name].materials[name]
```

Material(...)

This method creates a Material object.

Path

```
mdb.models[name].Material
session.odbs[name].Material
```

Required arguments

name A String specifying the name of the new material.

Optional arguments

description A String specifying user description of the material. The default value is an empty string.

materialIdentifier A String specifying material identifier for customer use. The default value is an empty string.

Return value

A Material object.

Exceptions

InvalidNameError.

materialsFromOdb(...)

This methods creates Material objects by reading an output database. The new materials are placed in the materials repository.

Path

mdb.models[*name*].materialsFromOdb

Required arguments

fileName A String specifying the name of the output database file (including the `.odb` extension) to be read. This String can also be the full path to the output database file if it is located in another directory.

Optional arguments

None.

Return value

A list of Material objects.

Exceptions

None.

Members

The Material object has members with the same names and descriptions as the arguments to the *Material* method.

In addition, the Material object can have the following members:

acousticMedium An *AcousticMedium* object.

brittleCracking A *BrittleCracking* object.

capPlasticity A *CapPlasticity* object.

castIronPlasticity A *CastIronPlasticity* object.

clayPlasticity A *ClayPlasticity* object.

concrete A *Concrete* object.

concreteDamagedPlasticity A *ConcreteDamagedPlasticity* object.

conductivity A *Conductivity* object.

creep A *Creep* object.

crushableFoam A *CrushableFoam* object.

crushStress A *CrushStress* object.

CrushStressVelocityFactor A *CrushStressVelocityFactor* object.

ductileDamageInitiation A *DamageInitiation* object.

fldDamageInitiation A *DamageInitiation* object.

flsdDamageInitiation A *DamageInitiation* object.

johnsonCookDamageInitiation A *DamageInitiation* object.

maxeDamageInitiation A *DamageInitiation* object.

maxsDamageInitiation A *DamageInitiation* object.

maxpeDamageInitiation A *DamageInitiation* object.

maxpsDamageInitiation A *DamageInitiation* object.

mkDamageInitiation A *DamageInitiation* object.

msfldDamageInitiation A *DamageInitiation* object.

quadeDamageInitiation A *DamageInitiation* object.

quadsDamageInitiation A *DamageInitiation* object.

shearDamageInitiation A *DamageInitiation* object.

hashinDamageInitiation A *DamageInitiation* object.

damping A *Damping* object.

deformationPlasticity A *DeformationPlasticity* object.

density A *Density* object.

depvar A *Depvar* object.

dielectric A *Dielectric* object.

diffusivity A *Diffusivity* object.

druckerPrager A *DruckerPrager* object.

elastic An *Elastic* object.

electricalConductivity An *ElectricalConductivity* object.

eos An *Eos* object.

expansion An *Expansion* object.

fluidLeakoff A *FluidLeakoff* object.

gapFlow A *GapFlow* object.

gasketThicknessBehavior A *GasketThicknessBehavior* object.

gasketTransverseShearElastic A *GasketTransverseShearElastic* object.

gasketMembraneElastic A *GasketMembraneElastic* object.

gel A *Gel* object.

heatGeneration A *HeatGeneration* object.

hyperelastic A *Hyperelastic* object.

hyperfoam A *Hyperfoam* object.

hypoelastic A *Hypoelastic* object.

inelasticHeatFraction An *InelasticHeatFraction* object.

jouleHeatFraction A *JouleHeatFraction* object.

latentHeat A *LatentHeat* object.

lowDensityFoam A *LowDensityFoam* object.

magneticPermeability A *MagneticPermeability* object.

mohrCoulombPlasticity A *MohrCoulombPlasticity* object.

moistureSwelling A *MoistureSwelling* object.

mullinsEffect A *MullinsEffect* object.

permeability A *Permeability* object.

piezoelectric A *Piezoelectric* object.

plastic A *Plastic* object.

poreFluidExpansion A *PoreFluidExpansion* object.

porousBulkModuli A *PorousBulkModuli* object.

porousElastic	A *PorousElastic* object.
porousMetalPlasticity	A *PorousMetalPlasticity* object.
regularization	A *Regularization* object.
solubility	A *Solubility* object.
sorption	A *Sorption* object.
specificHeat	A *SpecificHeat* object.
swelling	A *Swelling* object.
userDefinedField	A *UserDefinedField* object.
userMaterial	A *UserMaterial* object.
userOutputVariables	A *UserOutputVariables* object.
viscoelastic	A *Viscoelastic* object.
viscosity	A *Viscosity* object.
viscous	A *Viscous* object.

Corresponding analysis keywords

- **MATERIAL*

AcousticMedium object

The AcousticMedium object specifies the acoustic properties of a material.

Access

```
import material
mdb.models[name].materials[name].acousticMedium
import odbMaterial
session.odbs[name].materials[name].acousticMedium
```

AcousticMedium(...)

This method creates an AcousticMedium object.

Path

```
mdb.models[name].materials[name].AcousticMedium
session.odbs[name].materials[name].AcousticMedium
```

Required arguments

None.

Optional arguments

acousticVolumetricDrag

A Boolean specifying whether the volumetricTable data is specified. The default value is OFF.

temperatureDependencyB

A Boolean specifying whether the data in *bulkTable* depend on temperature. The default value is OFF.

temperatureDependencyV

A Boolean specifying whether the data in *volumetricTable* depend on temperature. The default value is OFF.

dependenciesB

An Int specifying the number of field variable dependencies for the data in *bulkTable*. The default value is 0.

dependenciesV

An Int specifying the number of field variable dependencies for the data in *volumetricTable*. The default value is 0.

bulkTable

A sequence of sequences of Floats specifying the following:

- Bulk modulus.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

The default value is an empty sequence.

volumetricTable

A sequence of sequences of Floats specifying the following:

- Volumetric drag.
- Frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

The default value is an empty sequence.

Return value

An AcousticMedium object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the AcousticMedium object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticMedium* method.

Return value

`None.`

Exceptions

RangeError.

Members

The AcousticMedium object has members with the same names and descriptions as the arguments to the *AcousticMedium* method.

Corresponding analysis keywords

- **ACOUSTIC MEDIUM*

AnnealTemperature object

The AnnealTemperature object specifies the material annealing temperature.

Access

```
import material
mdb.models[name].materials[name].plastic.annealTemperature
import odbMaterial
session.odbs[name].materials[name].plastic.annealTemperature
```

AnnealTemperature(...)

This method creates an AnnealTemperature object.

Path

```
mdb.models[name].materials[name].plastic.AnnealTemperature
session.odbs[name].materials[name].plastic.AnnealTemperature
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The annealing temperature, θ.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

An AnnealTemperature object.

Exceptions

RangeError.

setValues(...)

This method modifies the AnnealTemperature object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AnnealTemperature* method.

Return value

`None.`

Exceptions

RangeError.

Members

The AnnealTemperature object has members with the same names and descriptions as the arguments to the *AnnealTemperature* method.

Corresponding analysis keywords

- **ANNEAL TEMPERATURE*

BiaxialTestData object

The BiaxialTestData object provides equibiaxial test data (compression and/or tension).

Access

```
import material
mdb.models[name].materials[name].hyperelastic.biaxialTestData
mdb.models[name].materials[name].hyperfoam.biaxialTestData
mdb.models[name].materials[name].mullinsEffect.biaxialTests[i]
import odbMaterial
session.odbs[name].materials[name].hyperelastic.biaxialTestData
session.odbs[name].materials[name].hyperfoam.biaxialTestData
session.odbs[name].materials[name].mullinsEffect.biaxialTests[i]
```

BiaxialTestData(...)

This method creates a BiaxialTestData object.

Path

```
mdb.models[name].materials[name].hyperelastic.BiaxialTestData
mdb.models[name].materials[name].hyperfoam.BiaxialTestData
mdb.models[name].materials[name].mullinsEffect.BiaxialTestData
session.odbs[name].materials[name].hyperelastic.BiaxialTestData
session.odbs[name].materials[name].hyperfoam.BiaxialTestData
session.odbs[name].materials[name].mullinsEffect.BiaxialTestData
```

Required arguments

table

A sequence of sequences of Floats specifying the following:

- Nominal stress, T_B.
- Nominal strain, ϵ_B.

Optional arguments

smoothing

`None` or an Int specifying the value for smoothing. If *smoothing*=`None`, no smoothing is employed. The default value is `None`.

lateralNominalStrain

A Boolean specifying whether to include lateral nominal strain. The default value is OFF.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Return value

A BiaxialTestData object.

Exceptions

None.

setValues(...)

This method modifies the BiaxialTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BiaxialTestData* method.

Return value

`None.`

Exceptions

None.

Members

The BiaxialTestData object has members with the same names and descriptions as the arguments to the *BiaxialTestData* method.

Corresponding analysis keywords

- **BIAXIAL TEST DATA*

BrittleCracking object

The BrittleCracking object specifies cracking and postcracking properties for the brittle cracking material model.

Access

```
import material
mdb.models[name].materials[name].brittleCracking
import odbMaterial
session.odbs[name].materials[name].brittleCracking
```

BrittleCracking(...)

This method creates a BrittleCracking object.

Path

```
mdb.models[name].materials[name].BrittleCracking
session.odbs[name].materials[name].BrittleCracking
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

type A SymbolicConstant specifying the type of postcracking behavior. Possible values are STRAIN, DISPLACEMENT, and GFI. The default value is STRAIN.

Table data

If *type*=STRAIN the table data specify the following:

- Remaining direct stress after cracking.
- Direct cracking strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT the table data specify the following:

- Remaining direct stress after cracking.
- Direct cracking displacement.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=GFI the table data specify the following:

- Failure stress.
- Mode I fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleCracking object.

Exceptions

None.

`setValues(...)`

This method modifies the BrittleCracking object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BrittleCracking* method.

Return value

`None.`

Exceptions

None.

Members

The BrittleCracking object has members with the same names and descriptions as the arguments to the *BrittleCracking* method.

In addition, the BrittleCracking object can have the following members:

brittleShear A *BrittleShear* object.

brittleFailure A *BrittleFailure* object.

Corresponding analysis keywords

- **BRITTLE CRACKING*

BrittleFailure object

The BrittleFailure object specifies the brittle failure of the material.

Access

```
import material
mdb.models[name].materials[name].brittleCracking.brittleFailure
import odbMaterial
session.odbs[name].materials[name].brittleCracking.brittleFailure
```

BrittleFailure(...)

This method creates a BrittleFailure object.

Path

```
mdb.models[name].materials[name].brittleCracking.BrittleFailure
session.odbs[name].materials[name].brittleCracking.BrittleFailure
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

failureCriteria A SymbolicConstant specifying the failure criteria. Possible values are UNIDIRECTIONAL, BIDIRECTIONAL, and TRIDIRECTIONAL. The default value is UNIDIRECTIONAL.

Table data

If parent *BrittleCracking* member *type*=STRAIN the table data specify the following:

- Direct cracking failure strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If parent *BrittleCracking* member *type*=DISPLACEMENT or *type*=GFI the table data specify the following:

- Direct cracking failure displacement.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleFailure object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the BrittleFailure object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BrittleFailure* method.

Return value

`None.`

Exceptions

RangeError.

Members

The BrittleFailure object has members with the same names and descriptions as the arguments to the *BrittleFailure* method.

Corresponding analysis keywords

- **BRITTLE FAILURE*

BrittleShear object

The BrittleShear object specifies the postcracking shear behavior of a material used in a brittle cracking model.

Access

```
import material
mdb.models[name].materials[name].brittleCracking.brittleShear
import odbMaterial
session.odbs[name].materials[name].brittleCracking.brittleShear
```

BrittleShear(...)

This method creates a BrittleShear object.

Path

```
mdb.models[name].materials[name].brittleCracking.BrittleShear
session.odbs[name].materials[name].brittleCracking.BrittleShear
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

type A SymbolicConstant specifying the type of postcracking shear behavior. Possible values are RETENTION_FACTOR and POWER_LAW. The default value is RETENTION_FACTOR.

Table data

If *type*=RETENTION_FACTOR the table data specify the following:

- Shear retention factor.
- Crack opening strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=POWER_LAW the table data specify the following:

- e.
- p.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleShear object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the BrittleShear object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BrittleShear* method.

Return value

`None.`

Exceptions

RangeError.

Members

The BrittleShear object has members with the same names and descriptions as the arguments to the *BrittleShear* method.

Corresponding analysis keywords

- **BRITTLE SHEAR*

CapCreepCohesion object

The CapCreepCohesion object specifies a cap creep model and material properties.

Access

```
import material
mdb.models[name].materials[name].capPlasticity.capCreepCohesion
import odbMaterial
session.odbs[name].materials[name].capPlasticity.capCreepCohesion
```

CapCreepCohesion(...)

This method creates a CapCreepCohesion object.

Path

```
mdb.models[name].materials[name].capPlasticity.CapCreepCohesion
session.odbs[name].materials[name].capPlasticity.CapCreepCohesion
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

law A SymbolicConstant specifying the cap creep law. Possible values are STRAIN, TIME, SINGHM, USER, POWER_LAW, and TIME_POWER_LAW. The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

time A SymbolicConstant specifying the time increment for the relevant laws. Possible values are CREEP and TOTAL. The default value is TOTAL.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A.
- α.
- m.
- t_1.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=POWER_LAW or *law*=TIME_POWER_LAW, the table data specify the following:

- q_o.
- n.
- m.
- $\dot{\varepsilon}_o$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapCreepCohesion object.

Exceptions

None.

`setValues(...)`

This method modifies the CapCreepCohesion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CapCreepCohesion* method.

Return value

`None.`

Exceptions

None.

Members

The CapCreepCohesion object has members with the same names and descriptions as the arguments to the *CapCreepCohesion* method.

Corresponding analysis keywords

- **CAP CREEP*

CapCreepConsolidation object

The CapCreepConsolidation object specifies a cap creep model and material properties.

Access

```
import material
mdb.models[name].materials[name].capPlasticity.capCreepConsolidation
import odbMaterial
session.odbs[name].materials[name].capPlasticity.capCreepConsolidation
```

CapCreepConsolidation(...)

This method creates a CapCreepConsolidation object.

Path

```
mdb.models[name].materials[name].capPlasticity.CapCreepConsolidation
session.odbs[name].materials[name].capPlasticity.CapCreepConsolidation
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

law A SymbolicConstant specifying the cap creep law. Possible values are STRAIN, TIME, SINGHM, and USER. The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

time A SymbolicConstant specifying the time increment for the relevant laws. Possible values are CREEP and TOTAL. The default value is TOTAL.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A.
- α.
- m.
- t_1.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=POWER_LAW or *law*=TIME_POWER_LAW, the table data specify the following:

- q_o.
- n.
- m.
- $\dot{\varepsilon}_o$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapCreepConsolidation object.

Exceptions

None.

`setValues(...)`

This method modifies the CapCreepConsolidation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CapCreepConsolidation* method.

Return value

`None.`

Exceptions

None.

Members

The CapCreepConsolidation object has members with the same names and descriptions as the arguments to the *CapCreepConsolidation* method.

Corresponding analysis keywords

- **CAP CREEP*

CapHardening object

The CapHardening object specifies Drucker-Prager/Cap plasticity hardening.

Access

```
import material
mdb.models[name].materials[name].capPlasticity.capHardening
import odbMaterial
session.odbs[name].materials[name].capPlasticity.capHardening
```

CapHardening(...)

This method creates a CapHardening object.

Path

```
mdb.models[name].materials[name].capPlasticity.CapHardening
session.odbs[name].materials[name].capPlasticity.CapHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Hydrostatic pressure yield stress.
- Absolute value of the corresponding volumetric inelastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapHardening object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CapHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CapHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CapHardening object has members with the same names and descriptions as the arguments to the *CapHardening* method.

Corresponding analysis keywords

- **CAP HARDENING*

CapPlasticity object

The CapPlasticity object specifies the modified Drucker-Prager/Cap plasticity model.

Access

```
import material
mdb.models[name].materials[name].capPlasticity
import odbMaterial
session.odbs[name].materials[name].capPlasticity
```

CapPlasticity(...)

This method creates a CapPlasticity object.

Path

```
mdb.models[name].materials[name].CapPlasticity
session.odbs[name].materials[name].CapPlasticity
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Material cohesion, d, in the p–t plane (Abaqus/Standard) or in the p–q plane (Abaqus/Explicit).
- Material angle of friction, β, in the p–t plane (Abaqus/Standard) or in the p–q plane (Abaqus/Explicit). Give the value in degrees.
- Cap eccentricity parameter, R. Its value must be greater than zero (typically $0.0 < R < 1.0$).
- Initial cap yield surface position, $\varepsilon_{vol}^{pl}\big|_0$.
- Transition surface radius parameter, α. The default value is 0.0 (i.e., no transition surface).
- (Not used in Abaqus/Explicit) K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. Possible values are $0.778 \le K \le 1.0$. If the default value of 0.0 is accepted, Abaqus/Standard assumes $K = 1.0$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

Return value

A CapPlasticity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CapPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CapPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CapPlasticity object has members with the same names and descriptions as the arguments to the *CapPlasticity* method.

In addition, the CapPlasticity object can have the following members:

capCreepCohesion	A *CapCreepCohesion* object.
capCreepConsolidation	A *CapCreepConsolidation* object.
capHardening	A *CapHardening* object.

Corresponding analysis keywords

- **CAP PLASTICITY*

CastIronCompressionHardening object

The CastIronCompressionHardening object specifies hardening for the Cast- Iron plasticity model.

Access

```
import material
mdb.models[name].materials[name].castIronPlasticity\
.castIronCompressionHardening
import odbMaterial
session.odbs[name].materials[name].castIronPlasticity\
.castIronCompressionHardening
```

CastIronCompressionHardening(...)

This method creates a CastIronCompressionHardening object.

Path

```
mdb.models[name].materials[name].castIronPlasticity\
.CastIronCompressionHardening
session.odbs[name].materials[name].castIronPlasticity\
.CastIronCompressionHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in compression, σ_c.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronCompressionHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the CastIronCompressionHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CastIronCompressionHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CastIronCompressionHardening object has members with the same names and descriptions as the arguments to the *CastIronCompressionHardening* method.

Corresponding analysis keywords

- **CAST IRON COMPRESSION HARDENING*

CastIronPlasticity object

The CastIronPlasticity object specifies the Cast Iron plasticity model.

Access

```
import material
mdb.models[name].materials[name].castIronPlasticity
import odbMaterial
session.odbs[name].materials[name].castIronPlasticity
```

CastIronPlasticity(...)

This method creates a CastIronPlasticity object.

Path

```
mdb.models[name].materials[name].CastIronPlasticity
session.odbs[name].materials[name].CastIronPlasticity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Plastic Poisson's ratio, ν_{pl} (dimensionless).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronPlasticity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CastIronPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CastIronPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CastIronPlasticity object has members with the same names and descriptions as the arguments to the *CastIronPlasticity* method.

In addition, the CastIronPlasticity object can have the following members:

castIronTensionHardening A *CastIronTensionHardening* object.

castIronCompressionHardening A *CastIronCompressionHardening* object.

Corresponding analysis keywords

- **CAST IRON PLASTICITY*

CastIronTensionHardening object

The CastIronTensionHardening object specifies hardening for the Cast- Iron plasticity model.

Access

```
import material
mdb.models[name].materials[name].castIronPlasticity\
.castIronTensionHardening
import odbMaterial
session.odbs[name].materials[name].castIronPlasticity\
.castIronTensionHardening
```

CastIronTensionHardening(...)

This method creates a CastIronTensionHardening object.

Path

```
mdb.models[name].materials[name].castIronPlasticity\
.CastIronTensionHardening
session.odbs[name].materials[name].castIronPlasticity\
.CastIronTensionHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in uniaxial tension, σ_t.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronTensionHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the CastIronTensionHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CastIronTensionHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CastIronTensionHardening object has members with the same names and descriptions as the arguments to the *CastIronTensionHardening* method.

Corresponding analysis keywords

- **CAST IRON TENSION HARDENING*

ClayHardening object

The ClayHardening object specifies hardening for the clay plasticity model.

Access

```
import material
mdb.models[name].materials[name].clayPlasticity.clayHardening
import odbMaterial
session.odbs[name].materials[name].clayPlasticity.clayHardening
```

ClayHardening(...)

This method creates a ClayHardening object.

Path

```
mdb.models[name].materials[name].clayPlasticity.ClayHardening
session.odbs[name].materials[name].clayPlasticity.ClayHardening
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The hydrostatic pressure stress at yield, p_c.
- The absolute value of the corresponding volumetric plastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ClayHardening object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ClayHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ClayHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ClayHardening object has members with the same names and descriptions as the arguments to the *ClayHardening* method.

Corresponding analysis keywords

- **CLAY HARDENING*

ClayPlasticity object

The ClayPlasticity object specifies the extended Cam-clay plasticity model.

Access

```
import material
mdb.models[name].materials[name].clayPlasticity
import odbMaterial
session.odbs[name].materials[name].clayPlasticity
```

ClayPlasticity(...)

This method creates a ClayPlasticity object.

Path

```
mdb.models[name].materials[name].ClayPlasticity
session.odbs[name].materials[name].ClayPlasticity
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

intercept
`None` or a Float specifying e_1, the intercept of the virgin consolidation line with the void ratio axis in a plot of void ratio versus the logarithm of pressure stress. The default value is `None`.

This argument is valid only if *hardening*=EXPONENTIAL.

hardening
A SymbolicConstant specifying the type of hardening/softening definition. Possible values are EXPONENTIAL and TABULAR. The default value is EXPONENTIAL.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *hardening*=EXPONENTIAL, the table data specify the following:

- Logarithmic plastic bulk modulus, λ (dimensionless).
- Stress ratio at critical state, M.
- The initial yield surface size, α_0.
- β, the parameter defining the size of the yield surface on the “wet” side of critical state.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \leq K \leq 1.0$. If the default value of 0.0 is accepted, a value of 1.0 is assumed.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=TABULAR, the table data specify the following:

- Stress ratio at critical state, M.
- The initial volumetric plastic strain, $\varepsilon_{\text{vol}}^{pl}\big|_0$, corresponding to $p_c|_0$ according to the *ClayHardening* definition.
- β, the parameter defining the size of the yield surface on the "wet" side of critical state.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \leq K \leq 1.0$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ClayPlasticity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ClayPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ClayPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ClayPlasticity object has members with the same names and descriptions as the arguments to the *ClayPlasticity* method.

In addition, the ClayPlasticity object can have the following member:

clayHardening A *ClayHardening* object.

Corresponding analysis keywords

- **CLAY PLASTICITY*

CombinedTestData object

The CombinedTestData object specifies simultaneously the normalized shear and bulk compliances or relaxation moduli as functions of time.

Access

```
import material
mdb.models[name].materials[name].viscoelastic.combinedTestData
import odbMaterial
session.odbs[name].materials[name].viscoelastic.combinedTestData
```

CombinedTestData(...)

This method creates a CombinedTestData object.

Path

```
mdb.models[name].materials[name].viscoelastic.CombinedTestData
session.odbs[name].materials[name].viscoelastic.CombinedTestData
```

Required arguments

table

A sequence of sequences of Floats specifying the items described below. The values of the table data depend on the value of the *time* member of the *Viscoelastic* object.

Optional arguments

volinf

`None` or a Float specifying a normalized volume. The value of *volinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is `None`.

If *time*=RELAXATION_TEST_DATA, *volinf* specifies the value of the long-term normalized volumetric modulus, $k_R(\infty)$.

If *time*=CREEP_TEST_DATA, *volinf* specifies the value of the long-term normalized volumetric compliance, $j_K(\infty)$.

shrinf

`None` or a Float specifying a normalized shear. The value of *shrinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is `None`.

If *time*=RELAXATION_TEST_DATA, *shrinf* specifies the value of the long-term normalized shear modulus, $g_R(\infty)$.

If *time*=CREEP_TEST_DATA, *shrinf* specifies the value of the long-term normalized shear compliance, $j_S(\infty)$.

Table data

If *time*=RELAXATION_TEST_DATA, the table data specify the following:

- Normalized shear modulus, $g_R(t)(0 \le g_R(t) \le 1)$.

- Normalized volumetric (bulk) modulus, $k_R\,(t) (0 \le k_R\,(t) \le 1)$.
- Time $t (t > 0)$.

If *time*=CREEP_TEST_DATA, the table data specify the following:

- Normalized shear compliance, $j_S\,(t) (j_S\,(t) \ge 1)$.
- Normalized volumetric (bulk) compliance, $j_K\,(t) (j_K\,(t) \ge 1)$.
- Time $t (t > 0)$.

Return value

A CombinedTestData object.

Exceptions

None.

`setValues(...)`

This method modifies the CombinedTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CombinedTestData* method.

Return value

`None.`

Exceptions

None.

Members

The CombinedTestData object has members with the same names and descriptions as the arguments to the *CombinedTestData* method.

Corresponding analysis keywords

- **COMBINED TEST DATA*

Concrete object

The Concrete object defines concrete properties beyond the elastic range.

Access

```
import material
mdb.models[name].materials[name].concrete
import odbMaterial
session.odbs[name].materials[name].concrete
```

Concrete(...)

This method creates a Concrete object.

Path

```
mdb.models[name].materials[name].Concrete
session.odbs[name].materials[name].Concrete
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Absolute value of compressive stress.
- Absolute value of plastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Concrete object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Concrete object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Concrete* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Concrete object has members with the same names and descriptions as the arguments to the *Concrete* method.

In addition, the Concrete object can have the following members:

failureRatios	A *FailureRatios* object.
shearRetention	A *ShearRetention* object.
tensionStiffening	A *TensionStiffening* object.

Corresponding analysis keywords

- **CONCRETE*

ConcreteCompressionDamage object

The ConcreteCompressionDamage object specifies hardening for the concrete damaged plasticity model.

Access

```
import material
mdb.models[name].materials[name].concreteDamagedPlasticity\
.concreteCompressionDamage
import odbMaterial
session.odbs[name].materials[name].concreteDamagedPlasticity\
.concreteCompressionDamage
```

ConcreteCompressionDamage(...)

This method creates a ConcreteCompressionDamage object.

Path

```
mdb.models[name].materials[name].concreteDamagedPlasticity\
.ConcreteCompressionDamage
session.odbs[name].materials[name].concreteDamagedPlasticity\
.ConcreteCompressionDamage
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

tensionRecovery
A Float specifying the value of the stiffness recovery factor, w_t, that determines the amount of tension stiffness that is recovered as loading changes from compression to tension. The default value is 0.0.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Compressive damage variable, d_c.
- Inelastic (crushing) strain, ϵ_c^{in}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteCompressionDamage object.

Exceptions

RangeError.

setValues(...)

This method modifies the ConcreteCompressionDamage object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcreteCompressionDamage* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ConcreteCompressionDamage object has members with the same names and descriptions as the arguments to the *ConcreteCompressionDamage* method.

Corresponding analysis keywords

- **CONCRETE COMPRESSION DAMAGE*

ConcreteCompressionHardening object

The ConcreteCompressionHardening object specifies hardening for the concrete damaged plasticity model.

Access

```
import material
mdb.models[name].materials[name].concreteDamagedPlasticity\
.concreteCompressionHardening
import odbMaterial
session.odbs[name].materials[name].concreteDamagedPlasticity\
.concreteCompressionHardening
```

ConcreteCompressionHardening(...)

This method creates a ConcreteCompressionHardening object.

Path

```
mdb.models[name].materials[name].concreteDamagedPlasticity\
.ConcreteCompressionHardening
session.odbs[name].materials[name].concreteDamagedPlasticity\
.ConcreteCompressionHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

rate A Boolean specifying whether the data depend on rate. The default value is OFF.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in compression, σ_c.
- Inelastic (crushing) strain, ϵ_c^{in}.
- Inelastic (crushing) strain rate, $\dot{\epsilon}_c^{in}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteCompressionHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the ConcreteCompressionHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcreteCompressionHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ConcreteCompressionHardening object has members with the same names and descriptions as the arguments to the *ConcreteCompressionHardening* method.

Corresponding analysis keywords

- **CONCRETE COMPRESSION HARDENING*

ConcreteDamagedPlasticity object

The ConcreteDamagedPlasticity object specifies the concrete damaged plasticity model.

Access

```
import material
mdb.models[name].materials[name].concreteDamagedPlasticity
import odbMaterial
session.odbs[name].materials[name].concreteDamagedPlasticity
```

ConcreteDamagedPlasticity(...)

This method creates a ConcreteDamagedPlasticity object.

Path

```
mdb.models[name].materials[name].ConcreteDamagedPlasticity
session.odbs[name].materials[name].ConcreteDamagedPlasticity
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Dilation angle, ψ (in degrees) in the p–q plane.
- Flow potential eccentricity, ϵ. The default value is 0.1.
- σ_{b0}/σ_{t0}, the ratio of initial equibiaxial compressive yield stress to initial uniaxial compressive yield stress. The default value is 1.16.
- K_c, the ratio of the second stress invariant on the tensile meridian, to that on the compressive meridian, at initial yield for any given value of the pressure invariant p such that the maximum principal stress is negative. The default value is 2/3.
- Viscosity parameter, μ, used for the viscoplastic regularization of the concrete constitutive equations in an Abaqus/Standard analysis. This parameter is ignored in an Abaqus/Explicit analysis. The default value is 0.0.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

Return value

A ConcreteDamagedPlasticity object.

Exceptions

RangeError.

setValues(...)

This method modifies the ConcreteDamagedPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcreteDamagedPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ConcreteDamagedPlasticity object has members with the same names and descriptions as the arguments to the *ConcreteDamagedPlasticity* method.

In addition, the ConcreteDamagedPlasticity object can have the following members:

concreteCompressionHardening	A *ConcreteCompressionHardening* object.
concreteTensionStiffening	A *ConcreteTensionStiffening* object.
concreteCompressionDamage	A *ConcreteCompressionDamage* object.
concreteTensionDamage	A *ConcreteTensionDamage* object.

Corresponding analysis keywords

- **CONCRETE DAMAGED PLASTICITY*

ConcreteTensionDamage object

The ConcreteTensionDamage object specifies hardening for the concrete damaged plasticity model.

Access

```
import material
mdb.models[name].materials[name].concreteDamagedPlasticity\
.concreteTensionDamage
import odbMaterial
session.odbs[name].materials[name].concreteDamagedPlasticity\
.concreteTensionDamage
```

ConcreteTensionDamage(...)

This method creates a ConcreteTensionDamage object.

Path

```
mdb.models[name].materials[name].concreteDamagedPlasticity\
.ConcreteTensionDamage
session.odbs[name].materials[name].concreteDamagedPlasticity\
.ConcreteTensionDamage
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

compressionRecovery
A Float specifying the value of the stiffness recovery factor, w_c, that determines the amount of compression stiffness that is recovered as loading changes from tension to compression. The default value is 1.0.

type
A SymbolicConstant specifying the type of tensile damage data. Set *type*=STRAIN to specify the tensile damage variable as a function of cracking strain. Set *type*=DISPLACEMENT to specify the tensile damage variable as a function of cracking displacement. Possible values are STRAIN and DISPLACEMENT. The default value is STRAIN.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Tensile damage variable, d_t.
- Direct cracking strain, ϵ_t^{ck}.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Tensile damage variable, d_t.
- Direct cracking displacement, u_t^{ck}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteTensionDamage object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ConcreteTensionDamage object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcreteTensionDamage* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ConcreteTensionDamage object has members with the same names and descriptions as the arguments to the *ConcreteTensionDamage* method.

Corresponding analysis keywords

- **CONCRETE TENSION DAMAGE*

ConcreteTensionStiffening object

The ConcreteTensionStiffening object specifies hardening for the concrete damaged plasticity model.

Access

```
import material
mdb.models[name].materials[name].concreteDamagedPlasticity\
.concreteTensionStiffening
import odbMaterial
session.odbs[name].materials[name].concreteDamagedPlasticity\
.concreteTensionStiffening
```

ConcreteTensionStiffening(...)

This method creates a ConcreteTensionStiffening object.

Path

```
mdb.models[name].materials[name].concreteDamagedPlasticity\
.ConcreteTensionStiffening
session.odbs[name].materials[name].concreteDamagedPlasticity\
.ConcreteTensionStiffening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

rate A Boolean specifying whether the data depend on rate. The default value is OFF.

type A SymbolicConstant specifying the type of postcracking behavior data. Possible values are:

- STRAIN, specifying postfailure stress as a function of cracking strain.
- DISPLACEMENT, specifying postfailure stress as a function of cracking displacement.
- GFI, specifying failure stress as a function of the fracture energy.

The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Remaining direct stress after cracking, σ_t.
- Direct cracking strain, ϵ_t^{ck}.

- Direct cracking strain rate, $\dot{\epsilon}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Remaining direct stress after cracking, σ_t.
- Direct cracking displacement, u_t^{ck}.
- Direct cracking displacement rate, $\dot{u}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=GFI, the table data specify the following:

- Failure stress, σ_{t0}.
- Fracture energy, G_f.
- Direct cracking displacement rate, $\dot{u}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteTensionStiffening object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ConcreteTensionStiffening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConcreteTensionStiffening* method.

Return value
`None.`

Exceptions
RangeError.

Members
The ConcreteTensionStiffening object has members with the same names and descriptions as the arguments to the *ConcreteTensionStiffening* method.

Corresponding analysis keywords
- **CONCRETE TENSION STIFFENING*

Conductivity object

The Conductivity object specifies thermal conductivity.

Access

```
import material
mdb.models[name].materials[name].conductivity
import odbMaterial
session.odbs[name].materials[name].conductivity
```

Conductivity(...)

This method creates a Conductivity object.

Path

```
mdb.models[name].materials[name].Conductivity
session.odbs[name].materials[name].Conductivity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of conductivity. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Conductivity, k.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- k_{11}.
- k_{22}.

- k_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- k_{11}.
- k_{12}.
- k_{22}.
- k_{13}.
- k_{23}.
- k_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Conductivity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Conductivity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Conductivity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Conductivity object has members with the same names and descriptions as the arguments to the *Conductivity* method.

Corresponding analysis keywords

- **CONDUCTIVITY*

ContactArea object

A ContactArea object specifies a suboption of gasket thickness behavior when *variableUnits*=FORCE on the GasketThicknessBehavior object. The ContactArea object defines the contact area or contact width versus closure curves to output an average pressure through variable CS11.

Access

```
import material
mdb.models[name].materials[name].gasketThicknessBehavior.contactArea
import odbMaterial
session.odbs[name].materials[name].gasketThicknessBehavior.contactArea
```

ContactArea(...)

This method creates a ContactArea object.

Path

```
mdb.models[name].materials[name].gasketThicknessBehavior.ContactArea
session.odbs[name].materials[name].gasketThicknessBehavior.ContactArea
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether contact area data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies included in the definition of the contact area data, in addition to temperature. The default value is 0.

Table data

- Contact area or width; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ContactArea object.

Exceptions

None.

`setValues(...)`

This method modifies the ContactArea object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ContactArea* method.

Return value

`None.`

Exceptions

None.

Members

The ContactArea object has members with the same names and descriptions as the arguments to the *ContactArea* method.

Corresponding analysis keywords

- **GASKET CONTACT AREA*

Creep object

The Creep object defines a creep law.

Access

```
import material
mdb.models[name].materials[name].creep
import odbMaterial
session.odbs[name].materials[name].creep
```

Creep(...)

This method creates a Creep object.

Path

```
mdb.models[name].materials[name].Creep
session.odbs[name].materials[name].Creep
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

law A SymbolicConstant specifying the strain-hardening law. Possible values are STRAIN, TIME, HYPERBOLIC_SINE, USER, ANAND, DARVEAUX,DOUBLE_POWER, POWER_LAW, and TIME_POWER_LAW. The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

time A SymbolicConstant specifying the time interval for relevant laws. Possible values are CREEP and TOTAL. The default value is TOTAL.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=HYPERBOLIC_SINE, the table data specify the following:

- A.
- B.
- n.
- $\triangle H$, if the data depend on temperature.
- R.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=ANAND, the table data specify the following:

- s_1.
- $\frac{Q}{R}$.
- A.
- ξ.
- m.
- A_0.
- $\widehat{S}$.
- n.
- a.
- S_2.
- S_3.
- A_1.
- A_2.
- A_3.
- A_4.

If *law*=DARVEAUX, the table data specify the following:

- C_{ss}.
- $\frac{Q}{R}$.
- α.
- n.
- ϵ_T.
- B.

If *law*=DOUBLE_POWER, the table data specify the following:

- A_1.
- B_1.
- C_1.
- A_2.
- B_2.
- C_2.
- σ_0.

If *law*=POWER_LAW or *law*=TIME_POWER_LAW, the table data specify the following:

- q_0.
- n.
- m.
- $\dot{\varepsilon}_0$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Creep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Creep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Creep* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Creep object has members with the same names and descriptions as the arguments to the *Creep* method.

In addition, the Creep object can have the following members:

ornl

An *Ornl* object.

potential

A *Potential* object.

Corresponding analysis keywords

- **CREEP*

CrushableFoam object

The CrushableFoam object specifies the crushable foam plasticity model.

Access

```
import material
mdb.models[name].materials[name].crushableFoam
import odbMaterial
session.odbs[name].materials[name].crushableFoam
```

CrushableFoam(...)

This method creates a CrushableFoam object.

Path

```
mdb.models[name].materials[name].CrushableFoam
session.odbs[name].materials[name].CrushableFoam
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

hardening A SymbolicConstant specifying the type of hardening/softening definition. Possible values are VOLUMETRIC and ISOTROPIC. The default value is VOLUMETRIC.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *hardening*=VOLUMETRIC, the table data specify the following:

- Ratio, k, of initial yield stress in uniaxial compression, σ_c^0, to initial yield stress in hydrostatic compression, p_c^0; $0.0 < k < 3.0$.
- Ratio, k_t, of yield stress in hydrostatic tension, p_t, to initial yield stress in hydrostatic compression, p_c^0. The default value is 1.0.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=ISOTROPIC, the table data specify the following:

- Ratio, k, of initial yield stress in uniaxial compression, σ_c^0, to initial yield stress in hydrostatic compression, p_c^0; $0.0 \leq k \leq 3.0$.
- Plastic Poisson's ratio.ν_p; $-1 \leq \nu_p \leq 0.5$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CrushableFoam object.

Exceptions

RangeError.

setValues(...)

This method modifies the CrushableFoam object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CrushableFoam* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CrushableFoam object has members with the same names and descriptions as the arguments to the *CrushableFoam* method.

In addition, the CrushableFoam object can have the following members:

crushableFoamHardening A *CrushableFoamHardening* object.

rateDependent A *RateDependent* object.

Corresponding analysis keywords

- **CRUSHABLE FOAM*

CrushableFoamHardening object

The CrushableFoamHardening object specifies hardening for the crushable foam plasticity model.

Access

```
import material
mdb.models[name].materials[name].crushableFoam.crushableFoamHardening
import odbMaterial
session.odbs[name].materials[name].crushableFoam\
.crushableFoamHardening
```

CrushableFoamHardening(...)

This method creates a CrushableFoamHardening object.

Path

```
mdb.models[name].materials[name].crushableFoam.CrushableFoamHardening
session.odbs[name].materials[name].crushableFoam\
.CrushableFoamHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The yield stress in uniaxial compression, σ_c.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CrushableFoamHardening object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CrushableFoamHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CrushableFoamHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The CrushableFoamHardening object has members with the same names and descriptions as the arguments to the *CrushableFoamHardening* method.

Corresponding analysis keywords

- **CRUSHABLE FOAM HARDENING*

CrushStress object

The CrushStress object specifies the crush stress of a material.

Access

```
import material
mdb.models[name].materials[name].crushStress
import odbMaterial
session.odbs[name].materials[name].crushStress
```

CrushStress(...)

This method creates a CrushStress object.

Path

```
mdb.models[name].materials[name].CrushStress
session.odbs[name].materials[name].CrushStress
```

Required arguments

crushStressTable A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Crush Stress.
- Material angle in degrees.
- Temperature if the data depend on temperature.
- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CrushStress object.

Exceptions

None.

`setValues(...)`

This method modifies the CrushStress object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CrushStress* method.

Return value

`None.`

Exceptions

None.

Members

The CrushStress object has members with the same names and descriptions as the arguments to the *CrushStress* method.

In addition, the CrushStress object can have the following members:

crushStressVelocityFactor A *CrushStressVelocityFactor* object.

Corresponding analysis keywords

- **CRUSH STRESS*

CrushStressVelocityFactor object

The CrushStressVelocityFactor object defines how the approach velocity at a crushing interface influences a material's resistance to crushing.

Access

```
import material
mdb.models[name].materials[name].crushStress.crushStressVelocityFactor
import odbMaterial
session.odbs[name].materials[name].crushStress.crushStressVelocityFactor
```

CrushStressVelocityFactor(...)

This method creates a CrushStressVelocityFactor object.

Path

```
mdb.models[name].materials[name].crushStress.CrushStressVelocityFactor
session.odbs[name].materials[name].crushStress.CrushStressVelocityFactor
```

Required arguments

crushStressVelocityFactorTable	A sequence of sequences of Floats specifying the items described below.

Table data

The table data specify the following:

- Scaling factor.
- Relative velocity.

Return value

A CrushStressVelocityFactor object.

Exceptions

None.

setValues(...)

This method modifies the CrushStressVelocityFactor object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CrushStressVelocityFactor* method.

Return value
`None.`

Exceptions
None.

Members
The CrushStressVelocityFactor object has members with the same names and descriptions as the arguments to the *CrushStressVelocityFactor* method.

Corresponding analysis keywords
- **CRUSH STRESS VELOCITY FACTOR*

CycledPlastic object

The CycledPlastic object specifies cycled yield stress data for the ORNL constitutive model.

Access

```
import material
mdb.models[name].materials[name].plastic.cycledPlastic
import odbMaterial
session.odbs[name].materials[name].plastic.cycledPlastic
```

CycledPlastic(...)

This method creates a CycledPlastic object.

Path

```
mdb.models[name].materials[name].plastic.CycledPlastic
session.odbs[name].materials[name].plastic.CycledPlastic
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

Table data

- Yield stress.
- Plastic strain.
- Temperature, if the data depend on temperature.

Return value

A CycledPlastic object.

Exceptions

None.

setValues(...)

This method modifies the CycledPlastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CycledPlastic* method.

Return value

`None.`

Exceptions

None.

Members

The CycledPlastic object has members with the same names and descriptions as the arguments to the *CycledPlastic* method.

Corresponding analysis keywords

- **CYCLED PLASTIC*

CyclicHardening object

The CyclicHardening object defines the evolution of the elastic domain for the nonlinear isotropic/kinematic hardening model.

Access

```
import material
mdb.models[name].materials[name].plastic.cyclicHardening
import odbMaterial
session.odbs[name].materials[name].plastic.cyclicHardening
```

CyclicHardening(...)

This method creates a CyclicHardening object.

Path

```
mdb.models[name].materials[name].plastic.CyclicHardening
session.odbs[name].materials[name].plastic.CyclicHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

parameters A Boolean specifying whether material parameters are to be input directly. The default value is OFF.

Table data

- Equivalent stress.
- Q_∞ (only if *parameters*=ON).
- Hardening parameter (only if *parameters*=ON).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CyclicHardening object.

Exceptions

None.

`setValues(...)`

This method modifies the CyclicHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CyclicHardening* method.

Return value

`None.`

Exceptions

None.

Members

The CyclicHardening object has members with the same names and descriptions as the arguments to the *CyclicHardening* method.

Corresponding analysis keywords

- **CYCLIC HARDENING*

DamageEvolution object

The DamageEvolution object specifies material properties to define the evolution of damage.

Access

```
import material
mdb.models[name].materials[name].ductileDamageInitiation\
.damageEvolution
mdb.models[name].materials[name].fldDamageInitiation.damageEvolution
mdb.models[name].materials[name].flsdDamageInitiation.damageEvolution
mdb.models[name].materials[name].hashinDamageInitiation\
.damageEvolution
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.damageEvolution
mdb.models[name].materials[name].maxeDamageInitiation.damageEvolution
mdb.models[name].materials[name].maxpeDamageInitiation.damageEvolution
mdb.models[name].materials[name].maxpsDamageInitiation.damageEvolution
mdb.models[name].materials[name].maxsDamageInitiation.damageEvolution
mdb.models[name].materials[name].mkDamageInitiation.damageEvolution
mdb.models[name].materials[name].msfldDamageInitiation.damageEvolution
mdb.models[name].materials[name].quadeDamageInitiation.damageEvolution
mdb.models[name].materials[name].quadsDamageInitiation.damageEvolution
mdb.models[name].materials[name].shearDamageInitiation.damageEvolution
import odbMaterial
session.odbs[name].materials[name].ductileDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].fldDamageInitiation.damageEvolution
session.odbs[name].materials[name].flsdDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].hashinDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].maxeDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].maxpeDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].maxpsDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].maxsDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].mkDamageInitiation.damageEvolution
session.odbs[name].materials[name].msfldDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].quadeDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].quadsDamageInitiation\
.damageEvolution
session.odbs[name].materials[name].shearDamageInitiation\
.damageEvolution
```

DamageEvolution(...)

This method creates a DamageEvolution object.

Path

```
mdb.models[name].materials[name].ductileDamageInitiation\
.DamageEvolution
mdb.models[name].materials[name].fldDamageInitiation.DamageEvolution
mdb.models[name].materials[name].flsdDamageInitiation.DamageEvolution
mdb.models[name].materials[name].hashinDamageInitiation\
.DamageEvolution
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.DamageEvolution
mdb.models[name].materials[name].maxeDamageInitiation.DamageEvolution
mdb.models[name].materials[name].maxpeDamageInitiation.DamageEvolution
mdb.models[name].materials[name].maxpsDamageInitiation.DamageEvolution
mdb.models[name].materials[name].maxsDamageInitiation.DamageEvolution
mdb.models[name].materials[name].mkDamageInitiation.DamageEvolution
mdb.models[name].materials[name].msfldDamageInitiation.DamageEvolution
mdb.models[name].materials[name].quadeDamageInitiation.DamageEvolution
mdb.models[name].materials[name].quadsDamageInitiation.DamageEvolution
mdb.models[name].materials[name].shearDamageInitiation.DamageEvolution
session.odbs[name].materials[name].ductileDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].fldDamageInitiation.DamageEvolution
session.odbs[name].materials[name].flsdDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].hashinDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].maxeDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].maxpeDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].maxpsDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].maxsDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].mkDamageInitiation.DamageEvolution
session.odbs[name].materials[name].msfldDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].quadeDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].quadsDamageInitiation\
.DamageEvolution
session.odbs[name].materials[name].shearDamageInitiation\
.DamageEvolution
```

Required arguments

type

A SymbolicConstant specifying the type of damage evolution. Possible values are DISPLACEMENT and ENERGY.

table

A sequence of sequences of Floats specifying the items described below.

Optional arguments

degradation
A SymbolicConstant specifying the degradation. Possible values are MAXIMUM and MULTIPLICATIVE. The default value is MAXIMUM.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

mixedModeBehavior
A SymbolicConstant specifying the mixed mode behavior. Possible values are MODE_INDEPENDENT, TABULAR, POWER_LAW, and BK. The default value is MODE_INDEPENDENT.

modeMixRatio
A SymbolicConstant specifying the mode mix ratio. Possible values are ENERGY and TRACTION. The default value is ENERGY.

power
`None` or a Float specifying the exponent in the power law or the Benzeggagh-Kenane criterion that defines the variation of fracture energy with mode mix for cohesive elements. The default value is `None`.

softening
A SymbolicConstant specifying the softening. Possible values are LINEAR, EXPONENTIAL, and TABULAR. The default value is LINEAR.

Table data

If *type*=DISPLACEMENT, and *softening*=LINEAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Equivalent total or plastic displacement at failure, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=LINEAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.
- Appropriate mode mix ratio.

- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Fracture energy.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Equivalent total or plastic displacement at failure, measured from the time of damage initiation.
- Exponential law parameter.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.
- Exponential law parameter.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Fracture energy.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=TABULAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Damage variable.
- Equivalent total or plastic displacement, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=TABULAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Damage variable.
- Equivalent total or plastic displacement, measured from the time of damage initiation.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR or EXPONENTIAL, and *mixedModeBehavior*=POWER_LAW or BK, the table data specify the following:

- Normal mode fracture energy.
- Shear mode fracture energy for failure in the first shear direction.
- Shear mode fracture energy for failure in the second shear direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, *softening*=LINEAR and constructor for *DamageInitiation*=HashinDamageInitiation the table data specify the following:

- Fiber tensile fracture energy.
- Fiber compressive fracture energy.
- Matrix tensile fracture energy.
- Matrix compressive fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DamageEvolution object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DamageEvolution object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DamageEvolution* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DamageEvolution object has members with the same names and descriptions as the arguments to the *DamageEvolution* method.

Corresponding analysis keywords

- **DAMAGE EVOLUTION*

DamageInitiation object

The DamageInitiation object specifies material properties to define the initiation of damage.

Access

```
import material
mdb.models[name].materials[name].ductileDamageInitiation
mdb.models[name].materials[name].fldDamageInitiation
mdb.models[name].materials[name].flsdDamageInitiation
mdb.models[name].materials[name].hashinDamageInitiation
mdb.models[name].materials[name].johnsonCookDamageInitiation
mdb.models[name].materials[name].maxeDamageInitiation
mdb.models[name].materials[name].maxpeDamageInitiation
mdb.models[name].materials[name].maxpsDamageInitiation
mdb.models[name].materials[name].maxsDamageInitiation
mdb.models[name].materials[name].mkDamageInitiation
mdb.models[name].materials[name].msfldDamageInitiation
mdb.models[name].materials[name].quadeDamageInitiation
mdb.models[name].materials[name].quadsDamageInitiation
mdb.models[name].materials[name].shearDamageInitiation
mdb.models[name].materials[name].larc05DamageInitiation
mdb.models[name].materials[name].HosfordCoulombDamageInitiation
import odbMaterial
session.odbs[name].materials[name].ductileDamageInitiation
session.odbs[name].materials[name].fldDamageInitiation
session.odbs[name].materials[name].flsdDamageInitiation
session.odbs[name].materials[name].hashinDamageInitiation
session.odbs[name].materials[name].johnsonCookDamageInitiation
session.odbs[name].materials[name].maxeDamageInitiation
session.odbs[name].materials[name].maxpeDamageInitiation
session.odbs[name].materials[name].maxpsDamageInitiation
session.odbs[name].materials[name].maxsDamageInitiation
session.odbs[name].materials[name].mkDamageInitiation
session.odbs[name].materials[name].msfldDamageInitiation
session.odbs[name].materials[name].quadeDamageInitiation
session.odbs[name].materials[name].quadsDamageInitiation
session.odbs[name].materials[name].shearDamageInitiation
session.odbs[name].materials[name].laRC05DamageInitiation
session.odbs[name].materials[name].HosfordCoulombDamageInitiation
```

DuctileDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].DuctileDamageInitiation
session.odbs[name].materials[name].DuctileDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

lodeDependency A Boolean specifying whether the data depend on lode dependent data. The default value is OFF.

accumulationPower A Float specifying the value of the power coefficient, *N*. The default value is 0.0.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Table data

If the constructor is DuctileDamageInitiation, the table data specify the following:

- Equivalent fracture strain at damage initiation.
- Stress triaxiality.

- Lode-dependent, if the data depend on lode angle.
- Strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is FldDamageInitiation, the table data specify the following:

- Major principal strain at damage initiation.
- Minor principal strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor FlsdDamageInitiation, the table data specify the following:

- Major principal stress at damage initiation.
- Minor principal stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is JohnsonCookDamageInitiation, the table data specify the following:

- Johnson-Cook failure parameter D1.
- Johnson-Cook failure parameter D2.
- Johnson-Cook failure parameter D3.
- Johnson-Cook failure parameter D4.
- Johnson-Cook failure parameter D5.
- Melting temperature.
- Transition temperature.
- Reference strain rate.

If the constructor is MkDamageInitiation, the table data specify the following:

- Flaw size relative to the nominal thickness of the section.
- Angle (in degrees) with respect to the 1-direction of the local material orientation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is MsfldDamageInitiation and *definition*=MSFLD, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Equivalent plastic strain at initiation of localized necking.
- Ratio of minor to major principal strains.
- Equivalent plastic strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is MsfldDamageInitiation and *definition*=FLD, the table data specify the following:

- Major principal strain at initiation of localized necking.
- Equivalent plastic strain at initiation of localized necking.
- Ratio of minor to major principal strains.
- Equivalent plastic strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is QuadeDamageInitiation or MaxeDamageInitiation, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the first shear direction.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the second shear direction.
- Rate-dependent, if the data depend on the effective rate of separation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is QuadsDamageInitiation or MaxsDamageInitiation, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the first shear direction.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the second shear direction.
- Rate-dependent, if the data depend on the effective rate of separation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is MaxpeDamageInitiation, the table data specify the following:

- Maximum principal strain at damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is MaxpsDamageInitiation, the table data specify the following:

- Maximum principal stress at damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is ShearDamageInitiation, the table data specify the following:

- Equivalent fracture strain at damage initiation.
- Shear stress ratio.
- Strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is HosfordCoulombDamageInitiation, the table data specify the following:

- a.
- b.
- c.
- n.
- d.
- reference strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If the constructor is HashinDamageInitiation, the table data specify the following:

- Fiber tensile strength.
- Fiber compressive strength.
- Matrix tensile strength.
- Matrix compressive strength.
- Longitudinal shear strength.
- Transverse shear strength.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.
- If the constructor is LaRC05DamageInitiation, the table data specify the following:
  - Longitudinal tensile strength.
  - Longitudinal compressive strength.
  - Transverse tensile strength.
  - Transverse compressive strength.
  - In Plane shear strength.
  - Fracture plane angle for pure compression.
  - Misalignment angle at failure for pure compression
  - Transverse shear strength.
  - Longitudinal shear friction coefficient.
  - Transverse shear friction coefficient.
  - Temperature, if the data depend on temperature.
  - Value of the first field variable, if the data depend on field variables.
  - Value of the second field variable.
  - Etc.

Return value

A DamageInitiation object.

Exceptions

RangeError.

FldDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].FldDamageInitiation
session.odbs[name].materials[name].FldDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

FlsdDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].FlsdDamageInitiation
session.odbs[name].materials[name].FlsdDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

JohnsonCookDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].JohnsonCookDamageInitiation
session.odbs[name].materials[name].JohnsonCookDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

accumulationPower A Float specifying the value of the power coefficient, N. The default value is 0.0.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MaxeDamageInitiation
session.odbs[name].materials[name].MaxeDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

growthTolerance A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

rateDependency A Boolean specifying whether the data depend on the effective rate of separation. The default value is OFF.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, and COMBINED. The default value is CENTROID.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MaxsDamageInitiation
session.odbs[name].materials[name].MaxsDamageInitiation
```

Required arguments

table
A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition
A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq
A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Float specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

growthTolerance
A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance
A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

rateDependency
A Boolean specifying whether the data depend on the effective rate of separation. The default value is OFF.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, and COMBINED. The default value is CENTROID.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MkDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MkDamageInitiation
session.odbs[name].materials[name].MkDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MsfldDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MsfldDamageInitiation
session.odbs[name].materials[name].MsfldDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

peinc A Double specifying the accumulated increment in equivalent plastic strain used to trigger the evaluation of the MSFLD damage initiation criterion. The default value is 0.002.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

QuadeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].QuadeDamageInitiation
session.odbs[name].materials[name].QuadeDamageInitiation
```

Required arguments

table
A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition
A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq
A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Float specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

growthTolerance
A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance
A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

rateDependency
A Boolean specifying whether the data depend on the effective rate of separation. The default value is OFF.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, and COMBINED. The default value is CENTROID.

Return value

A DamageInitiation object.

Exceptions

RangeError.

QuadsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].QuadsDamageInitiation
session.odbs[name].materials[name].QuadsDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

growthTolerance A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

rateDependency A Boolean specifying whether the data depend on the effective rate of separation. The default value is OFF.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, and COMBINED. The default value is CENTROID.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxpeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MaxpeDamageInitiation
session.odbs[name].materials[name].MaxpeDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, COMBINED, and NONLOCAL. The default value is CENTROID.

growthTolerance A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

anglemax
A Float specifying the maximum allowed change in the crack propagation angle (in degrees) between the new crack propagation direction and the previous crack propagation direction. The default value is 85.0.

angSmooth
A Float specifying the maximum allowed difference (in degrees) below which the normals of the crack facets are included in the moving least-squares approximation to smooth out the crack normals to obtain the crack propagation direction along the crack front. The default value is 70.0.

rCrackDirection
A Float specifying the radius around the crack tip within which the elements are included for calculating the averaged stress/strain and for smoothing out the normals of the individual crack facets along the crack front used to obtain the crack propagation direction. The default value is 0.0.

iniSmooth
An SymbolicConstant specifying the damage initiation inismooth. Possible values are YES and NO. The default value is YES.

smoothing
An SymbolicConstant specifying the damage initiation smoothing. Possible values are NONE and NODAL. The default value is NONE.

npoly
An SymbolicConstant specifying the number of terms in the polynomial used for the moving least-squares approximation to smooth out the crack normals. Possible values are QUADRATIC, LEAST-SQUARE, LINEAR, and CUBIC. The default value is QUADRATIC.

weightingMethod
An SymbolicConstant specifying the damage initiation weighting method. Possible values are UNIFORM, GAUSS, CUBIC, and USER. The default value is UNIFORM.

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxpsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].MaxpsDamageInitiation
session.odbs[name].materials[name].MaxpsDamageInitiation
```

Required arguments

table
A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, COMBINED, and NONLOCAL. The default value is CENTROID.

growthTolerance A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

anglemax
A Float specifying the maximum allowed change in the crack propagation angle (in degrees) between the new crack propagation direction and the previous crack propagation direction. The default value is 85.0.

angSmooth
A Float specifying the maximum allowed difference (in degrees) below which the normals of the crack facets are included in the moving least-squares approximation to smooth out the crack normals to obtain the crack propagation direction along the crack front. The default value is 70.0.

rCrackDirection
A Float specifying the radius around the crack tip within which the elements are included for calculating the averaged stress/strain and for smoothing out the normals of the individual crack facets along the crack front used to obtain the crack propagation direction. The default value is 0.0.

iniSmooth
An SymbolicConstant specifying the damage initiation inismooth. Possible values are YES and NO. The default value is YES.

smoothing
An SymbolicConstant specifying the damage initiation smoothing. Possible values are NONE and NODAL. The default value is NONE.

npoly
An SymbolicConstant specifying the number of terms in the polynomial used for the moving least-squares approximation to smooth out the crack normals. Possible values are QUADRATIC, LEAST-SQUARE, LINEAR, and CUBIC. The default value is QUADRATIC.

weightingMethod
An SymbolicConstant specifying the damage initiation weighting method. Possible values are UNIFORM, GAUSS, CUBIC, and USER. The default value is UNIFORM.

Return value

A DamageInitiation object.

Exceptions

RangeError.

ShearDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].ShearDamageInitiation
session.odbs[name].materials[name].ShearDamageInitiation
```

Required arguments

table
A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

accumulationPower A Float specifying the value of the power coefficient, N. The default value is 0.0.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

HosfordCoulombDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].HosfordCoulombDamageInitiation
session.odbs[name].materials[name].HosfordCoulombDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

accumulationPower A Float specifying the value of the power coefficient, N. The default value is 0.0.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

HashinDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].HashinDamageInitiation
session.odbs[name].materials[name].HashinDamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

LaRC05DamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
mdb.models[name].materials[name].LaRC05DamageInitiation
session.odbs[name].materials[name].LaRC05DamageInitiation
```

Required arguments

table A sequence of sequences of Floats specifying the items described in the "Table data" section.

Optional arguments

definition A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Float specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

Return value

A DamageInitiation object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DamageInitiation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DamageInitiation* method.

Return value

`None.`

Exceptions

None.

Members

The DamageInitiation object can have the following members:

definition
A SymbolicConstant specifying the damage initiation definition. Possible values are FLD and MSFLD. The default value is MSFLD.

feq
A Float specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Float specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Float specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Float specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Float specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Float specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Float specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
A SymbolicConstant specifying the damage initiation direction. Possible values are NMORI and TMORI. The default value is NMORI.

position
An SymbolicConstant specifying the damage initiation position. Possible values are CENTROID, CRACKTIP, COMBINED, and NONLOCAL. The default value is CENTROID.

growthTolerance
A Float specifying a separate tolerance used for crack growth within which the damage initiation criterion must be satisfied. The default value is equal to tolerance.

unstableGrowthTolerance
A Float specifying a separate tolerance used for unstable crack growth within which the damage initiation criterion must be satisfied. The default value is infinity.

anglemax
A Float specifying the maximum allowed change in the crack propagation angle (in degrees) between the new crack propagation direction and the previous crack propagation direction. The default value is 85.0.

angSmooth
A Float specifying the maximum allowed difference (in degrees) below which the normals of the crack facets are included in the moving least-squares approximation to smooth out the crack normals to obtain the crack propagation direction along the crack front. The default value is 70.0.

rCrackDirection
A Float specifying the radius around the crack tip within which the elements are included for calculating the averaged stress/strain and for smoothing out the normals of the individual crack facets along the crack front used to obtain the crack propagation direction. The default value is 0.0.

iniSmooth
An SymbolicConstant specifying the damage initiation inismooth. Possible values are YES and NO. The default value is YES.

smoothing
An SymbolicConstant specifying the damage initiation smoothing. Possible values are NONE and NODAL. The default value is NONE.

npoly
An SymbolicConstant specifying the number of terms in the polynomial used for the moving least-squares approximation to smooth out the crack normals. Possible values are QUADRATIC, LEAST-SQUARE, LINEAR, and CUBIC. The default value is QUADRATIC.

weightingMethod
An SymbolicConstant specifying the damage initiation weighting method. Possible values are UNIFORM, GAUSS, CUBIC, and USER. The default value is UNIFORM.

table
A tuple of tuples of Floats specifying the items described in the "Table data" section.

damageEvolution
A *DamageEvolution* object.

damageStabilization
A *DamageStabilization* object.

damageStabilizationCohesive
A *DamageStabilizationCohesive* object.

Corresponding analysis keywords

- **DAMAGE INITIATION*

DamageStabilization object

The DamageStabilization object specifies the viscosity coefficients for the damage model for fiber-reinforced materials.

Access

```
import material
mdb.models[name].materials[name].ductileDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].fldDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].flsdDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].hashinDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].maxeDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].maxpeDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].maxpsDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].maxsDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].mkDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].msfldDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].quadeDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].quadsDamageInitiation\
.damageStabilization
mdb.models[name].materials[name].shearDamageInitiation\
.damageStabilization
import odbMaterial
session.odbs[name].materials[name].ductileDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].fldDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].flsdDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].hashinDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].maxeDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].maxpeDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].maxpsDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].maxsDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].mkDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].msfldDamageInitiation\
```

```
.damageStabilization
session.odbs[name].materials[name].quadeDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].quadsDamageInitiation\
.damageStabilization
session.odbs[name].materials[name].shearDamageInitiation\
.damageStabilization
```

DamageStabilization(...)

This method creates a DamageStabilization object.

Path

```
mdb.models[name].materials[name].ductileDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].fldDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].flsdDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].hashinDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].maxeDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].maxpeDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].maxpsDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].maxsDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].mkDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].msfldDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].quadeDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].quadsDamageInitiation\
.DamageStabilization
mdb.models[name].materials[name].shearDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].ductileDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].fldDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].flsdDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].hashinDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].maxeDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].maxpeDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].maxpsDamageInitiation\
```

```
.DamageStabilization
session.odbs[name].materials[name].maxsDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].mkDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].msfldDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].quadeDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].quadsDamageInitiation\
.DamageStabilization
session.odbs[name].materials[name].shearDamageInitiation\
.DamageStabilization
```

Required arguments

fiberTensileCoeff A Float specifying the viscosity coefficient for the fiber tensile failure mode.

fiberCompressiveCoeff A Float specifying the viscosity coefficient for the fiber compressive failure mode.

matrixTensileCoeff A Float specifying the viscosity coefficient for the matrix tensile failure mode.

matrixCompressiveCoeff A Float specifying the viscosity coefficient for the matrix compressive failure mode.

Optional arguments

None.

Return value

A DamageStabilization object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DamageStabilization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DamageStabilization* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DamageStabilization object has members with the same names and descriptions as the arguments to the *DamageStabilization* method.

Corresponding analysis keywords

- **DAMAGE STABILIZATION*

DamageStabilizationCohesive object

The DamageStabilizationCohesive object specifies the viscosity coefficients for the damage model for surface-based cohesive behavior or enriched cohesive behavior.

Access

```
import material
mdb.models[name].materials[name].ductileDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].fldDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].flsdDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].hashinDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].maxeDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].maxpeDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].maxpsDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].maxsDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].mkDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].msfldDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].quadeDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].quadsDamageInitiation\
.damageStabilizationCohesive
mdb.models[name].materials[name].shearDamageInitiation\
.damageStabilizationCohesive
import odbMaterial
session.odbs[name].materials[name].ductileDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].fldDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].flsdDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].hashinDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].maxeDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].maxpeDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].maxpsDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].maxsDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].mkDamageInitiation\
.damageStabilizationCohesive
```

```
session.odbs[name].materials[name].msfldDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].quadeDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].quadsDamageInitiation\
.damageStabilizationCohesive
session.odbs[name].materials[name].shearDamageInitiation\
.damageStabilizationCohesive
```

DamageStabilizationCohesive(...)

This method creates a DamageStabilizationCohesive object.

Path

```
mdb.models[name].materials[name].ductileDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].fldDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].flsdDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].hashinDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].johnsonCookDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].maxeDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].maxpeDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].maxpsDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].maxsDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].mkDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].msfldDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].quadeDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].quadsDamageInitiation\
.DamageStabilizationCohesive
mdb.models[name].materials[name].shearDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].ductileDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].fldDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].flsdDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].hashinDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].johnsonCookDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].maxeDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].maxpeDamageInitiation\
.DamageStabilizationCohesive
```

```
session.odbs[name].materials[name].maxpsDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].maxsDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].mkDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].msfldDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].quadeDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].quadsDamageInitiation\
.DamageStabilizationCohesive
session.odbs[name].materials[name].shearDamageInitiation\
.DamageStabilizationCohesive
```

Required arguments

None.

Optional arguments

cohesiveCoeff `None` or a Float specifying the viscosity coefficient. The default value is `None`.

Return value

A DamageStabilizationCohesive object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DamageStabilizationCohesive object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DamageStabilizationCohesive* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DamageStabilizationCohesive object has members with the same names and descriptions as the arguments to the *DamageStabilizationCohesive* method.

Corresponding analysis keywords

- **DAMAGE STABILIZATION*

Damping object

The Damping object specifies material damping.

Access

```
import material
mdb.models[name].materials[name].damping
import odbMaterial
session.odbs[name].materials[name].damping
```

Damping(...)

This method creates a Damping object.

Path

```
mdb.models[name].materials[name].Damping
session.odbs[name].materials[name].Damping
```

Required arguments

None.

Optional arguments

alpha

A Float specifying the α_R factor to create mass proportional damping in direct-integration and explicit dynamics. The default value is 0.0.

beta

A Float specifying the β_R factor to create stiffness proportional damping in direct-integration and explicit dynamics. The default value is 0.0.

composite

A Float specifying the fraction of critical damping to be used with this material in calculating composite damping factors for the modes (for use in modal dynamics). The default value is 0.0.

This argument applies only to Abaqus/Standard analyses.

structural

A Float specifying the structural factor to create material damping in direct-integration and explicit dynamics. The default value is 0.0.

Return value

A Damping object.

Exceptions

RangeError.

setValues(...)

This method modifies the Damping object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Damping* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Damping object has members with the same names and descriptions as the arguments to the *Damping* method.

Corresponding analysis keywords

- **DAMPING*

DeformationPlasticity object

The DeformationPlasticity object specifies the deformation plasticity model.

Access

```
import material
mdb.models[name].materials[name].deformationPlasticity
import odbMaterial
session.odbs[name].materials[name].deformationPlasticity
```

DeformationPlasticity(...)

This method creates a DeformationPlasticity object.

Path

```
mdb.models[name].materials[name].DeformationPlasticity
session.odbs[name].materials[name].DeformationPlasticity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

Table data

- Young's modulus, E.
- Poisson's ratio, ν.
- Yield stress, σ^0.
- Exponent, n.
- Yield offset, α.
- Temperature, if the data depend on temperature.

Return value

A DeformationPlasticity object.

Exceptions

RangeError.

setValues(...)

This method modifies the DeformationPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DeformationPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DeformationPlasticity object has members with the same names and descriptions as the arguments to the *DeformationPlasticity* method.

Corresponding analysis keywords

- **DEFORMATION PLASTICITY*

Density object

The Density object specifies the material density.

Access

```
import material
mdb.models[name].materials[name].density
import odbMaterial
session.odbs[name].materials[name].density
```

Density(...)

This method creates a Density object.

Path

```
mdb.models[name].materials[name].Density
session.odbs[name].materials[name].Density
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

distributionType A SymbolicConstant specifying how the density is distributed spatially. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

fieldName A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this material option. The *fieldName* argument applies only when *distributionType*=ANALYTICAL_FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

Table data

- The mass density.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Density object.

Exceptions

RangeError.

setValues(...)

This method modifies the Density object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Density* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Density object has members with the same names and descriptions as the arguments to the *Density* method.

Corresponding analysis keywords

- **DENSITY*

Depvar object

The Depvar object specifies solution-dependent state variables.

Access

```
import material
mdb.models[name].materials[name].depvar
import odbMaterial
session.odbs[name].materials[name].depvar
```

Depvar(...)

This method creates a Depvar object.

Path

```
mdb.models[name].materials[name].Depvar
session.odbs[name].materials[name].Depvar
```

Required arguments

None.

Optional arguments

deleteVar An Int specifying the state variable number controlling the element deletion flag. The default value is 0.

This argument applies only to Abaqus/Explicit analyses.

n An Int specifying the number of solution-dependent state variables required at each integration point. The default value is 0.

Return value

A Depvar object.

Exceptions

RangeError.

setValues(...)

This method modifies the Depvar object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Depvar* method.

Return value

```
None.
```

Exceptions

RangeError.

Members

The Depvar object has members with the same names and descriptions as the arguments to the *Depvar* method.

Corresponding analysis keywords

- **DEPVAR*

DetonationPoint object

A DetonationPoint object specifies a suboption of the Eos object. The DetonationPoint object defines either isotropic linear elastic shear or linear viscous shear behavior for a hydrodynamic material.

Access

```
import material
mdb.models[name].materials[name].eos.detonationPoint
import odbMaterial
session.odbs[name].materials[name].eos.detonationPoint
```

DetonationPoint(...)

This method creates a DetonationPoint object.

Path

```
mdb.models[name].materials[name].eos.DetonationPoint
session.odbs[name].materials[name].eos.DetonationPoint
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- X value for coordinate of detonation point.
- Y value for coordinate of detonation point.
- Z value for coordinate of detonation point.
- Detonation delay time.

Return value

A DetonationPoint object.

Exceptions

None.

setValues(...)

This method modifies the DetonationPoint object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DetonationPoint* method.

Return value

`None.`

Exceptions

None.

Members

The DetonationPoint object has members with the same names and descriptions as the arguments to the *DetonationPoint* method.

Corresponding analysis keywords

- **DETONATION POINT*

Dielectric object

The Dielectric object specifies dielectric material properties.

Access

```
import material
mdb.models[name].materials[name].dielectric
import odbMaterial
session.odbs[name].materials[name].dielectric
```

Dielectric(...)

This method creates a Dielectric object.

Path

```
mdb.models[name].materials[name].Dielectric
session.odbs[name].materials[name].Dielectric
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the dielectric behavior. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.

frequencyDependency A Boolean specifying whether the data depend on frequency. The default value is OFF.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Dielectric constant.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{11}^{φ}.
- D_{22}^{φ}.
- D_{33}^{φ}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{11}^{φ}.
- D_{12}^{φ}.
- D_{22}^{φ}.
- D_{13}^{φ}.
- D_{23}^{φ}.
- D_{33}^{φ}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Dielectric object.

Exceptions

None.

`setValues(...)`

This method modifies the Dielectric object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Dielectric* method.

Return value

`None.`

Exceptions

None.

Members

The Dielectric object has members with the same names and descriptions as the arguments to the *Dielectric* method.

Corresponding analysis keywords

- **DIELECTRIC*

Diffusivity object

The Diffusivity object specifies mass diffusivity.

Access

```
import material
mdb.models[name].materials[name].diffusivity
import odbMaterial
session.odbs[name].materials[name].diffusivity
```

Diffusivity(...)

This method creates a Diffusivity object.

Path

```
mdb.models[name].materials[name].Diffusivity
session.odbs[name].materials[name].Diffusivity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of diffusivity. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.

law A SymbolicConstant specifying the diffusion behavior. Possible values are GENERAL and FICK. The default value is GENERAL.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Diffusivity, D.
- Concentration, c.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{11}.
- D_{22}.
- D_{33}.
- Concentration, c.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{11}.
- D_{12}.
- D_{22}.
- D_{13}.
- D_{23}.
- D_{33}.
- Concentration, c.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Diffusivity object.

Exceptions

RangeError.

setValues(...)

This method modifies the Diffusivity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Diffusivity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Diffusivity object has members with the same names and descriptions as the arguments to the *Diffusivity* method. In addition, the Diffusivity object can have the following members:

pressureEffect
A *PressureEffect* object.

soretEffect
A *SoretEffect* object.

Corresponding analysis keywords

- **DIFFUSIVITY*

DruckerPrager object

The DruckerPrager object specifies the extended Drucker-Prager plasticity model.

Access

```
import material
mdb.models[name].materials[name].druckerPrager
import odbMaterial
session.odbs[name].materials[name].druckerPrager
```

DruckerPrager(...)

This method creates a DruckerPrager object.

Path

```
mdb.models[name].materials[name].DruckerPrager
session.odbs[name].materials[name].DruckerPrager
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

shearCriterion A SymbolicConstant specifying the yield criterion. Possible values are LINEAR, HYPERBOLIC, and EXPONENTIAL. The default value is LINEAR.

This argument applies only to Abaqus/Standard analyses. Only the linear Drucker-Prager model is available in Abaqus/Explicit analyses.

eccentricity A Float specifying the flow potential eccentricity, ϵ, a small positive number that defines the rate at which the hyperbolic flow potential approaches its asymptote. The default value is 0.1.

This argument applies only to Abaqus/Standard analyses.

testData A Boolean specifying whether the material constants for the exponent model are to be computed by Abaqus/Standard from triaxial test data at different levels of confining pressure. The default value is OFF.

This argument is valid only if *shearCriterion*=EXPONENTIAL.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *shearCriterion*=LINEAR (the only option allowed in an Abaqus/Explicit analysis), the table data specify the following:

- Material angle of friction, β, in the p–t plane. Give the value in degrees.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \leq K \leq 1.0$. If the default value of 0.0 is accepted, a value of 1.0 is assumed.
- Dilation angle, ψ, in the p–t plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shearCriterion*=HYPERBOLIC, the table data specify the following:

- Material angle of friction, β, at high confining pressure in the p–q plane. Give the value in degrees.
- Initial hydrostatic tension strength, $p_t|_0$.
- Dilation angle, ψ, at high confining pressure in the p–q plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shearCriterion*=EXPONENTIAL, the table data specify the following:

- Dilation angle, ψ, at high confining pressure in the p–q plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DruckerPrager object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DruckerPrager object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DruckerPrager* method.

Return value
None.

Exceptions
RangeError.

Members
The DruckerPrager object has members with the same names and descriptions as the arguments to the *DruckerPrager* method.

In addition, the DruckerPrager object can have the following members:

druckerPragerCreep	A *DruckerPragerCreep* object.
druckerPragerHardening	A *DruckerPragerHardening* object.
rateDependent	A *RateDependent* object.
triaxialTestData	A *TriaxialTestData* object.

Corresponding analysis keywords
- **DRUCKER PRAGER*

DruckerPragerCreep object

The DruckerPragerCreep object specifies creep for Drucker-Prager plasticity models.

Access

```
import material
mdb.models[name].materials[name].druckerPrager.druckerPragerCreep
import odbMaterial
session.odbs[name].materials[name].druckerPrager.druckerPragerCreep
```

DruckerPragerCreep(...)

This method creates a DruckerPragerCreep object.

Path

```
mdb.models[name].materials[name].druckerPrager.DruckerPragerCreep
session.odbs[name].materials[name].druckerPrager.DruckerPragerCreep
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

law
A SymbolicConstant specifying the type of data defining the creep law. Possible values are:

- STRAIN, specifying a strain-hardening power law.
- TIME, specifying a time-hardening power law.
- SINGHM, specifying a Singh-Mitchell type law.
- USER, specifying the creep law is input from user subroutine *CREEP*.

The default value is STRAIN.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=TIME or *law*=STRAIN, the table data specify the following:

- A. (Units of $F^{-n}L^{2n}T^{-1-m}$.)
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A. (Units of T^{-1}.)
- α. (Units of $F^{-1}L^2$.)
- m.
- t_1. (Units of T.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DruckerPragerCreep object.

Exceptions

RangeError.

setValues(...)

This method modifies the DruckerPragerCreep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DruckerPragerCreep* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DruckerPragerCreep object has members with the same names and descriptions as the arguments to the *DruckerPragerCreep* method.

Corresponding analysis keywords

- **DRUCKER PRAGER CREEP*

DruckerPragerHardening object

The DruckerPragerHardening object specifies hardening for Drucker-Prager plasticity models.

Access

```
import material
mdb.models[name].materials[name].druckerPrager.druckerPragerHardening
import odbMaterial
session.odbs[name].materials[name].druckerPrager\
.druckerPragerHardening
```

DruckerPragerHardening(...)

This method creates a DruckerPragerHardening object.

Path

```
mdb.models[name].materials[name].druckerPrager.DruckerPragerHardening
session.odbs[name].materials[name].druckerPrager\
.DruckerPragerHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of data defining the hardening behavior. Possible values are COMPRESSION, TENSION, and SHEAR. The default value is COMPRESSION.

rate A Boolean specifying whether the data depend on rate. The default value is OFF.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress.
- Absolute value of the corresponding plastic strain. (The first tabular value entered must always be zero.)
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$, for which this hardening curve applies.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DruckerPragerHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the DruckerPragerHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DruckerPragerHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The DruckerPragerHardening object has members with the same names and descriptions as the arguments to the *DruckerPragerHardening* method.

Corresponding analysis keywords

- **DRUCKER PRAGER HARDENING*

Elastic object

The Elastic object specifies elastic material properties.

Access

```
import material
mdb.models[name].materials[name].elastic
import odbMaterial
session.odbs[name].materials[name].elastic
```

Elastic(...)

This method creates an Elastic object.

Path

```
mdb.models[name].materials[name].Elastic
session.odbs[name].materials[name].Elastic
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

type
A SymbolicConstant specifying the type of elasticity data provided. Possible values are:

- ISOTROPIC
- ORTHOTROPIC
- ANISOTROPIC
- ENGINEERING_CONSTANTS
- LAMINA
- TRACTION
- COUPLED_TRACTION
- SHORT_FIBER
- SHEAR
- BILAMINA

The default value is ISOTROPIC.

noCompression
A Boolean specifying whether compressive stress is allowed. The default value is OFF.

noTension
A Boolean specifying whether tensile stress is allowed. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

moduli

A SymbolicConstant specifying the time-dependence of the elastic material constants. Possible values are INSTANTANEOUS and LONG_TERM. The default value is LONG_TERM.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- The Young's modulus, E.
- The Poisson's ratio, ν.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=SHEAR, the table data specify the following:

- The shear modulus,G.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENGINEERING_CONSTANTS, the table data specify the following:

- E_1.
- E_2.
- E_3.
- ν_{12}.
- ν_{13}.
- ν_{23}.
- G_{12}.
- G_{13}.
- G_{23}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=LAMINA, the table data specify the following:

- E_1.
- E_2.

- ν_{12}.
- G_{12}.
- G_{13}. This shear modulus is needed to define transverse shear behavior in shells.
- G_{23}. This shear modulus is needed to define transverse shear behavior in shells.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{1111}.
- D_{1122}.
- D_{2222}.
- D_{1133}.
- D_{2233}.
- D_{3333}.
- D_{1212}.
- D_{1313}.
- D_{2323}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{1111}.
- D_{1122}.
- D_{2222}.
- D_{1133}.
- D_{2233}.
- D_{3333}.
- D_{1112}.
- D_{2212}.
- D_{3312}.
- D_{1212}.

- D_{1113}.
- D_{2213}.
- D_{3313}.
- D_{1213}.
- D_{1313}.
- D_{1123}.
- D_{2223}.
- D_{3323}.
- D_{1223}.
- D_{1323}.
- D_{2323}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=TRACTION, the table data specify the following:

- E for warping elements; E_{nn} for cohesive elements.
- G_1 for warping elements; E_{ss} for cohesive elements.
- G_2 for warping elements; E_{tt} for cohesive elements.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=BILAMINA, the table data specify the following:

- E_{1+}.
- E_{2+}.
- ν_{12+}.
- G_{12}.
- E_{1-}.
- E_{2-}.
- ν_{12-}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *type*=SHORT_FIBER, there is no table data.

Return value

An Elastic object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Elastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Elastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Elastic object has members with the same names and descriptions as the arguments to the *Elastic* method.

In addition, the Elastic object can have the following members:

failStress A *FailStress* object.

failStrain A *FailStrain* object.

Corresponding analysis keywords

- **ELASTIC*

ElectricalConductivity object

The ElectricalConductivity object specifies electrical conductivity.

Access

```
import material
mdb.models[name].materials[name].electricalConductivity
import odbMaterial
session.odbs[name].materials[name].electricalConductivity
```

ElectricalConductivity(...)

This method creates an ElectricalConductivity object.

Path

```
mdb.models[name].materials[name].ElectricalConductivity
session.odbs[name].materials[name].ElectricalConductivity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of electrical conductivity. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.

frequencyDependency A Boolean specifying whether the data depend on frequency. The default value is OFF.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Electrical conductivity.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- σ^E_{11}.
- σ^E_{22}.
- σ^E_{33}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- σ^E_{11}.
- σ^E_{12}.
- σ^E_{22}.
- σ^E_{13}.
- σ^E_{23}.
- σ^E_{33}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

An ElectricalConductivity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ElectricalConductivity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ElectricalConductivity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ElectricalConductivity object has members with the same names and descriptions as the arguments to the *ElectricalConductivity* method.

Corresponding analysis keywords

- **ELECTRICAL CONDUCTIVITY*

Eos object

The Eos object specifies an equation of state model.

Access

```
import material
mdb.models[name].materials[name].eos
import odbMaterial
session.odbs[name].materials[name].eos
```

Eos(...)

This method creates an Eos object.

Path

```
mdb.models[name].materials[name].Eos
session.odbs[name].materials[name].Eos
```

Required arguments

None.

Optional arguments

type

A SymbolicConstant specifying the equation of state. Possible values are USUP, JWL, IDEALGAS, TABULAR, and IGNITIONANDGROWTH. The default value is IDEALGAS.

temperatureDependency

A Boolean specifying whether the data in *gasSpecificTable* depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variable dependencies for the data in *gasSpecificTable*. The default value is 0.

detonationEnergy

A Float specifying the detonation energy text field. The default value is 0.0.

solidTable

A sequence of sequences of Floats specifying the following:

- A_{s}.
- B_{s}.
- ${\omega}_{s}$.
- R_{1s}.
- R_{2s}.

The default value is an empty sequence.

gasTable

A sequence of sequences of Floats specifying the following:

- A_{g}.
- B_{g}.

- ${\omega}_{g}$.
- R_{1g}.
- R_{2g}.

The default value is an empty sequence.

reactionTable

A sequence of sequences of Floats specifying the following:

- Initial Pressure, I.
- Product co-volume, a.
- Exponent on the unreacted fraction (ignition term), x.
- First burn rate coefficient, G_{1}
- Exponent on the unreacted fraction (growth term), c.
- Exponent on the reacted fraction (growth term), d.
- Pressure exponent (growth term), y.
- Second burn rate coefficient, G_{2}.
- Exponent on the unreacted fraction (completion term), e.
- Exponent on the reacted fraction (completion term), g.
- Pressure exponent (completion term), z.
- Initial reacted fraction, ${F^{max}}_{ig}$.
- Maximum reacted fraction for the growth term, ${F^{max}}_{G1}$.
- Minimum reacted fraction, ${F^{min}}_{G2}$.

The default value is an empty sequence.

gasSpecificTable

A sequence of sequences of Floats specifying the following:

- Specific Heat per unit mass.
- Temperature dependent data.
- Value of first field variable.
- Value of second field variable.
- Etc.

The default value is an empty sequence.

table

A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

Table data

If *type*=IDEALGAS, the table data represents the following:

- Gas constant, R.
- The ambient pressure, p_A. If this field is left blank, a default of 0.0 is used.

If *type*=JWL, the table data represents the following:

- Detonation wave speed, C_d.
- A.
- B.
- ω. (Dimensionless.)
- R_1. (Dimensionless.)
- R_2. (Dimensionless.)
- Pre-detonation bulk modulus, K_{pd}.
- Detonation energy density, E_0.

If *type*=USUP, the table data represents the following:

- c_0.
- s. (Dimensionless.)
- Γ_0. (Dimensionless.)

If *type*=TABULAR, the table data represents the following:

- F_1.
- F_2.
- ε_{vol}^c. (Dimensionless.)

Return value

An Eos object.

Exceptions

None.

`setValues(...)`

This method modifies the Eos object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Eos* method.

Return value

`None.`

Exceptions

None.

Members

The Eos object has members with the same names and descriptions as the arguments to the *Eos* method.

In addition, the Eos object can have the following members:

detonationPoint A *DetonationPoint* object.

eosCompaction An *EosCompaction* object.

tensileFailure A *TensileFailure* object.

Corresponding analysis keywords

- **EOS*

EosCompaction object

The EosCompaction object specifies material eos compaction.

Access

```
import material
mdb.models[name].materials[name].eos.eosCompaction
import odbMaterial
session.odbs[name].materials[name].eos.eosCompaction
```

EosCompaction(...)

This method creates an EosCompaction object.

Path

```
mdb.models[name].materials[name].eos.EosCompaction
session.odbs[name].materials[name].eos.EosCompaction
```

Required arguments

soundSpeed	A Float specifying reference sound speed in the porous material.
porosity	A Float specifying value of the porosity of the unloaded material.
pressure	A Float specifying pressure required to initialize plastic behavior.
compactionPressure	A Float specifying compaction pressure at which all pores are crushed.

Optional arguments

None.

Return value

An EosCompaction object.

Exceptions

RangeError.

setValues(...)

This method modifies the EosCompaction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EosCompaction* method.

Return value

`None.`

Exceptions

RangeError.

Members

The EosCompaction object has members with the same names and descriptions as the arguments to the *EosCompaction* method.

Corresponding analysis keywords

- **EOS COMPACTION*

Expansion object

The Expansion object specifies thermal expansion.

Access

```
import material
mdb.models[name].materials[name].expansion
import odbMaterial
session.odbs[name].materials[name].expansion
```

Expansion(...)

This method creates an Expansion object.

Path

```
mdb.models[name].materials[name].Expansion
session.odbs[name].materials[name].Expansion
```

Required arguments

None.

Optional arguments

type
A SymbolicConstant specifying the type of expansion. Possible values are ISOTROPIC, ORTHOTROPIC, ANISOTROPIC, and SHORT_FIBER. The default value is ISOTROPIC.

userSubroutine
A Boolean specifying whether a user subroutine is used to define the increments of thermal strain. The default value is OFF.

zero
A Float specifying the value of θ^0 if the thermal expansion is temperature-dependent or field-variable-dependent. The default value is 0.0.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

table
A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

This argument is required only if *type* is not USER.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- α in Abaqus/Standard or Abaqus/Explicit analysis.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- α_{11}.
- α_{22}.
- α_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- α_{11}.
- α_{22}.
- α_{33}. (Not used for plane stress case.)
- α_{12}.
- α_{13}.
- α_{23}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=SHORT_FIBER, there is no table data.

Return value

An Expansion object.

Exceptions

RangeError.

setValues(...)

This method modifies the Expansion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Expansion* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Expansion object has members with the same names and descriptions as the arguments to the *Expansion* method.

Corresponding analysis keywords

- **EXPANSION*

FailStrain object

The FailStrain object defines parameters for strain-based failure measures.

Access

```
import material
mdb.models[name].materials[name].elastic.failStrain
import odbMaterial
session.odbs[name].materials[name].elastic.failStrain
```

FailStrain(...)

This method creates a FailStrain object.

Path

```
mdb.models[name].materials[name].elastic.FailStrain
session.odbs[name].materials[name].elastic.FailStrain
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tensile strain limit in fiber direction, $X_{\varepsilon t}$.
- Compressive strain limit in fiber direction, $X_{\varepsilon c}$.
- Tensile strain limit in transverse direction, $Y_{\varepsilon t}$.
- Compressive strain limit in transverse direction, $Y_{\varepsilon c}$.
- Shear strain limit in the X–Y plane, S_{ε}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FailStrain object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the FailStrain object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FailStrain* method.

Return value

`None.`

Exceptions

RangeError.

Members

The FailStrain object has members with the same names and descriptions as the arguments to the *FailStrain* method.

Corresponding analysis keywords

- **FAIL STRAIN*

FailStress object

The FailStress object defines parameters for stress-based failure measures.

Access

```
import material
mdb.models[name].materials[name].elastic.failStress
import odbMaterial
session.odbs[name].materials[name].elastic.failStress
```

FailStress(...)

This method creates a FailStress object.

Path

```
mdb.models[name].materials[name].elastic.FailStress
session.odbs[name].materials[name].elastic.FailStress
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tensile stress limit in fiber direction, X_t.
- Compressive stress limit in fiber direction, X_c.
- Tensile stress limit in transverse direction, Y_t.
- Compressive stress limit in transverse direction, Y_c.
- Shear strength in the X–Y plane, S.
- Cross product term coefficient, $\overset{*}{f}$ ($-1.0 \leq \overset{*}{f} \leq 1.0$). The default value is zero.
- Biaxial stress limit, σ_{biax}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FailStress object.

Exceptions

RangeError.

setValues(...)

This method modifies the FailStress object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FailStress* method.

Return value

`None.`

Exceptions

RangeError.

Members

The FailStress object has members with the same names and descriptions as the arguments to the *FailStress* method.

Corresponding analysis keywords

- **FAIL STRESS*

FailureRatios object

The FailureRatios object specifies the shape of the failure surface for a *Concrete* model.

Access

```
import material
mdb.models[name].materials[name].concrete.failureRatios
import odbMaterial
session.odbs[name].materials[name].concrete.failureRatios
```

FailureRatios(...)

This method creates a FailureRatios object.

Path

```
mdb.models[name].materials[name].concrete.FailureRatios
session.odbs[name].materials[name].concrete.FailureRatios
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Ratio of the ultimate biaxial compressive stress to the uniaxial compressive ultimate stress. The default value is 1.16.
- Absolute value of the ratio of the uniaxial tensile stress at failure to the uniaxial compressive stress at failure. The default value is 0.09.
- Ratio of the magnitude of a principal component of plastic strain at ultimate stress in biaxial compression to the plastic strain at ultimate stress in uniaxial compression. The default value is 1.28.
- Ratio of the tensile principal stress value at shear in plane stress, when the other nonzero principal stress component is at the ultimate compressive stress value, to the tensile cracking stress under uniaxial tension. The default value is 1/3.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FailureRatios object.

Exceptions

RangeError.

setValues(...)

This method modifies the FailureRatios object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FailureRatios* method.

Return value

`None.`

Exceptions

RangeError.

Members

The FailureRatios object has members with the same names and descriptions as the arguments to the *FailureRatios* method.

Corresponding analysis keywords

- **FAILURE RATIOS*

FluidLeakoff object

The FluidLeakoff object specifies leak-off coefficients for pore pressure cohesive elements.

Access

```
import material
mdb.models[name].materials[name].fluidLeakoff
import odbMaterial
session.odbs[name].materials[name].fluidLeakoff
```

FluidLeakoff(...)

This method creates a FluidLeakoff object.

Path

```
mdb.models[name].materials[name].FluidLeakoff
session.odbs[name].materials[name].FluidLeakoff
```

Required arguments

None.

Optional arguments

temperatureDependency	A Boolean specifying whether the data depend on temperature. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.
type	A SymbolicConstant specifying the type of fluid leak-off. Possible values are COEFFICIENTS and USER. The default value is COEFFICIENTS.
table	A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

Table data

The table data specify the following:

- Fluid leak-off coefficient at top element surface.
- Fluid leak-off coefficient at bottom element surface.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FluidLeakoff object.

Exceptions

None.

setValues(...)

This method modifies the FluidLeakoff object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidLeakoff* method.

Return value

`None.`

Exceptions

None.

Members

The FluidLeakoff object has members with the same names and descriptions as the arguments to the *FluidLeakoff* method.

Corresponding analysis keywords

- **FLUID LEAKOFF*

GapConductance object

The GapConductance object specifies conductive heat transfer between closely adjacent (or contacting) surfaces.

Access

```
import material
mdb.models[name].materials[name].gapConductance
import odbMaterial
session.odbs[name].materials[name].gapConductance
```

GapConductance(...)

This method creates a GapConductance object.

Path

```
mdb.models[name].materials[name].GapConductance
session.odbs[name].materials[name].GapConductance
```

Required arguments

Optional arguments

pressureDependency	A Boolean specifying whether the data depend on pressure. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.
table	A sequence of sequences of Floats specifying the items described below.

Table data

- Gap Conductance or Cohesive Separation.
- Gap Clearance, Gap Pressure (if optional parameter pressureDependency is used), or Closure, c (for coupled temperature-displacement gasket elements).
- Average Temperature if the data depend on temperature.
- Mass Flow Rate per unit area if the data depend on the average mass flow rate.
- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapConductance object.

`setValues(...)`

This method modifies the GapConductance object.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapConductance* method.

Members

The GapConductance object has members with the same names and descriptions as the arguments to the *GapConductance* method.

Corresponding analysis keywords

- *GAP CONDUCTANCE*

GapConvection object

The GapConvection object specifies the Nusselt number (**Nu**) to calculate the convective coefficient for heat transfer between the gap flow and both the top and bottom surfaces of a coupled temperature-pore pressure cohesive element.

Access

```
import material
mdb.models[name].materials[name].gapConvection
import odbMaterial
session.odbs[name].materials[name].gapConvection
```

GapConvection(...)

This method creates a GapConvection object.

Path

```
mdb.models[name].materials[name].GapConvection
session.odbs[name].materials[name].GapConvection
```

Required arguments

type An odb_String specifying the type of gap convection. Possible values are FLUX, TEMPERATURE, and TABULAR. The default value is FLUX.

Optional arguments

If *type*=TABULAR, the following optional arguments can be used:

table A sequence of sequences of Floats specifying the items described below.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For *type*=TABULAR the table data specify the following:

- Nusselt number (**Nu**)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapConvection object.

Exceptions

None.

setValues(...)

This method modifies the GapConvection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapConvection* method.

Return value

`None.`

Members

The GapConvection object has members with the same names and descriptions as the arguments to the *GapConvection* method.

Corresponding analysis keywords

- **GAP CONVECTION*

GapFlow object

The GapFlow object specifies tangential flow constitutive parameters for pore pressure cohesive elements.

Access

```
import material
mdb.models[name].materials[name].gapFlow
import odbMaterial
session.odbs[name].materials[name].gapFlow
```

GapFlow(...)

This method creates a GapFlow object.

Path

```
mdb.models[name].materials[name].GapFlow
session.odbs[name].materials[name].GapFlow
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

kmax `None` or a Float specifying the maximum permeability value that should be used. If *kmax*=`None`, Abaqus assumes that the permeability is not bounded. This value is meaningful only when *type*=NEWTONIAN. The default value is `None`.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

type A SymbolicConstant specifying the type of gap flow. Possible values are NEWTONIAN, POWER_LAW, BINGHAM_PLASTIC, and HERSCHEL-BULKLEY. The default value is NEWTONIAN.

Table data

If *type*=NEWTONIAN the table data specify the following:

- Pore viscosity.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=POWER_LAW the table data specify the following:

- Consistency.
- Exponent.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=BINGHAM_PLASTIC the table data specify the following:

- Consistency.
- Yield stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=HERSCHEL-BULKLEY the table data specify the following:

- Consistency.
- Exponent.
- Yield stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapFlow object.

Exceptions

None.

`setValues(...)`

This method modifies the GapFlow object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapFlow* method.

Return value

`None.`

Exceptions

None.

Members

The GapFlow object has members with the same names and descriptions as the arguments to the *GapFlow* method.

Corresponding analysis keywords

- **GAP FLOW*

GapRadiation object

The GapRadiation object specifies radiative heat transfer between closely adjacent surfaces.

Access

```
import material
mdb.models[name].materials[name].gapRadiation
import odbMaterial
session.odbs[name].materials[name].gapRadiation
```

GapRadiation(...)

This method creates a GapRadiation object.

Path

```
mdb.models[name].materials[name].Gapradiation
session.odbs[name].materials[name].Gapradiation
```

Required arguments

mainSurfaceEmissivity	A Float specifying the Emissivity of main surface.ϵ_A
secondarySurfaceEmissivity	A Float specifying the Emissivity of the secondary surfaceϵ_B.
table	A sequence of sequences of Floats specifying the items described below.

Optional arguments

Table data

- Effective view factor.
- Gap clearance.
- Repeat this data line as often as necessary to define the dependence of the view factor on gap clearance.

Return value

A Gapradiation object.

Exceptions

None.

setValues(...)

This method modifies the GapRadiation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GapRadiation* method.

Return value

`None.`

Exceptions

None.

Members

The Gap Radiation object has members with the same names and descriptions as the arguments to the *GapRadiation* method.

Corresponding analysis keywords

- **GAP RADIATION*

GasketMembraneElastic object

The GasketMembraneElastic object defines the elastic parameters for the membrane shear behavior of a gasket.

Access

```
import material
mdb.models[name].materials[name].gasketMembraneElastic
import odbMaterial
session.odbs[name].materials[name].gasketMembraneElastic
```

GasketMembraneElastic(...)

This method creates a GasketMembraneElastic object.

Path

```
mdb.models[name].materials[name].GasketMembraneElastic
session.odbs[name].materials[name].GasketMembraneElastic
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Young's modulus, E.
- Poisson's ratio, ν.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketMembraneElastic object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the GasketMembraneElastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GasketMembraneElastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The GasketMembraneElastic object has members with the same names and descriptions as the arguments to the *GasketMembraneElastic* method.

Corresponding analysis keywords

- **GASKET ELASTICITY*

GasketThicknessBehavior object

The GasketThicknessBehavior object defines the behavior in the thickness direction for a gasket.

Access

```
import material
mdb.models[name].materials[name].gasketThicknessBehavior
import odbMaterial
session.odbs[name].materials[name].gasketThicknessBehavior
```

GasketThicknessBehavior(...)

This method creates a GasketThicknessBehavior object.

Path

```
mdb.models[name].materials[name].GasketThicknessBehavior
session.odbs[name].materials[name].GasketThicknessBehavior
```

Required arguments

table
A sequence of sequences of Floats specifying loading data. The first sequence must contain only 0. At least two sequences must be specified if *type*=DAMAGE, and at least 3 sequences must be specified if *type*=ELASTIC_PLASTIC. The items in the table data are described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the loading data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies included in the definition of the loading data, in addition to temperature. The default value is 0.

tensileStiffnessFactor
A Float specifying the fraction of the initial compressive stiffness that defines the stiffness in tension. The default value is 10^{-3}.

type
A SymbolicConstant specifying a damage elasticity model or an elastic-plastic model for gasket thickness-direction behavior. Possible values are ELASTIC_PLASTIC and DAMAGE. The default value is ELASTIC_PLASTIC.

unloadingDependencies
An Int specifying the number of field variable dependencies included in the definition of the unloading data, in addition to temperature. The default value is 0.

unloadingTemperatureDependency
A Boolean specifying whether unloading data depends on temperature. The default value is OFF.

variableUnits

A SymbolicConstant specifying the behavior in terms of units of force (or force in unit length) versus closure or pressure versus closure. Possible values are STRESS and FORCE. The default value is STRESS.

yieldOnset

A Float specifying the closure value at which the onset of yield occurs or the relative drop in slope on the loading curve that defines the onset of plastic deformation (depending on the value of *yieldOnsetMethod*). The default value is 0.1.

yieldOnsetMethod

A SymbolicConstant specifying the method used to determine yield onset. Possible values are RELATIVE_SLOPE_DROP and CLOSURE_VALUE. The default value is RELATIVE_SLOPE_DROP.

unloadingTable

A sequence of sequences of Floats specifying unloading data. The items in the table data are described below. The default value is an empty sequence.

Table data

If *variableUnits*=STRESS, the loading table data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE, the loading table data specify the following:

- Force or force per unit length; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=STRESS and *type*=ELASTIC_PLASTIC, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Plastic closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE and *type*=ELASTIC_PLASTIC, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Plastic closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=STRESS and *type*=DAMAGE, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Maximum closure reached while loading the gasket; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE and *type*=DAMAGE, the *unloadingTable* data specify the following:

- Force or force per unit length; this value must be positive.
- Closure; this value must be positive.
- Maximum closure reached while loading the gasket; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketThicknessBehavior object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the GasketThicknessBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GasketThicknessBehavior* method.

Return value

```
None.
```

Exceptions

RangeError.

Members

The GasketThicknessBehavior object has members with the same names and descriptions as the arguments to the *GasketThicknessBehavior* method.

In addition, the GasketThicknessBehavior object can have the following member:

contactArea
A *ContactArea* object.

Corresponding analysis keywords

- **GASKET THICKNESS BEHAVIOR*

GasketTransverseShearElastic object

The GasketTransverseShearElastic object defines the elastic parameters for the transverse shear behavior of a gasket.

Access

```
import material
mdb.models[name].materials[name].gasketTransverseShearElastic
import odbMaterial
session.odbs[name].materials[name].gasketTransverseShearElastic
```

GasketTransverseShearElastic(...)

This method creates a GasketTransverseShearElastic object.

Path

```
mdb.models[name].materials[name].GasketTransverseShearElastic
session.odbs[name].materials[name].GasketTransverseShearElastic
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

variableUnits
A SymbolicConstant specifying the unit system in which the transverse shear behavior will be defined. Possible values are STRESS and FORCE. The default value is STRESS.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Shear stiffness. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketTransverseShearElastic object.

Exceptions

RangeError.

setValues(...)

This method modifies the GasketTransverseShearElastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GasketTransverseShearElastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The GasketTransverseShearElastic object has members with the same names and descriptions as the arguments to the *GasketTransverseShearElastic* method.

Corresponding analysis keywords

- **GASKET ELASTICITY*

Gel object

The Gel object defines a swelling gel.

Access

```
import material
mdb.models[name].materials[name].gel
import odbMaterial
session.odbs[name].materials[name].gel
```

Gel(...)

This method creates a Gel object.

Path

```
mdb.models[name].materials[name].Gel
session.odbs[name].materials[name].Gel
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- Radius of gel particles when completely dry, r_a^{dry}.
- Fully swollen radius of gel particles, r_a^f.
- Number of gel particles per unit volume, k_a.
- Relaxation time constant for long-term swelling of gel particles, τ_1.

Return value

A Gel object.

Exceptions

None.

setValues(...)

This method modifies the Gel object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Gel* method.

Return value

`None.`

Exceptions

None.

Members

The Gel object has members with the same names and descriptions as the arguments to the *Gel* method.

Corresponding analysis keywords

- **GEL*

HeatGeneration object

The HeatGeneration object includes volumetric heat generation in heat transfer analyses.

Access

```
import material
mdb.models[name].materials[name].heatGeneration
import odbMaterial
session.odbs[name].materials[name].heatGeneration
```

HeatGeneration()

This method creates a HeatGeneration object.

Path

```
mdb.models[name].materials[name].HeatGeneration
session.odbs[name].materials[name].HeatGeneration
```

Arguments

None.

Return value

A HeatGeneration object.

Exceptions

None.

Members

The HeatGeneration object has no members.

Corresponding analysis keywords

- **HEAT GENERATION*

Hyperelastic object

The Hyperelastic object specifies elastic properties for approximately incompressible elastomers.

Access

```
import material
mdb.models[name].materials[name].hyperelastic
import odbMaterial
session.odbs[name].materials[name].hyperelastic
```

Hyperelastic(...)

This method creates a Hyperelastic object.

Path

```
mdb.models[name].materials[name].Hyperelastic
session.odbs[name].materials[name].Hyperelastic
```

Required arguments

table

A sequence of sequences of Floats specifying the items described below. This argument is valid only if *testData*=OFF.

Optional arguments

type

A SymbolicConstant specifying the type of strain energy potential. Possible values are:

- ARRUDA_BOYCE
- MARLOW
- MOONEY_RIVLIN
- NEO_HOOKE
- OGDEN
- POLYNOMIAL
- REDUCED_POLYNOMIAL
- USER
- VAN_DER_WAALS
- YEOH
- UNKNOWN
- VALANIS_LANDEL

The default value is UNKNOWN.

moduliTimeScale

A SymbolicConstant specifying the nature of the time response. Possible values are INSTANTANEOUS and LONG_TERM. The default value is LONG_TERM.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is OFF.

n

An Int specifying the order of the strain energy potential. The default value is 1.

If *testData*=ON and *type*=POLYNOMIAL, *n* can take only the values 1 or 2.

If *testData*=ON and *type*=OGDEN or if *testData*=OFF for either type, $1 \leq n \leq 6$.

If *type*=USER, this argument cannot be used.

beta

The SymbolicConstant FITTED_VALUE or a Float specifying the invariant mixture parameter. This argument is valid only if *testData*=ON and *type*=VAN_DER_WAALS. The default value is FITTED_VALUE.

testData

A Boolean specifying whether test data are supplied. The default value is ON.

compressible

A Boolean specifying whether the hyperelastic material is compressible. This parameter is applicable only to user-defined hyperelastic materials. The default value is OFF.

properties

An Int specifying the number of property values needed as data for the user-defined hyperelastic material. The default value is 0.

deviatoricResponse

A SymbolicConstant specifying which test data to use. Possible values are UNIAXIAL, BIAXIAL, and PLANAR. The default value is UNIAXIAL.

volumetricResponse

A SymbolicConstant specifying the volumetric response. Possible values are DEFAULT, VOLUMETRIC_DATA, POISSON_RATIO, and LATERAL_NOMINAL_STRAIN. The default value is DEFAULT.

poissonRatio

A Float specifying the Poisson's ratio. This argument is valid only if *volumetricResponse*=POISSON_RATIO. The default value is 0.0.

materialType

A SymbolicConstant specifying the type of material. Possible values are ISOTROPIC and ANISOTROPIC. The default value is ISOTROPIC.

anisotropicType

A SymbolicConstant specifying the type of strain energy potential. Possible values are FUNG_ANISOTROPIC, FUNG_ORTHOTROPIC, HOLZAPFEL, and USER_DEFINED. The default value is FUNG_ANISOTROPIC.

formulation

A SymbolicConstant specifying the type of formulation. Possible values are STRAIN and INVARIANT. The default value is STRAIN.

behaviorType

A SymbolicConstant specifying the type of anisotropic hyperelastic material behavior. Possible values are INCOMPRESSIBLE and COMPRESSIBLE. The default value is INCOMPRESSIBLE.

dependencies

An Int specifying the number of field variable dependencies for the data in*volumetricTable*. The default value is 0.

localDirections

An Int specifying the number of local directions for the data in*volumetricTable*. The default value is 0.

Table data

If *type*=ARRUDA_BOYCE, the table data specify the following:

- μ.
- λ_m.
- D.
- Temperature, if the data depend on temperature.

If *type*=MOONEY_RIVLIN, the table data specify the following:

- C_{10}.
- C_{01}.
- D_1.
- Temperature, if the data depend on temperature.

If *type*=NEO_HOOKE, the table data specify the following:

- C_{10}.
- D_1.
- Temperature, if the data depend on temperature.

If *type*=OGDEN, the table data specify the following for values of n:

- μ_i and α_i for i from 1 to n.
- n coefficients D_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=POLYNOMIAL, the table data specify the following for values of n:

- C_{ij} for each value of $(i + j)$ from 1 to n with i decreasing from $(i + j)$ to zero and j increasing from zero to $(i + j)$.
- n coefficients D_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $3 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=REDUCED_POLYNOMIAL, the table data specify the following for values of n:

- C_{i0} for i from 1 to n.
- n coefficients D_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=VAN_DER_WAALS, the table data specify the following:

- μ.
- λ_m.

- a.
- β.
- D.
- Temperature, if the data depend on temperature.

If *type*=YEOH, the table data specify the following:

- C_{10}.
- C_{20}.
- C_{30}.
- D_1.
- D_2.
- D_3.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed in an Abaqus/Explicit analysis.

The `None` object is the default value if *testData*=ON.

Return value

A Hyperelastic object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the Hyperelastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Hyperelastic* method.

Return value

`None`.

Exceptions

RangeError.

Members

The Hyperelastic object has members with the same names and descriptions as the arguments to the *Hyperelastic* method.

In addition, the Hyperelastic object can have the following members:

biaxialTestData	A *BiaxialTestData* object.
planarTestData	A *PlanarTestData* object.
uniaxialTestData	A *UniaxialTestData* object.
volumetricTestData	A *VolumetricTestData* object.
hysteresis	A *Hysteresis* object.

Corresponding analysis keywords

- **HYPERELASTIC*

Hyperfoam object

The Hyperfoam object specifies elastic properties for a hyperelastic foam.

Access

```
import material
mdb.models[name].materials[name].hyperfoam
import odbMaterial
session.odbs[name].materials[name].hyperfoam
```

Hyperfoam(...)

This method creates a Hyperfoam object.

Path

```
mdb.models[name].materials[name].Hyperfoam
session.odbs[name].materials[name].Hyperfoam
```

Required arguments

None.

Optional arguments

testData A Boolean specifying whether test data are supplied. The default value is OFF.

poisson `None` or a Float specifying the effective Poisson's ratio, ν, of the material. This argument is valid only when *testData*=ON. The default value is `None`.

n An Int specifying the order of the strain energy potential. Possible values are $1 \le n \le 6$. The default value is 1.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

moduli A SymbolicConstant specifying the time-dependence of the material constants. Possible values are INSTANTANEOUS and LONG_TERM. The default value is LONG_TERM.

table A sequence of sequences of Floats specifying the items described below. This argument is valid only when *testData*=OFF. The default value is an empty sequence.

Table data

The items in the table data specify the following for values of n:

- μ_i and α_i for i from 1 to n.
- ν_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \le n \le 6$ in an Abaqus/Explicit analysis.

Return value

A Hyperfoam object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Hyperfoam object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Hyperfoam* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Hyperfoam object has members with the same names and descriptions as the arguments to the *Hyperfoam* method.

In addition, the Hyperfoam object can have the following members:

biaxialTestData	A *BiaxialTestData* object.
volumetricTestData	A *VolumetricTestData* object.
planarTestData	A *PlanarTestData* object.
simpleShearTestData	A *SimpleShearTestData* object.
uniaxialTestData	A *UniaxialTestData* object.

Corresponding analysis keywords

- **HYPERFOAM*

Hypoelastic object

The Hypoelastic object specifies hypoelastic material properties.

Access

```
import material
mdb.models[name].materials[name].hypoelastic
import odbMaterial
session.odbs[name].materials[name].hypoelastic
```

Hypoelastic(...)

This method creates a Hypoelastic object.

Path

```
mdb.models[name].materials[name].Hypoelastic
session.odbs[name].materials[name].Hypoelastic
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

user A Boolean specifying that hypoelasticity is defined by user subroutine *UHYPEL*. The default value is OFF.

Table data

- Instantaneous Young's modulus, E.
- Instantaneous Poisson's ratio, ν.
- First strain invariant, I_1.
- Second strain invariant, I_2.
- Third strain invariant, I_3.

Return value

A Hypoelastic object.

Exceptions

None.

setValues(...)

This method modifies the Hypoelastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Hypoelastic* method.

Return value

`None.`

Exceptions

None.

Members

The Hypoelastic object has members with the same names and descriptions as the arguments to the *Hypoelastic* method.

Corresponding analysis keywords

- **HYPOELASTIC*

Hysteresis object

The Hysteresis object specifies the creep part of the material model for the hysteretic behavior of elastomers.

Access

```
import material
mdb.models[name].materials[name].hyperelastic.hysteresis
import odbMaterial
session.odbs[name].materials[name].hyperelastic.hysteresis
```

Hysteresis(...)

This method creates a Hysteresis object.

Path

```
mdb.models[name].materials[name].hyperelastic.Hysteresis
session.odbs[name].materials[name].hyperelastic.Hysteresis
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- Stress scaling factor.
- Creep parameter.
- Effective stress exponent.
- Creep strain exponent.

Return value

A Hysteresis object.

Exceptions

RangeError.

setValues(...)

This method modifies the Hysteresis object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Hysteresis* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Hysteresis object has members with the same names and descriptions as the arguments to the *Hysteresis* method.

Corresponding analysis keywords

- **HYSTERESIS*

InelasticHeatFraction object

The InelasticHeatFraction object defines the fraction of the rate of inelastic dissipation that appears as a heat source.

Access

```
import material
mdb.models[name].materials[name].inelasticHeatFraction
import odbMaterial
session.odbs[name].materials[name].inelasticHeatFraction
```

InelasticHeatFraction(...)

This method creates an InelasticHeatFraction object.

Path

```
mdb.models[name].materials[name].InelasticHeatFraction
session.odbs[name].materials[name].InelasticHeatFraction
```

Required arguments

None.

Optional arguments

fraction A Float specifying the fraction of inelastic dissipation rate that appears as a heat flux per unit volume. The fraction may include a unit conversion factor if required. Possible values are 0.0 ≤*fraction*≤ 1.0. The default value is 0.9.

Return value

An InelasticHeatFraction object.

Exceptions

RangeError.

setValues(...)

This method modifies the InelasticHeatFraction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *InelasticHeatFraction* method.

Return value

`None.`

Exceptions

RangeError.

Members

The InelasticHeatFraction object has members with the same names and descriptions as the arguments to the *InelasticHeatFraction* method.

Corresponding analysis keywords

- **INELASTIC HEAT FRACTION*

JouleHeatFraction object

The JouleHeatFraction object defines the fraction of electric energy released as heat.

Access

```
import material
mdb.models[name].materials[name].jouleHeatFraction
import odbMaterial
session.odbs[name].materials[name].jouleHeatFraction
```

JouleHeatFraction(...)

This method creates a JouleHeatFraction object.

Path

```
mdb.models[name].materials[name].JouleHeatFraction
session.odbs[name].materials[name].JouleHeatFraction
```

Required arguments

None.

Optional arguments

fraction A Float specifying the fraction of electrical energy released as heat, including any unit conversion factor. The default value is 1.0.

Return value

A JouleHeatFraction object.

Exceptions

RangeError.

setValues(...)

This method modifies the JouleHeatFraction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *JouleHeatFraction* method.

Return value

`None.`

Exceptions

RangeError.

Members

The JouleHeatFraction object has members with the same names and descriptions as the arguments to the *JouleHeatFraction* method.

Corresponding analysis keywords

- **JOULE HEAT FRACTION*

LatentHeat object

The LatentHeat object specifies a material's latent heat.

Access

```
import material
mdb.models[name].materials[name].latentHeat
import odbMaterial
session.odbs[name].materials[name].latentHeat
```

LatentHeat(...)

This method creates a LatentHeat object.

Path

```
mdb.models[name].materials[name].LatentHeat
session.odbs[name].materials[name].LatentHeat
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- Latent heat per unit mass.
- Solidus temperature.
- Liquidus temperature.

Return value

A LatentHeat object.

Exceptions

RangeError.

setValues(...)

This method modifies the LatentHeat object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LatentHeat* method.

Return value

`None.`

Exceptions

RangeError.

Members

The LatentHeat object has members with the same names and descriptions as the arguments to the *LatentHeat* method.

Corresponding analysis keywords

- **LATENT HEAT*

LowDensityFoam object

The LowDensityFoam object specifies properties for low-density foam.

Access

```
import material
mdb.models[name].materials[name].lowDensityFoam
import odbMaterial
session.odbs[name].materials[name].lowDensityFoam
```

LowDensityFoam(...)

This method creates a LowDensityFoam object.

Path

```
mdb.models[name].materials[name].LowDensityFoam
session.odbs[name].materials[name].LowDensityFoam
```

Required arguments

None.

Optional arguments

elementRemoval A Boolean specifying whether elements are removed if exceeding maximum principal tensile stress. This argument is valid only when *maxAllowablePrincipalStress* is defined. The default value is OFF.

maxAllowablePrincipalStress `None` or a Float specifying the maximum allowable principal tensile stress. The default value is `None`.

extrapolateStressStrainCurve A Boolean specifying whether the stress-strain curve is extrapolated if exceeding maximum strain rate. The default value is OFF.

strainRateType A SymbolicConstant specifying strain rate measure used for constitutive calculations. Possible values are PRINCIPAL and VOLUMETRIC. The default value is VOLUMETRIC.

mu0 A Float specifying the relaxation coefficient μ_0. The default value is 10^{-4}.

mu1 A Float specifying the relaxation coefficient μ_1. The default value is 0.5×10^{-2}.

alpha A Float specifying the relaxation coefficient α. The default value is 2.0.

Return value

A LowDensityFoam object.

Exceptions

RangeError.

setValues(...)

This method modifies the LowDensityFoam object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LowDensityFoam* method.

Return value

`None.`

Exceptions

RangeError.

Members

The LowDensityFoam object has members with the same names and descriptions as the arguments to the *LowDensityFoam* method.

In addition, the LowDensityFoam object can have the following members:

uniaxialTensionTestData

A *UniaxialTestData* object.

uniaxialCompressionTestData

A *UniaxialTestData* object.

Corresponding analysis keywords

- **LOW DENSITY FOAM*

MagneticPermeability object

The MagneticPermeability object specifies magnetic permeability.

Access

```
import material
mdb.models[name].materials[name].magneticPermeability
import odbMaterial
session.odbs[name].materials[name].magneticPermeability
```

MagneticPermeability(...)

This method creates a MagneticPermeability object.

Path

```
mdb.models[name].materials[name].MagneticPermeability
session.odbs[name].materials[name].MagneticPermeability
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below in "Table data." If *type*=ORTHOTROPIC and nonlinearBH=ON, the data specified in the *table* is for the first direction and *table2* and *table3* must be specified.

table2
A sequence of sequences of Floats specifying the items described below in "Table data" in the second direction. *table2* must be specified only if *type*=ORTHOTROPIC and nonlinearBH=ON.

table3
A sequence of sequences of Floats specifying the items described below in "Table data" in the third direction. *table3* must be specified only if *type*=ORTHOTROPIC and nonlinearBH=ON.

Optional arguments

type
A SymbolicConstant specifying the type of magnetic permeability. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.

frequencyDependency
A Boolean specifying whether the data depend on frequency. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

nonlinearBH
A Boolean specifying whether the magnetic behavior is nonlinear and available in tabular form of magnetic flux density versus magnetic field values. The default value is OFF.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Magnetic permeability.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ISOTROPIC, and *nonlinearBH*=TRUE, the table data specify the following:

- Magntitude of the magnetic flux density vector.
- Magnitude of the magnetic field vector.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- μ_{11}^{E}.
- μ_{22}^{E}.
- μ_{33}^{E}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, and *nonlinearBH*=TRUE, the table data specify the following:

- Magntitude of the magnetic flux density vector in the first direction.
- Magnitude of the magnetic field vector in the second direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- μ_{11}^{E}.
- μ_{12}^{E}.

- μ_{22}^{E}.
- μ_{13}^{E}.
- μ_{23}^{E}.
- μ_{33}^{E}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MagneticPermeability object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MagneticPermeability object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MagneticPermeability* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MagneticPermeability object has members with the same names and descriptions as the arguments to the *MagneticPermeability* method.

Corresponding analysis keywords

- **MAGNETIC PERMEABILITY*

MeanFieldHomogenization object

The MeanFieldHomogenization object specifies the multiscale material definition.

Access

```
import material
mdb.models[name].materials[name].meanFieldHomogenization
import odbMaterial
session.odbs[name].materials[name].meanFieldHomogenization
```

MeanFieldHomogenization(...)

This method creates a MeanFieldHomogenization object.

Path

```
mdb.models[name].materials[name].MeanFieldHomogenization
session.odbs[name].materials[name].MeanFieldHomogenization
```

Required arguments

None.

Optional arguments

angleSubdivision An Int specifying the number of angle increments used for the discretization of the orientation space.

formulation A SymbolicConstant specifying the type of homogenization model. Possible values are MT, REUSS, VOIGT, INVERSED_MT, BALANCED, and UNSPECIFIED. The default value is MT.

isotropization A SymbolicConstant specifying the type of isotropization method. Possible values are ALLISO, EISO, and PISO. The default value is ALLISO.

uniformMatrixStrain A SymbolicConstant specifying whether the average strain in the matrix is uniform across all pseudo-grains. Possible values are NO and YES. The default value is NO.

Return value

A MeanFieldHomogenization object.

Exceptions

RangeError.

setValues(...)

This method modifies the MeanFieldHomogenization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MeanFieldHomogenization(...)* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MeanFieldHomogenization object has members with the same names and descriptions as the arguments to the *MeanFieldHomogenization(...)* method.

Corresponding analysis keywords

- **MEAN FIELD HOMOGENIZATION*

MeanFieldInclusion object

The MeanFieldInclusion object specifies the inclusion type multiscale material property.

Access

```
import material
mdb.models[name].materials[name].constituents[name]
import odbMaterial
session.odbs[name].materials[name].constituents[name]
```

MeanFieldInclusion(...)

This method creates a MeanFieldInclusion object.

Path

```
mdb.models[name].materials[name].meanFieldHomogenization.MeanFieldInclusion
session.odbs[name].materials[name].meanFieldHomogenization.MeanFieldInclusion
```

Required arguments

name A String specifying the constituent repository key.

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

material A String specifying the name of the material.

isotropizationCoefficient A Float specifying the factor used for scaling the plastic strain of the constituent when calculating the isotropic part of the tangent.

volumeFractionType A SymbolicConstant specifying the type of volume fraction. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

volumeFractionFieldName A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.

aspectRatioType A SymbolicConstant specifying the type of aspect ratio. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

aspectRatioFieldName A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.

orientationTensorType A SymbolicConstant specifying the type of orientation tensor. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

orientationTensorFieldName	A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.
shape	A SymbolicConstant specifying the type of inclusion shapes. Possible values are SPHERE, PROLATE, OBLATE, CYLINDER, PENNY, and ELLIPTIC_CYLINDER. The default value is SPHERE.
direction	A SymbolicConstant specifying the type of inclusion direction. Possible values are FIXED, RANDOM3D, and ORIENTATION_TENSOR.
strainConcentrationTensor	A sequence of Floats defining the 36 components of the strain concentration tensor.
temperatureGradientConcentrationTensor	A sequence of Floats defining the 9 components of the temperature gradient concentration tensor.

Table data

The table data specify the following:

- Volume fraction.
- Aspect ratio.
- Components of the direction vector defined in the local coordinate system when *direction*=FIXED. Components of the second-order orientation tensor in the local coordinate system when *direction*=ORIENTATION_TENSOR.
- Etc.

Return value

A MeanFieldInclusion object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MeanFieldInclusion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MeanFieldInclusion(...)* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MeanFieldInclusion object has members with the same names and descriptions as the arguments to the *MeanFieldInclusion(...)* method.

Corresponding analysis keywords

- **CONSTITUENT*

MeanFieldMatrix object

The MeanFieldMatrix object specifies the matrix property.

Access

```
import material
mdb.models[name].materials[name].constituents[name]
import odbMaterial
session.odbs[name].materials[name].constituents[name]
```

MeanFieldMatrix(...)

This method creates a MeanFieldMatrix object.

Path

```
mdb.models[name].materials[name].meanFieldHomogenization.MeanFieldMatrix
session.odbs[name].materials[name].meanFieldHomogenization.MeanFieldMatrix
```

Required arguments

name	A String specifying the constituent repository key.

Optional arguments

material	A String specifying the name of the material.
isotropizationCoefficient	A Float specifying the factor used for scaling the plastic strain of the constituent when calculating the isotropic part of the tangent.

Return value

A MeanFieldMatrix object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MeanFieldMatrix object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MeanFieldMatrix(...)* method.

Return value
`None.`

Exceptions
RangeError.

Members
The MeanFieldMatrix object has members with the same names and descriptions as the arguments to the *MeanFieldMatrix(...)* method.

Corresponding analysis keywords
- **CONSTITUENT*

MeanFieldVoid object

The MeanFieldVoid object specifies the void inclusion property.

Access

```
import material
mdb.models[name].materials[name].constituents[name]
import odbMaterial
session.odbs[name].materials[name].constituents[name]
```

MeanFieldVoid(...)

This method creates a MeanFieldVoid object.

Path

```
mdb.models[name].materials[name].meanFieldHomogenization.MeanFieldVoid
session.odbs[name].materials[name].meanFieldHomogenization.MeanFieldVoid
```

Required arguments

name A String specifying the constituent repository key.

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

material A String specifying the name of the material.

isotropizationCoefficient A Float specifying the factor used for scaling the plastic strain of the constituent when calculating the isotropic part of the tangent.

volumeFractionType A SymbolicConstant specifying the type of volume fraction. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

volumeFractionFieldName A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.

aspectRatioType A SymbolicConstant specifying the type of aspect ratio. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

aspectRatioFieldName A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.

orientationTensorType A SymbolicConstant specifying the type of orientation tensor. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

orientationTensorFieldName A String specifying the name of the *AnalyticalField* object or *DiscreteField* object.

shape A SymbolicConstant specifying the type of inclusion shapes. Possible values are SPHERE, PROLATE, OBLATE, CYLINDER, PENNY, and ELLIPTIC_CYLINDER. The default value is SPHERE.

direction A SymbolicConstant specifying the type of inclusion direction. Possible values are FIXED, RANDOM3D, and ORIENTATION_TENSOR.

strainConcentrationTensor A sequence of Floats defining the 36 components of the strain concentration tensor.

temperatureGradientConcentrationTensor A sequence of Floats defining the 9 components of the temperature gradient concentration tensor.

Table data

The table data specify the following:

- Volume fraction.
- Aspect ratio.
- Components of the direction vector defined in the local coordinate system when *direction*=FIXED. Components of the second-order orientation tensor in the local coordinate system when *direction*=ORIENTATION_TENSOR.
- Etc.

Return value

A MeanFieldVoid object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MeanFieldVoid object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MeanFieldVoid(...)* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MeanFieldVoid object has members with the same names and descriptions as the arguments to the *MeanFieldVoid(...)* method.

Corresponding analysis keywords

- **CONSTITUENT*

MohrCoulombHardening object

The MohrCoulombHardening object specifies hardening for the Mohr-Coulomb plasticity model.

Access

```
import material
mdb.models[name].materials[name].mohrCoulombPlasticity\
.mohrCoulombHardening
import odbMaterial
session.odbs[name].materials[name].mohrCoulombPlasticity\
.mohrCoulombHardening
```

MohrCoulombHardening(...)

This method creates a MohrCoulombHardening object.

Path

```
mdb.models[name].materials[name].mohrCoulombPlasticity\
.MohrCoulombHardening
session.odbs[name].materials[name].mohrCoulombPlasticity\
.MohrCoulombHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Cohesion yield stress.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MohrCoulombHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the MohrCoulombHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MohrCoulombHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MohrCoulombHardening object has members with the same names and descriptions as the arguments to the *MohrCoulombHardening* method.

Corresponding analysis keywords

- **MOHR COULOMB HARDENING*

MohrCoulombPlasticity object

The MohrCoulombPlasticity object specifies the extended Mohr-Coulomb plasticity model.

Access

```
import material
mdb.models[name].materials[name].mohrCoulombPlasticity
import odbMaterial
session.odbs[name].materials[name].mohrCoulombPlasticity
```

MohrCoulombPlasticity(...)

This method creates a MohrCoulombPlasticity object.

Path

```
mdb.models[name].materials[name].MohrCoulombPlasticity
session.odbs[name].materials[name].MohrCoulombPlasticity
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

deviatoricEccentricity
`None` or a Float specifying the flow potential eccentricity in the deviatoric plane, e; $1/2 \leq e \leq 1.0$. If *deviatoricEccentricity*=`None`, Abaqus calculates the value using the specified Mohr-Coulomb angle of friction. The default value is `None`.

meridionalEccentricity
A Float specifying the flow potential eccentricity in the meridional plane, ϵ. The default value is 0.1.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

useTensionCutoff
A Boolean specifying whether tension cutoff specification is needed. The default value is OFF.

Table data

The table data specify the following:

- Friction angle (given in degrees), ϕ, at high confining pressure in the p–$R_{mc}q$ plane.
- Dilation angle, ψ, at high confining pressure in the p–$R_{mw}q$ plane.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MohrCoulombPlasticity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MohrCoulombPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MohrCoulombPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The MohrCoulombPlasticity object has members with the same names and descriptions as the arguments to the *MohrCoulombPlasticity* method.

In addition, the MohrCoulombPlasticity object can have the following members:

mohrCoulombHardening	A *MohrCoulombHardening* object.
tensionCutOff	A *TensionCutOff* object.

Corresponding analysis keywords

- **MOHR COULOMB*

MoistureSwelling object

The MoistureSwelling object defines moisture-driven swelling.

Access

```
import material
mdb.models[name].materials[name].moistureSwelling
import odbMaterial
session.odbs[name].materials[name].moistureSwelling
```

MoistureSwelling(...)

This method creates a MoistureSwelling object.

Path

```
mdb.models[name].materials[name].MoistureSwelling
session.odbs[name].materials[name].MoistureSwelling
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- Volumetric moisture swelling strain, ε^{ms}.
- Saturation, s. This value must lie in the range $0.0 \le s \le 1.0$.

Return value

A MoistureSwelling object.

Exceptions

None.

setValues(...)

This method modifies the MoistureSwelling object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MoistureSwelling* method.

Return value
`None.`

Exceptions
None.

Members
The MoistureSwelling object has members with the same names and descriptions as the arguments to the *MoistureSwelling* method.

In addition, the MoistureSwelling object can have the following member:

ratios A *Ratios* object.

Corresponding analysis keywords
- **MOISTURE SWELLING*

MullinsEffect object

The MullinsEffect specifies properties for mullins data.

Access

```
import material
mdb.models[name].materials[name].mullinsEffect
import odbMaterial
session.odbs[name].materials[name].mullinsEffect
```

Members

The MullinsEffect object can have the following members:

definition

A SymbolicConstant specifying the method of specifying the data. Possible values are USER, CONSTANTS, and TEST_DATA. The default value is CONSTANTS.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

properties

An Int specifying the number of property values needed as data for the user-defined hyperelastic material. The default value is 0.

table

A tuple of tuples of Floats specifying the items described below. The default value is an empty sequence.

uniaxialTests

A *UniaxialTestDataArray* object.

biaxialTests

A *BiaxialTestDataArray* object.

planarTests

A *PlanarTestDataArray* object.

Ornl object

The Ornl object specifies the constitutive model developed by Oak Ridge National Laboratory.

Access

```
import material
mdb.models[name].materials[name].creep.ornl
mdb.models[name].materials[name].plastic.ornl
import odbMaterial
session.odbs[name].materials[name].creep.ornl
session.odbs[name].materials[name].plastic.ornl
```

Ornl(...)

This method creates an Ornl object.

Path

```
mdb.models[name].materials[name].creep.Ornl
mdb.models[name].materials[name].plastic.Ornl
session.odbs[name].materials[name].creep.Ornl
session.odbs[name].materials[name].plastic.Ornl
```

Required arguments

None.

Optional arguments

a
A Float specifying the saturation rates for kinematic shift caused by creep strain, as defined by Equation (15) of Section 4.3.3–3 of the Nuclear Standard. The default value corresponds to that section of the Standard. Set *a*=0.0 to use the 1986 revision of the Standard. The default value is 0.3.

h
`None` or a Float specifying the rate of kinematic shift with respect to creep strain [Equation (7) of Section 4.3.2–1 of the Nuclear Standard]. If *h*=`None`, the value of *h* is determined according to Section 4.3.3–3 of the 1981 revision of the Standard. Set *h*=0.0 to use the 1986 revision of the Standard. The default value is `None`.

reset
A Boolean specifying whether to invoke the optional α reset procedure described in Section 4.3.5 of the Nuclear Standard. The default value is OFF.

Return value

An Ornl object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Ornl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Ornl* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Ornl object has members with the same names and descriptions as the arguments to the *Ornl* method.

Corresponding analysis keywords

- **ORNL*

Permeability object

The Permeability object defines permeability for pore fluid flow.

Access

```
import material
mdb.models[name].materials[name].permeability
import odbMaterial
session.odbs[name].materials[name].permeability
```

Permeability(...)

This method creates a Permeability object.

Path

```
mdb.models[name].materials[name].Permeability
session.odbs[name].materials[name].Permeability
```

Required arguments

specificWeight	A Float specifying the specific weight of the wetting liquid, γ_w.
inertialDragCoefficient	A Float specifying The inertial drag coefficient of the wetting liquid, γ_w.
table	A sequence of sequences of Floats specifying the items described below.

Optional arguments

type	A SymbolicConstant specifying the type of permeability. Possible values are ISOTROPIC, ORTHOTROPIC, and ANISOTROPIC. The default value is ISOTROPIC.
temperatureDependency	A Boolean specifying whether the data depend on temperature. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- k.
- Void ratio, e.
- Temperature, if the data depend on temperature.

If *type*=ORTHOTROPIC, the table data specify the following:

- k_{11}.

- k_{22}.
- k_{33}.
- Void ratio, e.
- Temperature, if the data depend on temperature.

If *type*=ANISOTROPIC, the table data specify the following:

- k_{11}.
- k_{12}.
- k_{22}.
- k_{13}.
- k_{23}.
- k_{33}.
- Void ratio, e.
- Temperature, if the data depend on temperature.

Return value

A Permeability object.

Exceptions

RangeError.

setValues(...)

This method modifies the Permeability object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Permeability* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Permeability object has members with the same names and descriptions as the arguments to the *Permeability* method.

In addition, the Permeability object can have the following members:

saturationDependence
A *SaturationDependence* object specifying the dependence of the permeability of a material on the saturation of the wetting liquid.

velocityDependence
A *VelocityDependence* object specifying the dependence of the permeability of a material on the velocity of fluid flow.

Corresponding analysis keywords

- **PERMEABILITY*

Piezoelectric object

The Piezoelectric object specifies piezoelectric material properties.

Access

```
import material
mdb.models[name].materials[name].piezoelectric
import odbMaterial
session.odbs[name].materials[name].piezoelectric
```

Piezoelectric(...)

This method creates a Piezoelectric object.

Path

```
mdb.models[name].materials[name].Piezoelectric
session.odbs[name].materials[name].Piezoelectric
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of material coefficients for the piezoelectric property. Possible values are STRAIN and STRESS. The default value is STRESS.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRESS, the table data specify the following:

- $e^{\varphi}_{1\,11}$.
- $e^{\varphi}_{1\,22}$.
- $e^{\varphi}_{1\,33}$.
- $e^{\varphi}_{1\,12}$.
- $e^{\varphi}_{1\,13}$.
- $e^{\varphi}_{1\,23}$.
- $e^{\varphi}_{2\,11}$.

- $e^{\varphi}_{2\,22}$.
- $e^{\varphi}_{2\,33}$.
- $e^{\varphi}_{2\,12}$.
- $e^{\varphi}_{2\,13}$.
- $e^{\varphi}_{2\,23}$.
- $e^{\varphi}_{3\,11}$.
- $e^{\varphi}_{3\,22}$.
- $e^{\varphi}_{3\,33}$.
- $e^{\varphi}_{3\,12}$.
- $e^{\varphi}_{3\,13}$.
- $e^{\varphi}_{3\,23}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=STRAIN, the table data specify the following:

- $d^{\varphi}_{1\,11}$.
- $d^{\varphi}_{1\,22}$.
- $d^{\varphi}_{1\,33}$.
- $d^{\varphi}_{1\,12}$.
- $d^{\varphi}_{1\,13}$.
- $d^{\varphi}_{1\,23}$.
- $d^{\varphi}_{2\,11}$.
- $d^{\varphi}_{2\,22}$.
- $d^{\varphi}_{2\,33}$.
- $d^{\varphi}_{2\,12}$.
- $d^{\varphi}_{2\,13}$.
- $d^{\varphi}_{2\,23}$.
- $d^{\varphi}_{3\,11}$.

- $d^{\varphi}_{3\,22}$.
- $d^{\varphi}_{3\,33}$.
- $d^{\varphi}_{3\,13}$.
- $d^{\varphi}_{3\,23}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Piezoelectric object.

Exceptions

None.

setValues(...)

This method modifies the Piezoelectric object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Piezoelectric* method.

Return value

`None.`

Exceptions

None.

Members

The Piezoelectric object has members with the same names and descriptions as the arguments to the *Piezoelectric* method.

Corresponding analysis keywords

- **PIEZOELECTRIC*

PlanarTestData object

The PlanarTestData object specifies planar test (or pure shear) data (compression and/or tension).

Access

```
import material
mdb.models[name].materials[name].hyperelastic.planarTestData
mdb.models[name].materials[name].hyperfoam.planarTestData
mdb.models[name].materials[name].mullinsEffect.planarTests[i]
import odbMaterial
session.odbs[name].materials[name].hyperelastic.planarTestData
session.odbs[name].materials[name].hyperfoam.planarTestData
session.odbs[name].materials[name].mullinsEffect.planarTests[i]
```

PlanarTestData(...)

This method creates a PlanarTestData object.

Path

```
mdb.models[name].materials[name].hyperelastic.PlanarTestData
mdb.models[name].materials[name].hyperfoam.PlanarTestData
mdb.models[name].materials[name].mullinsEffect.PlanarTestData
session.odbs[name].materials[name].hyperelastic.PlanarTestData
session.odbs[name].materials[name].hyperfoam.PlanarTestData
session.odbs[name].materials[name].mullinsEffect.PlanarTestData
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

smoothing
`None` or an Int specifying the value for smoothing. If *smoothing*=`None`, no smoothing is employed. The default value is `None`.

lateralNominalStrain
A Boolean specifying whether to include lateral nominal strain. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic material model, the table data specify the following:

- Nominal stress, T_S.

- Nominal strain in the direction of loading, ϵ_S.

For a hyperfoam material model, the table data specify the following:

- Nominal stress, T_L.
- Nominal strain in the direction of loading, ϵ_p.
- Nominal transverse strain, ϵ_3. The default value is 0.

Return value

A PlanarTestData object.

Exceptions

None.

`setValues(...)`

This method modifies the PlanarTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PlanarTestData* method.

Return value

`None.`

Exceptions

None.

Members

The PlanarTestData object has members with the same names and descriptions as the arguments to the *PlanarTestData* method.

Corresponding analysis keywords

- **PLANAR TEST DATA*

Plastic object

The Plastic object specifies a metal plasticity model.

Access

```
import material
mdb.models[name].materials[name].plastic
import odbMaterial
session.odbs[name].materials[name].plastic
```

Plastic(...)

This method creates a Plastic object.

Path

```
mdb.models[name].materials[name].Plastic
session.odbs[name].materials[name].Plastic
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

hardening A SymbolicConstant specifying the type of hardening. Possible values are ISOTROPIC, KINEMATIC, COMBINED, JOHNSON_COOK, and USER. The default value is ISOTROPIC.

rate A Boolean specifying whether the data depend on rate. The default value is OFF.

dataType A SymbolicConstant specifying the type of combined hardening. This argument is only valid if *hardening*=COMBINED. Possible values are HALF_CYCLE, PARAMETERS, and STABILIZED. The default value is HALF_CYCLE.

strainRangeDependency A Boolean specifying whether the data depend on strain range. This argument is only valid if *hardening*=COMBINED and *dataType*=STABILIZED. The default value is OFF.

numBackstresses An Int specifying the number of backstresses. This argument is only valid if *hardening*=COMBINED. The default value is 1.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

extrapolation A SymbolicConstant specifying the extrapolation method for the yield stress with respect to the equivalent plastic strain. This argument is valid only if *hardening*=ISOTROPIC. Possible values are CONSTANT and LINEAR . The default value is CONSTANT.

Table data

If *hardening*=ISOTROPIC, or if *hardening*=COMBINED and *dataType*=HALF_CYCLE, the table data specify the following:

- Yield stress.
- Plastic strain.
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=COMBINED and *dataType*=STABILIZED, the table data specify the following:

- Yield stress.
- Plastic strain.
- Strain range, if the data depend on strain range.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=COMBINED and *dataType*=PARAMETERS, the table data specify the following:

- Yield stress at zero plastic strain.
- The first kinematic hardening parameter, C_1.
- The first kinematic hardening parameter, γ_1.
- If applicable, the second kinematic hardening parameter, C_2.
- If applicable, the second kinematic hardening parameter, γ_2.
- Etc.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=KINEMATIC, the table data specify the following:

- Yield stress.
- Plastic strain.
- Temperature, if the data depend on temperature.

If *hardening*=JOHNSON_COOK, the table data specify the following:

- A.
- B.

- n.
- m.
- Melting temperature.
- Transition temperature.

If *hardening*=USER, the table data specify the following:

- Hardening properties.

Return value

A Plastic object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Plastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Plastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Plastic object has members with the same names and descriptions as the arguments to the *Plastic* method.

In addition, the Plastic object can have the following members:

rateDependent A *RateDependent* object.

potential A *Potential* object.

cyclicHardening A *CyclicHardening* object.

ornl An *Ornl* object.

cycledPlastic A *CycledPlastic* object.

annealTemperature	An *AnnealTemperature* object.
tensileFailure	A *TensileFailure* object.

Corresponding analysis keywords

- **PLASTIC*

PlasticityCorrection object

The PlasticityCorrection object specifies the elastic-plastic response of the material with linear elasticity.

Access

```
import material
mdb.models[name].materials[name].plasticityCorrection
```

PlasticityCorrection(...)

This method creates a PlasticityCorrection object.

Path

```
mdb.models[name].materials[name].PlasticityCorrection
```

Required arguments

type	Set type=RAMBERG_OSGOOD to specify the Ramberg-Osgood relationship. Set type=TABULAR to specify the tabular form.
table	A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency	A Boolean specifying whether the data depend on temperature. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=RAMBERG_OSGOOD, the table specifies the following:

- K.
- n.
- Temperature if the data depend on temperature.
- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=TABULAR, the table specifies the following:

- Yield stress.
- Plastic strain.
- Temperature if the data depend on temperature.

- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PlasticityCorrection object.

Exceptions

None.

`setValues(...)`

This method modifies the PlasticityCorrection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PlasticityCorrection* method.

Return value

`None.`

Exceptions

None.

Members

The PlasticityCorrection object has members with the same names and descriptions as the arguments to the *PlasticityCorrection* method.

Corresponding analysis keywords

- **PLASTICITY CORRECTION*

PoreFluidExpansion object

The PoreFluidExpansion object specifies the thermal expansion coefficient for a hydraulic fluid.

Access

```
import material
mdb.models[name].materials[name].poreFluidExpansion
import odbMaterial
session.odbs[name].materials[name].poreFluidExpansion
```

PoreFluidExpansion(...)

This method creates a PoreFluidExpansion object.

Path

```
mdb.models[name].materials[name].PoreFluidExpansion
session.odbs[name].materials[name].PoreFluidExpansion
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

zero A Float specifying the value of θ^0. The default value is 0.0.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Mean coefficient of thermal expansion, α.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PoreFluidExpansion object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the PoreFluidExpansion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PoreFluidExpansion* method.

Return value

`None.`

Exceptions

RangeError.

Members

The PoreFluidExpansion object has members with the same names and descriptions as the arguments to the *PoreFluidExpansion* method.

Corresponding analysis keywords

- **EXPANSION*

PorousBulkModuli object

The PorousBulkModuli object defines bulk moduli for soils and rocks.

Access

```
import material
mdb.models[name].materials[name].porousBulkModuli
import odbMaterial
session.odbs[name].materials[name].porousBulkModuli
```

PorousBulkModuli(...)

This method creates a PorousBulkModuli object.

Path

```
mdb.models[name].materials[name].PorousBulkModuli
session.odbs[name].materials[name].PorousBulkModuli
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

Table data

- Bulk modulus of solid grains.
- Bulk modulus of permeating fluid.
- Temperature, if the data depend on temperature.

Return value

A PorousBulkModuli object.

Exceptions

None.

setValues(...)

This method modifies the PorousBulkModuli object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PorousBulkModuli* method.

Return value

`None.`

Exceptions

None.

Members

The PorousBulkModuli object has members with the same names and descriptions as the arguments to the *PorousBulkModuli* method.

Corresponding analysis keywords

- **POROUS BULK MODULI*

PorousElastic object

The PorousElastic object specifies elastic material properties for porous materials.

Access

```
import material
mdb.models[name].materials[name].porousElastic
import odbMaterial
session.odbs[name].materials[name].porousElastic
```

PorousElastic(...)

This method creates a PorousElastic object.

Path

```
mdb.models[name].materials[name].PorousElastic
session.odbs[name].materials[name].PorousElastic
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

shear A SymbolicConstant specifying the shear definition form. Possible values are G and POISSON. The default value is POISSON.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *shear*=G, the table data specify the following:

- The logarithmic bulk modulus, κ. (Dimensionless.)
- The shear modulus, G.
- The elastic tensile limit, p_t^{el}. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shear*=POISSON, the table data specify the following:

- The logarithmic bulk modulus, κ. (Dimensionless.)

- The Poisson's ratio, ν.
- The elastic tensile limit, p_t^{el}. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PorousElastic object.

Exceptions

RangeError.

setValues(...)

This method modifies the PorousElastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PorousElastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The PorousElastic object has members with the same names and descriptions as the arguments to the *PorousElastic* method.

Corresponding analysis keywords

- **POROUS ELASTIC*

PorousFailureCriteria object

The PorousFailureCriteria object specifies the material failure criteria for a porous metal.

Access

```
import material
mdb.models[name].materials[name].porousMetalPlasticity\
.porousFailureCriteria
import odbMaterial
session.odbs[name].materials[name].porousMetalPlasticity\
.porousFailureCriteria
```

PorousFailureCriteria(...)

This method creates a PorousFailureCriteria object.

Path

```
mdb.models[name].materials[name].porousMetalPlasticity\
.PorousFailureCriteria
session.odbs[name].materials[name].porousMetalPlasticity\
.PorousFailureCriteria
```

Required arguments

None.

Optional arguments

fraction

A Float specifying the void volume fraction at total failure, $f_F > 0$. The default value is 1.0.

criticalFraction

A Float specifying the critical void volume fraction, $f_c \geq 0$. The default value is 1.0.

Return value

A PorousFailureCriteria object.

Exceptions

RangeError.

setValues(...)

This method modifies the PorousFailureCriteria object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PorousFailureCriteria* method.

Return value

`None.`

Exceptions

RangeError.

Members

The PorousFailureCriteria object has members with the same names and descriptions as the arguments to the *PorousFailureCriteria* method.

Corresponding analysis keywords

- **POROUS FAILURE CRITERIA*

PorousMetalPlasticity object

The PorousMetalPlasticity object specifies a porous metal plasticity model.

Access

```
import material
mdb.models[name].materials[name].porousMetalPlasticity
import odbMaterial
session.odbs[name].materials[name].porousMetalPlasticity
```

PorousMetalPlasticity(...)

This method creates a PorousMetalPlasticity object.

Path

```
mdb.models[name].materials[name].PorousMetalPlasticity
session.odbs[name].materials[name].PorousMetalPlasticity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

relativeDensity `None` or a Float specifying the initial relative density of the material, r_0. The default value is `None`.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- q_1.
- q_2.
- q_3.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PorousMetalPlasticity object.

Exceptions

RangeError.

setValues(...)

This method modifies the PorousMetalPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PorousMetalPlasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The PorousMetalPlasticity object has members with the same names and descriptions as the arguments to the *PorousMetalPlasticity* method.

In addition, the PorousMetalPlasticity object can have the following members:

porousFailureCriteria
A *PorousFailureCriteria* object.

voidNucleation
A *VoidNucleation* object.

Corresponding analysis keywords

- **POROUS METAL PLASTICITY*

Potential object

The Potential object defines an anisotropic yield/creep model.

Access

```
import material
mdb.models[name].materials[name].creep.potential
mdb.models[name].materials[name].plastic.potential
mdb.models[name].materials[name].viscous.potential
import odbMaterial
session.odbs[name].materials[name].creep.potential
session.odbs[name].materials[name].plastic.potential
session.odbs[name].materials[name].viscous.potential
```

Potential(...)

This method creates a Potential object.

Path

```
mdb.models[name].materials[name].creep.Potential
mdb.models[name].materials[name].plastic.Potential
mdb.models[name].materials[name].viscous.Potential
session.odbs[name].materials[name].creep.Potential
session.odbs[name].materials[name].plastic.Potential
session.odbs[name].materials[name].viscous.Potential
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- R_{11}.
- R_{22}.
- R_{33}.
- R_{12}.
- R_{13}.
- R_{23}.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Potential object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Potential object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Potential* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Potential object has members with the same names and descriptions as the arguments to the *Potential* method.

Corresponding analysis keywords

- **POTENTIAL*

PressureEffect object

The PressureEffect object defines equivalent pressure stress driven mass diffusion.

Access

```
import material
mdb.models[name].materials[name].diffusivity.pressureEffect
import odbMaterial
session.odbs[name].materials[name].diffusivity.pressureEffect
```

PressureEffect(...)

This method creates a PressureEffect object.

Path

```
mdb.models[name].materials[name].diffusivity.PressureEffect
session.odbs[name].materials[name].diffusivity.PressureEffect
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Pressure stress factor, κ_p.
- Concentration.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PressureEffect object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the PressureEffect object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PressureEffect* method.

Return value

`None.`

Exceptions

RangeError.

Members

The PressureEffect object has members with the same names and descriptions as the arguments to the *PressureEffect* method.

Corresponding analysis keywords

- **KAPPA*

RateDependent object

The RateDependent object defines a rate-dependent viscoplastic model.

Access

```
import material
mdb.models[name].materials[name].crushableFoam.rateDependent
mdb.models[name].materials[name].druckerPrager.rateDependent
mdb.models[name].materials[name].plastic.rateDependent
import odbMaterial
session.odbs[name].materials[name].crushableFoam.rateDependent
session.odbs[name].materials[name].druckerPrager.rateDependent
session.odbs[name].materials[name].plastic.rateDependent
```

RateDependent(...)

This method creates a RateDependent object.

Path

```
mdb.models[name].materials[name].crushableFoam.RateDependent
mdb.models[name].materials[name].druckerPrager.RateDependent
mdb.models[name].materials[name].plastic.RateDependent
session.odbs[name].materials[name].crushableFoam.RateDependent
session.odbs[name].materials[name].druckerPrager.RateDependent
session.odbs[name].materials[name].plastic.RateDependent
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

type
A SymbolicConstant specifying the type of rate-dependent data. Possible values are POWER_LAW, YIELD_RATIO, and JOHNSON_COOK. The default value is POWER_LAW.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=POWER_LAW, the table data specify the following:

- D.
- n.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=YIELD_RATIO, the table data specify the following:

- Yield stress ratio, $R = \overline{\sigma}/\sigma^0$.
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=JOHNSON_COOK, the table data specify the following:

- C.
- $\dot{\varepsilon}_0$.

Return value

A RateDependent object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the RateDependent object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RateDependent* method.

Return value

`None.`

Exceptions

RangeError.

Members

The RateDependent object has members with the same names and descriptions as the arguments to the *RateDependent* method.

Corresponding analysis keywords

- **RATE DEPENDENT*

Ratios object

The Ratios object specifies ratios that define anisotropic swelling.

Access

```
import material
mdb.models[name].materials[name].moistureSwelling.ratios
mdb.models[name].materials[name].swelling.ratios
import odbMaterial
session.odbs[name].materials[name].moistureSwelling.ratios
session.odbs[name].materials[name].swelling.ratios
```

Ratios(...)

This method creates a Ratios object.

Path

```
mdb.models[name].materials[name].moistureSwelling.Ratios
mdb.models[name].materials[name].swelling.Ratios
session.odbs[name].materials[name].moistureSwelling.Ratios
session.odbs[name].materials[name].swelling.Ratios
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- r_{11}.
- r_{22}.
- r_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Ratios object.

Exceptions

RangeError.

setValues(...)

This method modifies the Ratios object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Ratios* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Ratios object has members with the same names and descriptions as the arguments to the *Ratios* method.

Corresponding analysis keywords

- **RATIOS*

Regularization object

The Regularization object defines the tolerance to be used for regularizing material data.

Access

```
import material
mdb.models[name].materials[name].regularization
import odbMaterial
session.odbs[name].materials[name].regularization
```

Regularization(...)

This method creates a Regularization object.

Path

```
mdb.models[name].materials[name].Regularization
session.odbs[name].materials[name].Regularization
```

Required arguments

None.

Optional arguments

rtol A Float specifying the tolerance to be used for regularizing material data. The default value is 0.03.

strainRateRegularization A SymbolicConstant specifying the form of regularization of strain-rate-dependent material data. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.

Return value

A Regularization object.

Exceptions

RangeError.

setValues(...)

This method modifies the Regularization object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Regularization* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Regularization object has members with the same names and descriptions as the arguments to the *Regularization* method.

Corresponding analysis keywords

- **DASHPOT*

SaturationDependence object

The SaturationDependence object specifies the dependence of the permeability of a material on the saturation of the wetting liquid.

Access

```
import material
mdb.models[name].materials[name].permeability.saturationDependence
import odbMaterial
session.odbs[name].materials[name].permeability.saturationDependence
```

SaturationDependence(...)

This method creates a SaturationDependence object.

Path

```
mdb.models[name].materials[name].permeability.SaturationDependence
session.odbs[name].materials[name].permeability.SaturationDependence
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- k_s. (Dimensionless.)
- Saturation, s. (Dimensionless.)

Return value

A SaturationDependence object.

Exceptions

RangeError.

setValues(...)

This method modifies the SaturationDependence object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SaturationDependence* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SaturationDependence object has members with the same names and descriptions as the arguments to the *SaturationDependence* method.

Corresponding analysis keywords

- **PERMEABILITY*

ShearRetention object

The ShearRetention object defines the reduction of the shear modulus associated with crack surfaces in a *Concrete* model as a function of the tensile strain across the crack.

Access

```
import material
mdb.models[name].materials[name].concrete.shearRetention
import odbMaterial
session.odbs[name].materials[name].concrete.shearRetention
```

ShearRetention(...)

This method creates a ShearRetention object.

Path

```
mdb.models[name].materials[name].concrete.ShearRetention
session.odbs[name].materials[name].concrete.ShearRetention
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- ϱ^{close} for dry concrete. The default value is 1.0.
- ε^{max} for dry concrete. The default value is a very large number (full shear retention).
- ϱ^{close} for wet concrete. The default value is 1.0.
- ε^{max} for wet concrete. The default value is a very large number (full shear retention).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ShearRetention object.

Exceptions

RangeError.

setValues(...)

This method modifies the ShearRetention object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShearRetention* method.

Return value

`None.`

Exceptions

RangeError.

Members

The ShearRetention object has members with the same names and descriptions as the arguments to the *ShearRetention* method.

Corresponding analysis keywords

- **SHEAR RETENTION*

ShearTestData object

The ShearTestData object specifies the normalized shear creep compliance or relaxation modulus as a function of time.

Access

```
import material
mdb.models[name].materials[name].viscoelastic.shearTestData
import odbMaterial
session.odbs[name].materials[name].viscoelastic.shearTestData
```

ShearTestData(...)

This method creates a ShearTestData object.

Path

```
mdb.models[name].materials[name].viscoelastic.ShearTestData
session.odbs[name].materials[name].viscoelastic.ShearTestData
```

Required arguments

table

A sequence of sequences of Floats specifying values that depend on the *time* member of the *Viscoelastic* object.

If *time*=RELAXATION_TEST_DATA, the table data specify the following:

- Normalized shear relaxation modulus $g_R(t)$. $(0 \le g_R(t) \le 1)$.
- Time t. $(t > 0)$.

If *time*=CREEP_TEST_DATA, the table data specify the following:

- Normalized shear compliance $j_S(t)$. $(j_S(t) \ge 1)$.
- Time t. $(t > 0)$.

Optional arguments

shrinf

`None` or a Float specifying a normalized shear. The value of *shrinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is `None`.

If *time*=RELAXATION_TEST_DATA, *shrinf* specifies the value of the long-term, normalized shear modulus $g_R(\infty)$.

If *time*=CREEP_TEST_DATA, *shrinf* specifies the value of the long-term, normalized shear compliance $j_S(\infty)$.

Return value

A ShearTestData object.

Exceptions

None.

setValues(...)

This method modifies the ShearTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShearTestData* method.

Return value

`None.`

Exceptions

None.

Members

The ShearTestData object has members with the same names and descriptions as the arguments to the *ShearTestData* method.

Corresponding analysis keywords

- **SHEAR TEST DATA*

SimpleShearTestData object

The SimpleShearTestData object provides simple shear test data.

Access

```
import material
mdb.models[name].materials[name].hyperfoam.simpleShearTestData
import odbMaterial
session.odbs[name].materials[name].hyperfoam.simpleShearTestData
```

SimpleShearTestData(...)

This method creates a SimpleShearTestData object.

Path

```
mdb.models[name].materials[name].hyperfoam.SimpleShearTestData
session.odbs[name].materials[name].hyperfoam.SimpleShearTestData
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- Nominal shear stress, T_S.
- Nominal shear strain, γ.
- Nominal transverse stress, T_T (normal to edge with shear stress). This stress value is optional.

Return value

A SimpleShearTestData object.

Exceptions

None.

setValues(...)

This method modifies the SimpleShearTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SimpleShearTestData* method.

Return value

`None.`

Exceptions

None.

Members

The SimpleShearTestData object has members with the same names and descriptions as the arguments to the *SimpleShearTestData* method.

Corresponding analysis keywords

- **SIMPLE SHEAR TEST DATA*

Solubility object

The Solubility object specifies solubility.

Access

```
import material
mdb.models[name].materials[name].solubility
import odbMaterial
session.odbs[name].materials[name].solubility
```

Solubility(...)

This method creates a Solubility object.

Path

```
mdb.models[name].materials[name].Solubility
session.odbs[name].materials[name].Solubility
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Solubility.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Solubility object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Solubility object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Solubility* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Solubility object has members with the same names and descriptions as the arguments to the *Solubility* method.

Corresponding analysis keywords

- **SOLUBILITY*

SoretEffect object

The SoretEffect object defines temperature gradient driven mass diffusion.

Access

```
import material
mdb.models[name].materials[name].diffusivity.soretEffect
import odbMaterial
session.odbs[name].materials[name].diffusivity.soretEffect
```

SoretEffect(...)

This method creates a SoretEffect object.

Path

```
mdb.models[name].materials[name].diffusivity.SoretEffect
session.odbs[name].materials[name].diffusivity.SoretEffect
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Soret effect factor, κ_s.
- Concentration.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A SoretEffect object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SoretEffect object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SoretEffect* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SoretEffect object has members with the same names and descriptions as the arguments to the *SoretEffect* method.

Corresponding analysis keywords

- **KAPPA*

Sorption object

The Sorption object defines absorption and exsorption behaviors of a partially saturated porous medium in the analysis of coupled wetting liquid flow and porous medium stress.

Access

```
import material
mdb.models[name].materials[name].sorption
import odbMaterial
session.odbs[name].materials[name].sorption
```

Sorption(...)

This method creates a Sorption object.

Path

```
mdb.models[name].materials[name].Sorption
session.odbs[name].materials[name].Sorption
```

Required arguments

absorptionTable A sequence of sequences of Floats specifying the items described below.

Optional arguments

lawAbsorption A SymbolicConstant specifying absorption behavior. Possible values are LOG and TABULAR. The default value is TABULAR.

exsorption A Boolean specifying whether the exsorption data is specified. The default value is OFF.

lawExsorption A SymbolicConstant specifying exsorption behavior. Possible values are LOG and TABULAR. The default value is TABULAR.

scanning A Float specifying the slope of the scanning line, $(du_w/ds)|_s$. This slope must be positive and larger than the slope of the absorption or exsorption behaviors. The default value is 0.0.

exsorptionTable A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

Table data

If *lawAbsorption*=TABULAR or *lawExsorption*=TABULAR, the *absorptionTable* and *exsorptionTable* data respectively specify the following:

- Pore pressure, u_w.
- Saturation, s.

If *lawAbsorption*=LOG or *lawExsorption*=LOG, the *absorptionTable* and *exsorptionTable* data respectively specify the following:

- A.
- B.
- s_0.
- s_1.

Return value

A Sorption object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Sorption object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Sorption* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Sorption object has members with the same names and descriptions as the arguments to the *Sorption* method.

Corresponding analysis keywords

- **SORPTION*

SpecificHeat object

The SpecificHeat object specifies a material's specific heat.

Access

```
import material
mdb.models[name].materials[name].specificHeat
import odbMaterial
session.odbs[name].materials[name].specificHeat
```

SpecificHeat(...)

This method creates a SpecificHeat object.

Path

```
mdb.models[name].materials[name].SpecificHeat
session.odbs[name].materials[name].SpecificHeat
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

law A SymbolicConstant specifying the specific heat behavior. Possible values are CONSTANTVOLUME and CONSTANTPRESSURE. The default value is CONSTANTVOLUME.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Specific heat per unit mass.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A SpecificHeat object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SpecificHeat object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SpecificHeat* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SpecificHeat object has members with the same names and descriptions as the arguments to the *SpecificHeat* method.

Corresponding analysis keywords

- **SPECIFIC HEAT*

SuperElasticHardening object

The SuperElasticHardening object specifies the dependence of the yield stress on the total strain to define the piecewise linear hardening of a martensite material model.

Access

```
import material
mdb.models[name].materials[name].superElasticity.SuperElasticHardening
import odbMaterial
session.odbs[name].materials[name].superElasticity.SuperElasticHardening
```

SuperElasticHardening(...)

This method creates a SuperElasticHardening object.

Path

```
mdb.models[name].materials[name].superElasticity.SuperElasticHardening
session.odbs[name].materials[name].superElasticity.SuperElasticHardening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Table data

- Yield Stress.
- Total Strain.

Return value

A SuperElasticHardening object.

Exceptions

RangeError.

setValues(...)

This method modifies the SuperElasticHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SuperElasticHardening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SuperElasticHardening object has members with the same names and descriptions as the arguments to the *SuperElasticHardening* method.

Corresponding analysis keywords

- **SUPERELASTIC HARDENING*

SuperElasticHardeningModifications object

The SuperElasticHardeningModifications object specifies the variation of the transformation stress levels of a material model.

Access

```
import material
mdb.models[name].materials[name].superElasticity.SuperElasticHardening
import odbMaterial
session.odbs[name].materials[name].superElasticity.SuperElasticHardening
```

SuperElasticHardeningModifications(...)

This method creates a SuperElasticHardeningModifications object.

Path

```
mdb.models[name].materials[name].superElasticity.SuperElasticHardeningModifications
session.odbs[name].materials[name].superElasticity.SuperElasticHardeningModifications
```

Required arguments

table A sequence of sequences of Floats specifying the items described below or user-defined data if the dependence of the transformation stress levels on plastic strain is specified in a user subroutine.

Table data

- Start of Transformation (Loading).
- End of Transformation (Loading).
- Start of Transformation (Unloading).
- End of Transformation (Unloading).
- Plastic Strain.

Return value

A SuperElasticHardeningModifications object.

Exceptions

RangeError.

setValues(...)

This method modifies the SuperElasticHardeningModifications object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SuperElasticHardeningModifications* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SuperElasticHardeningModifications object has members with the same names and descriptions as the arguments to the *SuperElasticHardeningModifications* method.

Corresponding analysis keywords

- **SUPERELASTIC HARDENING MODIFICATIONS*

SuperElasticity object

The SuperElasticity object specifies a superelastic material model.

Access

```
import material
mdb.models[name].materials[name].superElasticity
import odbMaterial
session.odbs[name].materials[name].superElasticity
```

SuperElasticity(...)

This method creates a SuperElasticity object.

Path

```
mdb.models[name].materials[name].SuperElasticity
session.odbs[name].materials[name].SuperElasticity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

nonassociated `None` or a Float specifying the volumetric transformation strain. If *nonassociated*=`None`, the value of the volumetric transformation strain is equal to the uniaxial transformation strain. The default value is `None`.

Table data

- Young's Modulus (Martensite).
- Poisson's Ratio (Martensite).
- Transformation Strain.
- Start of Transformation (Loading).
- End of Transformation (Loading).
- Start of Transformation (Unloading).
- End of Transformation (Unloading).
- Start of Transformation in Compression (Loading).
- Reference Temperature.
- Loading.
- Unloading.

Return value

A SuperElasticity object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SuperElasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SuperElasticity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The SuperElasticity object has members with the same names and descriptions as the arguments to the *SuperElasticity* method.

In addition, the SuperElasticity object can have the following members:

superElasticHardening

A *SuperElasticHardening object*.

superElasticHardeningModifications

A *SuperElasticHardeningModifications object*.

Corresponding analysis keywords

- **SUPERELASTIC*

Swelling object

The Swelling object specifies time-dependent volumetric swelling for a material.

Access

```
import material
mdb.models[name].materials[name].swelling
import odbMaterial
session.odbs[name].materials[name].swelling
```

Swelling(...)

This method creates a Swelling object.

Path

```
mdb.models[name].materials[name].Swelling
session.odbs[name].materials[name].Swelling
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

This argument is valid only when *law*=INPUT.

Optional arguments

law
A SymbolicConstant specifying the type of data defining the swelling behavior. Possible values are INPUT and USER. The default value is INPUT.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Volumetric swelling strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Swelling object.

Exceptions

RangeError.

setValues(...)

This method modifies the Swelling object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Swelling* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Swelling object has members with the same names and descriptions as the arguments to the *Swelling* method.

In addition, the Swelling object can have the following member:

ratios
A *Ratios* object.

Corresponding analysis keywords

- **SWELLING*

TensileFailure object

The TensileFailure object specifies the material tensile failure.

Access

```
import material
mdb.models[name].materials[name].plastic.tensileFailure
mdb.models[name].materials[name].eos.tensileFailure
import odbMaterial
session.odbs[name].materials[name].plastic.tensileFailure
session.odbs[name].materials[name].eos.tensileFailure
```

tensileFailure(...)

This method creates a tensileFailure object.

Path

```
mdb.models[name].materials[name].plastic.TensileFailure
mdb.models[name].materials[name].eos.TensileFailure
session.odbs[name].materials[name].plastic.TensileFailure
session.odbs[name].materials[name].eos.TensileFailure
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

temperatureDependency A boolean specifying whether the data depends on temperature. The default value is OFF.

elementDeletion A boolean specifying whether element deletion is allowed. The default value is True.

pressure A SymbolicConstant specifying the pressure stress. The Possible values are BRITTLE and DUCTILE.

shear A SymbolicConstant specifying the deviatoric stress. Possible values are BRITTLE and DUCTILE.

Table data

- The Hydrostatic cutoff stress (positive in tension).
- Temperature, if the data depend on temperature.
- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.

- Etc.

Return value

An TensileFailure object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the TensileFailure object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TensileFailure* method.

Return value

`None.`

Exceptions

RangeError.

Members

The TensileFailure object has members with the same names and descriptions as the arguments to the *TensileFailure* method.

Corresponding analysis keywords

- **TENSILE FAILURE*

TensionCutOff object

The TensionCutOff object specifies tension cutoff for different material models for example the Mohr-Coulomb plasticity model.

Access

```
import material
mdb.models[name].materials[name].mohrCoulombPlasticity.tensionCutOff
import odbMaterial
session.odbs[name].materials[name].mohrCoulombPlasticity.tensionCutOff
```

TensionCutOff(...)

This method creates a TensionCutOff object.

Path

```
mdb.models[name].materials[name].mohrCoulombPlasticity.TensionCutOff
session.odbs[name].materials[name].mohrCoulombPlasticity.TensionCutOff
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tension cutoff stress.
- The value of the corresponding tensile plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A TensionCutOff object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the TensionCutOff object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TensionCutOff* method.

Return value

`None.`

Exceptions

RangeError.

Members

The TensionCutOff object has members with the same names and descriptions as the arguments to the *TensionCutOff* method.

Corresponding analysis keywords

- **TENSION CUTOFF*

TensionStiffening object

The TensionStiffening object defines the retained tensile stress normal to a crack in a *Concrete* model.

Access

```
import material
mdb.models[name].materials[name].concrete.tensionStiffening
import odbMaterial
session.odbs[name].materials[name].concrete.tensionStiffening
```

TensionStiffening(...)

This method creates a TensionStiffening object.

Path

```
mdb.models[name].materials[name].concrete.TensionStiffening
session.odbs[name].materials[name].concrete.TensionStiffening
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying how the postcracking behavior is defined. Possible values are DISPLACEMENT and STRAIN. The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Fraction of remaining stress to stress at cracking.
- Absolute value of the direct strain minus the direct strain at cracking.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Displacement, u_0, at which a linear loss of strength after cracking gives zero stress.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A TensionStiffening object.

Exceptions

RangeError.

setValues(...)

This method modifies the TensionStiffening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TensionStiffening* method.

Return value

`None.`

Exceptions

RangeError.

Members

The TensionStiffening object has members with the same names and descriptions as the arguments to the *TensionStiffening* method.

Corresponding analysis keywords

- **TENSION STIFFENING*

TriaxialTestData object

The TriaxialTestData object provides triaxial test data.

Access

```
import material
mdb.models[name].materials[name].druckerPrager.triaxialTestData
import odbMaterial
session.odbs[name].materials[name].druckerPrager.triaxialTestData
```

TriaxialTestData(...)

This method creates a TriaxialTestData object.

Path

```
mdb.models[name].materials[name].druckerPrager.TriaxialTestData
session.odbs[name].materials[name].druckerPrager.TriaxialTestData
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

a
`None` or a Float specifying the value of the material constant a. `None` is used when the value is unknown or it is not held fixed at the input value. The default value is `None`.

b
`None` or a Float specifying the value of the material constant b. `None` is used when the value is unknown or it is not held fixed at the input value. The default value is `None`.

pt
`None` or a Float specifying the value of the material constant p_t. `None` is used when the value is unknown or it is not held fixed at the input value. The default value is `None`.

Table data

- Sign and magnitude of confining stress, $\sigma_1 = \sigma_2$.
- Sign and magnitude of the stress in loading direction, σ_3.

Return value

A TriaxialTestData object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the TriaxialTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TriaxialTestData* method.

Return value

`None.`

Exceptions

RangeError.

Members

The TriaxialTestData object has members with the same names and descriptions as the arguments to the *TriaxialTestData* method.

Corresponding analysis keywords

- **TRIAXIAL TEST DATA*

Trs object

The Trs object defines the temperature-time shift for time history viscoelastic analysis.

Access

```
import material
mdb.models[name].materials[name].viscoelastic.trs
mdb.models[name].materials[name].viscosity.trs
import odbMaterial
session.odbs[name].materials[name].viscoelastic.trs
session.odbs[name].materials[name].viscosity.trs
```

Trs(...)

This method creates a Trs object.

Path

```
mdb.models[name].materials[name].viscoelastic.Trs
mdb.models[name].materials[name].viscosity.Trs
session.odbs[name].materials[name].viscoelastic.Trs
session.odbs[name].materials[name].viscosity.Trs
```

Required arguments

None.

Optional arguments

definition A SymbolicConstant specifying the definition of the shift function. Possible values are WLF, ARRHENIUS, and USER. The default value is WLF.

table A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

This argument is valid only when *definition*=WLF.

Table data

- Reference temperature, θ_0.
- Calibration constant, C_1.
- Calibration constant, C_2.

Return value

A Trs object.

Exceptions

None.

setValues(...)

This method modifies the Trs object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Trs* method.

Return value

`None.`

Exceptions

None.

Members

The Trs object has members with the same names and descriptions as the arguments to the *Trs* method.

Corresponding analysis keywords

- **TRS*

UniaxialTestData object

The UniaxialTestData object provides uniaxial test data (compression and/or tension).

Access

```
import material
mdb.models[name].materials[name].hyperelastic.uniaxialTestData
mdb.models[name].materials[name].hyperfoam.uniaxialTestData
mdb.models[name].materials[name].lowDensityFoam\
.uniaxialCompressionTestData
mdb.models[name].materials[name].lowDensityFoam\
.uniaxialTensionTestData
mdb.models[name].materials[name].mullinsEffect.uniaxialTests[i]
import odbMaterial
session.odbs[name].materials[name].hyperelastic.uniaxialTestData
session.odbs[name].materials[name].hyperfoam.uniaxialTestData
session.odbs[name].materials[name].lowDensityFoam\
.uniaxialCompressionTestData
session.odbs[name].materials[name].lowDensityFoam\
.uniaxialTensionTestData
session.odbs[name].materials[name].mullinsEffect.uniaxialTests[i]
```

UniaxialTestData(...)

This method creates a UniaxialTestData object.

Path

```
mdb.models[name].materials[name].hyperelastic.UniaxialTestData
mdb.models[name].materials[name].hyperfoam.UniaxialTestData
mdb.models[name].materials[name].lowDensityFoam.UniaxialTestData
mdb.models[name].materials[name].mullinsEffect.UniaxialTestData
session.odbs[name].materials[name].hyperelastic.UniaxialTestData
session.odbs[name].materials[name].hyperfoam.UniaxialTestData
session.odbs[name].materials[name].lowDensityFoam.UniaxialTestData
session.odbs[name].materials[name].mullinsEffect.UniaxialTestData
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

smoothing
`None` or an Int specifying the value for smoothing. If *smoothing*=`None`, no smoothing is employed. The default value is `None`.

lateralNominalStrain
A Boolean specifying whether to include lateral nominal strain. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic material model, the table data specify the following:

- Nominal stress, T_U.
- Nominal strain, ϵ_U.

For a hyperfoam material model, the table data specify the following:

- Nominal stress, T_L.
- Nominal strain, ϵ_U.
- Nominal lateral strain, $\epsilon_2 = \epsilon_3$. The default value is 0.

For a low-density foam material model, the table data specify the following:

- Nominal stress, T_U.
- Nominal strain, ϵ_U.
- Nominal strain rate, $\dot{\epsilon_U}$.

Return value

A UniaxialTestData object.

Exceptions

None.

`setValues(...)`

This method modifies the UniaxialTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *UniaxialTestData* method.

Return value

`None.`

Exceptions

None.

Members

The UniaxialTestData object has members with the same names and descriptions as the arguments to the *UniaxialTestData* method.

Corresponding analysis keywords

- **UNIAXIAL TEST DATA*

UserDefinedField object

The UserDefinedField object redefines field variables at a material point.

Access

```
import material
mdb.models[name].materials[name].userDefinedField
import odbMaterial
session.odbs[name].materials[name].userDefinedField
```

UserDefinedField()

This method defines a UserDefinedField object.

Path

```
mdb.models[name].materials[name].UserDefinedField
session.odbs[name].materials[name].UserDefinedField
```

Arguments

None.

Return value

A UserDefinedField object.

Exceptions

None.

Members

The UserDefinedField object has no members.

Corresponding analysis keywords

- **USER DEFINED FIELD*

UserMaterial object

The UserMaterial object defines material constants for use in subroutines *UMAT*, *UMATHT*, or *VUMAT*.

Access

```
import material
mdb.models[name].materials[name].userMaterial
import odbMaterial
session.odbs[name].materials[name].userMaterial
```

UserMaterial(...)

This method creates a UserMaterial object.

Path

```
mdb.models[name].materials[name].UserMaterial
session.odbs[name].materials[name].UserMaterial
```

Required arguments

None.

Optional arguments

type

A SymbolicConstant specifying the type of material behavior defined by the command. Possible values are MECHANICAL, THERMAL, and THERMOMECHANICAL. The default value is MECHANICAL.

unsymm

A Boolean specifying if the material stiffness matrix, $\partial \Delta \boldsymbol{\sigma} / \partial \Delta \boldsymbol{\varepsilon}$, is not symmetric or, when a thermal constitutive model is used, if $\partial \mathbf{f} / \partial (\partial \theta / \partial \mathbf{x})$ is not symmetric. The default value is OFF.

This argument is valid only for an Abaqus/Standard analysis.

mechanicalConstants

A sequence of Floats specifying the mechanical constants of the material. This argument is valid only when *type*=MECHANICAL or THERMOMECHANICAL. The default value is an empty sequence.

thermalConstants

A sequence of Floats specifying the thermal constants of the material. This argument is valid only when *type*=THERMAL or THERMOMECHANICAL. The default value is an empty sequence.

effmod

A Boolean specifying if effective bulk modulus and shear modulus are returned by user subroutine *VUMAT*. The default value is OFF.

This argument is valid only in an Abaqus/Explicit analysis.

hybridFormulation

A SymbolicConstant to specify the formulation of the hybrid elements with user subroutine *UMAT*. Possible values are TOTAL, INCREMENTAL, INCOMPRESSIBLE, and NONE,

which means that Abaqus/CAE will not specify a hybrid formulation and the Abaqus solver default will be used. The default value is NONE.

This argument is valid only in an Abaqus/Standard analysis.

Return value

A UserMaterial object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the UserMaterial object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *UserMaterial* method.

Return value

`None.`

Exceptions

RangeError.

Members

The UserMaterial object has members with the same names and descriptions as the arguments to the *UserMaterial* method.

Corresponding analysis keywords

- **USER MATERIAL*

UserOutputVariables object

The UserOutputVariables object specifies the number of user-defined output variables.

Access

```
import material
mdb.models[name].materials[name].userOutputVariables
import odbMaterial
session.odbs[name].materials[name].userOutputVariables
```

UserOutputVariables(...)

This method creates a UserOutputVariables object.

Path

```
mdb.models[name].materials[name].UserOutputVariables
session.odbs[name].materials[name].UserOutputVariables
```

Required arguments

None.

Optional arguments

n An Int specifying the number of user-defined variables required at each material point. The default value is 0.

Return value

A UserOutputVariables object.

Exceptions

RangeError.

setValues(...)

This method modifies the UserOutputVariables object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *UserOutputVariables* method.

Return value

`None.`

Exceptions

RangeError.

Members

The UserOutputVariables object has members with the same names and descriptions as the arguments to the *UserOutputVariables* method.

Corresponding analysis keywords

- **USER OUTPUT VARIABLES*

VelocityDependence object

The VelocityDependence object specifies the dependence of the permeability of a material on the velocity of fluid flow.

Access

```
import material
mdb.models[name].materials[name].permeability.velocityDependence
import odbMaterial
session.odbs[name].materials[name].permeability.velocityDependence
```

VelocityDependence(...)

This method creates a VelocityDependence object.

Path

```
mdb.models[name].materials[name].permeability.VelocityDependence
session.odbs[name].materials[name].permeability.VelocityDependence
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

None.

Table data

- β. Only $\beta > 0.0$ is allowed.
- Void ratio, e.

Return value

A VelocityDependence object.

Exceptions

RangeError.

setValues(...)

This method modifies the VelocityDependence object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VelocityDependence* method.

Return value

`None.`

Exceptions

RangeError.

Members

The VelocityDependence object has members with the same names and descriptions as the arguments to the *VelocityDependence* method.

Corresponding analysis keywords

- **PERMEABILITY*

Viscoelastic object

The Viscoelastic object specifies dissipative behavior for use with elasticity.

Access

```
import material
mdb.models[name].materials[name].viscoelastic
import odbMaterial
session.odbs[name].materials[name].viscoelastic
```

Viscoelastic(...)

This method creates a Viscoelastic object.

Path

```
mdb.models[name].materials[name].Viscoelastic
session.odbs[name].materials[name].Viscoelastic
```

Required arguments

domain

A SymbolicConstant specifying the domain definition. Possible values are:

- FREQUENCY, specifying a frequency domain. This domain is only available for an Abaqus/Standard analysis.
- TIME, specifying a time domain.

table

A sequence of sequences of Floats specifying the items described below.

Optional arguments

frequency

A SymbolicConstant specifying the frequency domain definition. This argument is required only when *domain*=FREQUENCY. Possible values are FORMULA, TABULAR, PRONY, CREEP_TEST_DATA, and RELAXATION_TEST_DATA. The default value is FORMULA.

type

A SymbolicConstant specifying the type. This argument is required only when *domain*=FREQUENCY and *frequency*=TABULAR. Possible values are ISOTROPIC and TRACTION. The default value is ISOTROPIC.

preload

A SymbolicConstant specifying the preload. This argument is required only when *domain*=FREQUENCY and *frequency*=TABULAR. Possible values are NONE, UNIAXIAL, VOLUMETRIC, and UNIAXIAL_VOLUMETRIC. The default value is NONE.

time

A SymbolicConstant specifying the time domain definition. This argument is required only when *domain*=TIME. Possible values are PRONY, CREEP_TEST_DATA, RELAXATION_TEST_DATA, and FREQUENCY_DATA. The default value is PRONY.

errtol

A Float specifying the allowable average root-mean-square error of the data points in the least-squares fit. The Float values correspond to percentages; for example, 0.01 is 1%. The default value is 0.01.

This argument is valid only when *time*=CREEP_TEST_DATA, RELAXATION_TEST_DATA or FREQUENCY_DATA; or only when *frequency*=CREEP_TEST_DATA or RELAXATION_TEST_DATA.

nmax

An Int specifying the maximum number of terms N in the Prony series. The maximum value is 13. The default value is 13.

This argument is valid only when *time*=CREEP_TEST_DATA, RELAXATION_TEST_DATA or FREQUENCY_DATA; or only when *frequency*=CREEP_TEST_DATA or RELAXATION_TEST_DATA.

volumetricTable

A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

Table data

If *frequency*=FORMULA, the table data for *table* specify the following:

- Real part of $g_1^*(g^*(\omega) = g_1^* f^{-a})$.
- Imaginary part of g_1^*.
- Value of a.
- Real part of $k_1^*(k^*(\omega) = k_1^* f^{-b})$. If the material is incompressible, this value is ignored.
- Imaginary part of k_1^*. If the material is incompressible, this value is ignored.
- Value of b. If the material is incompressible, this value is ignored.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=NONE, or *time*=FREQUENCY_DATA the table data for *table* specify the following:

- Real part of $\omega g^*(\omega \Re(g^*) = G_\ell / G_\infty)$.
- Imaginary part of $\omega g^*(\omega \Im(g^*) = 1 - G_s / G_\infty)$.
- Real part of $\omega k^*(\omega \Re(k^*) = \mathrm{K}_\ell / \mathrm{K}_\infty)$. If the material is incompressible, this value is ignored.
- Imaginary part of $\omega k^*(\omega \Im(k^*) = 1 - \mathrm{K}_s / \mathrm{K}_\infty)$. If the material is incompressible, this value is ignored.
- Frequency f in cycles per time.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=UNIAXIAL the table data for *table* specify the following:

- Loss modulus.
- Storage modulus.
- Frequency.
- Uniaxial strain.

If *frequency*=TABULAR and *type*=TRACTION and *preload*=NONE the table data for *table* specify the following:

- Normalized loss modulus.
- Normalized shear modulus.

- Frequency.

If *frequency*=TABULAR and *type*=TRACTION and *preload*=UNIAXIAL or *preload*=UNIAXIAL_VOLUMETRIC the table data for *table* specify the following:

- Loss modulus.
- Storage modulus.
- Frequency.
- Closure.

If *time*=PRONY or *frequency*=PRONY, the table data for *table* specify the following:

- $\overline{g}_1^P$, the modulus ratio in the first term in the Prony series expansion of the shear relaxation modulus.
- $\overline{k}_1^P$, the modulus ratio in the first term in the Prony series expansion of the bulk relaxation modulus.
- τ_1, the relaxation time for the first term in the Prony series expansion.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=VOLUMETRIC or *preload*=UNIAXIAL_VOLUMETRIC the table data for *volumetricTable* specify the following:

- Loss modulus.
- Storage modulus.
- Frequency.
- Volume ratio.

Return value

A Viscoelastic object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Viscoelastic object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Viscoelastic* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Viscoelastic object has members with the same names and descriptions as the arguments to the *Viscoelastic* method.

In addition, the Viscoelastic object can have the following members:

combinedTestData	A *CombinedTestData* object.
shearTestData	A *ShearTestData* object.
trs	A *Trs* object.
volumetricTestData	A *VolumetricTestData* object.

Corresponding analysis keywords

- **VISCOELASTIC*

Viscosity object

The Viscosity object specifies mechanical viscosity.

Access

```
import material
mdb.models[name].materials[name].viscosity
import odbMaterial
session.odbs[name].materials[name].viscosity
```

Viscosity(...)

This method creates a Viscosity object.

Path

```
mdb.models[name].materials[name].Viscosity
session.odbs[name].materials[name].Viscosity
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

type A SymbolicConstant specifying the type of viscosity. The default value is NEWTONIAN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=NEWTONIAN, the table data specify the following:

- Viscosity, k.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Viscosity object.

Exceptions

RangeError.

setValues(...)

This method modifies the Viscosity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Viscosity* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Viscosity object has members with the same names and descriptions as the arguments to the *Viscosity* method.

In addition, the Viscosity object can have the following member:

trs A *Trs* object.

Corresponding analysis keywords

- **VISCOSITY*

Viscous object

The Viscous object specifies the viscous properties for a two-layer viscoplastic material model.

Access

```
import material
mdb.models[name].materials[name].viscous
import odbMaterial
session.odbs[name].materials[name].viscous
```

Viscous(...)

This method creates a Viscous object.

Path

```
mdb.models[name].materials[name].Viscous
session.odbs[name].materials[name].Viscous
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

law A SymbolicConstant specifying the creep law. Possible values are STRAIN, TIME, USER, ANAND, DARVEAUX, DOUBLE_POWER, POWER_LAW, and TIME_POWER_LAW. The default value is STRAIN.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

time A SymbolicConstant specifying the time interval for relevant laws. Possible values are CREEP and TOTAL. The default value is TOTAL.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- f.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=USER, the table data specify the following:

- f.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=ANAND, the table data specify the following:

- s_1.
- $\frac{Q}{R}$.
- A.
- ξ.
- m.
- A_0.
- $\hat{s}$.
- n.
- a.
- S_2.
- S_3.
- A_1.
- A_2.
- A_3.
- A_4.
- f.

If *law*=DARVEAUX, the table data specify the following:

- C_{ss}.
- $\frac{Q}{R}$.
- α.
- n.
- ϵ_T.
- B.
- f.

If *law*=DOUBLE_POWER, the table data specify the following:

- A_1.
- B_1.
- C_1.
- A_2.
- B_2.
- C_2.
- σ_0.
- f.

If *law*=POWER_LAW or *law*=TIME_POWER_LAW, the table data specify the following:

- q_0.
- n.
- m.
- $\dot{\varepsilon}_0$.
- f.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Viscous object.

Exceptions

None.

`setValues(...)`

This method modifies the Viscous object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Viscous* method.

Return value

`None.`

Exceptions

None.

Members

The Viscous object has members with the same names and descriptions as the arguments to the *Viscous* method.

In addition, the Viscous object can have the following member:

potential
A *Potential* object.

Corresponding analysis keywords

- **VISCOUS*

VoidNucleation object

The VoidNucleation object defines the nucleation of voids in a porous material.

Access

```
import material
mdb.models[name].materials[name].porousMetalPlasticity.voidNucleation
import odbMaterial
session.odbs[name].materials[name].porousMetalPlasticity\
.voidNucleation
```

VoidNucleation(...)

This method creates a VoidNucleation object.

Path

```
mdb.models[name].materials[name].porousMetalPlasticity.VoidNucleation
session.odbs[name].materials[name].porousMetalPlasticity\
.VoidNucleation
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- ε_N, the mean value of the nucleation-strain normal distribution.
- s_N, the standard deviation of the nucleation-strain normal distribution.
- f_N, the volume fraction of nucleating voids.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A VoidNucleation object.

Exceptions

RangeError.

setValues(...)

This method modifies the VoidNucleation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VoidNucleation* method.

Return value

`None.`

Exceptions

RangeError.

Members

The VoidNucleation object has members with the same names and descriptions as the arguments to the *VoidNucleation* method.

Corresponding analysis keywords

- **VOID NUCLEATION*

VolumetricTestData object

The VolumetricTestData object provides volumetric test data.

Access

```
import material
mdb.models[name].materials[name].hyperelastic.volumetricTestData
mdb.models[name].materials[name].hyperfoam.volumetricTestData
mdb.models[name].materials[name].viscoelastic.volumetricTestData
import odbMaterial
session.odbs[name].materials[name].hyperelastic.volumetricTestData
session.odbs[name].materials[name].hyperfoam.volumetricTestData
session.odbs[name].materials[name].viscoelastic.volumetricTestData
```

VolumetricTestData(...)

This method creates a VolumetricTestData object.

Path

```
mdb.models[name].materials[name].hyperelastic.VolumetricTestData
mdb.models[name].materials[name].hyperfoam.VolumetricTestData
mdb.models[name].materials[name].viscoelastic.VolumetricTestData
session.odbs[name].materials[name].hyperelastic.VolumetricTestData
session.odbs[name].materials[name].hyperfoam.VolumetricTestData
session.odbs[name].materials[name].viscoelastic.VolumetricTestData
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

volinf
`None` or a Float specifying a normalized volumetric value that depends on the value of the *time* member of the *Viscoelastic* object. The default value is `None`.

If *time*=RELAXATION_TEST_DATA, *volinf* specifies the value of the long-term, normalized volumetric modulus, $k_R(\infty)$.

If *time*=CREEP_TEST_DATA, *volinf* specifies the value of the long-term, normalized volumetric compliance, $j_K(\infty)$.

This argument is valid only for a viscoelastic material model.

smoothing
`None` or an Int specifying the value for smoothing. If *smoothing*=`None`, no smoothing is employed. The default value is `None`.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic or hyperfoam material model, the table data specify the following:

- Pressure, p.
- Volume ratio, J (current volume/original volume).

For a viscoelastic material model, the values depend on the value of the *time* member of the *Viscoelastic* object.

If *time*=RELAXATION_TEST_DATA, the table data specify the following:

- Normalized volumetric (bulk) modulus $k_R(t), \quad (0 \le k_R(t) \le 1)$.
- Time $t(t > 0)$.

If *time*=CREEP_TEST_DATA, the table data specify the following:

- Normalized volumetric (bulk) compliance $j_K(t), \quad (j_K(t) \ge 1)$.
- Time $t(t > 0)$.

Return value

A VolumetricTestData object.

Exceptions

None.

setValues(...)

This method modifies the VolumetricTestData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *VolumetricTestData* method.

Return value

`None.`

Exceptions

None.

Members

The VolumetricTestData object has members with the same names and descriptions as the arguments to the *VolumetricTestData* method.

Corresponding analysis keywords

- **VOLUMETRIC TEST DATA*

evaluateMaterial

This command evaluates the behavior of a hyperelastic material under standard test conditions.

Access

```
from material import evaluateMaterial
```

evaluateMaterial

This method evaluates the behavior of a hyperelastic material under standard test conditions.

Path

```
evaluateMaterial
```

Required arguments

material A Material object.

simulationName A String specifying the name to be used for the material evaluation simulation.

Optional arguments



Note:

Although *uniaxialStrainRange*, *biaxialStrainRange*, *planarStrainRange*, *volumeRatioRange*, and *simpleShearStrainRange* are optional arguments, at least one of them must be specified with a nonzero value.

dataSource A SymbolicConstant specifying whether test data or coefficients should be used for the material definition in the unit element tests. Possible values are TEST_DATA or COEFFICIENTS.

strainEnergyPotentials A sequence of SymbolicConstants specifying for which material models the material is to be evaluated. Possible values are POLY_N1, POLY_N2, POLY_N3, POLY_N4, POLY_N5, POLY_N6, OGDEN_N1, OGDEN_N2, OGDEN_N3, OGDEN_N4, OGDEN_N5, OGDEN_N6, REDUCED_POLY_N1, REDUCED_POLY_N2, REDUCED_POLY_N3, REDUCED_POLY_N4, REDUCED_POLY_N5, REDUCED_POLY_N6, ARRUDA_BOYCE, VAN_DER_WAALS, YEOH, MOONEY_RIVLIN, and NEO_HOOKE.



Note:

The options POLY_N3, POLY_N4, POLY_N5, and POLY_N6 are valid only if the material was defined by providing coefficients of the strain energy potential.

marlowData
`None` or a sequence of SymbolicConstants specifying the types of test data to be included in the material definition of the Marlow material that is being evaluated. Possible values are UNIAXIAL, BIAXIAL, PLANAR, or VOLUMETRIC. The default value is `None`.

marlowDataType
`None` or a SymbolicConstant specifying the input data type for the Marlow material model. Possible values are TENSION, COMPRESSION, or BOTH.

testDataTypes
A sequence of SymbolicConstants specifying the types of test data to be included in the material definition of the material being evaluated. Possible values are UNIAXIAL, BIAXIAL, PLANAR, and VOLUMETRIC.

uniaxialStrainRange
A tuple of Floats specifying minimum and maximum nominal strains to be applied in the uniaxial tension test.

biaxialStrainRange
A tuple of Floats specifying the minimum and maximum nominal strains to be applied in the biaxial tension test.

planarStrainRange
A tuple of Floats specifying the minimum and maximum nominal strains to be applied in the planar test. The planar test is equivalent to a pure shear test.

volumeRatioRange
A tuple of Floats specifying the minimum and maximum compressive volume ratio.

simpleShearStrainRange
A tuple of Floats specifying the minimum and maximum nominal strains to be applied in the simple shear test.

viscoDataSource
`None` or a SymbolicConstant specifying whether test data or coefficients should be used for the viscoelastic material definition in the element tests. Possible values are TEST_DATA or COEFFICIENTS. The default value is `None`.

viscoTestDataTypes
`None` or a sequence of SymbolicConstants specifying the types of test data to be included in the material definition of the viscoelastic material being evaluated. Possible values are UNIAXIAL, BIAXIAL, PLANAR, or VOLUMETRIC. The default value is `None`.

relaxationTime
`None` or a Float specifying the time period for the stress relaxation response mode. The default value is `None`.

creepTime
`None` or a Float specifying the time period for the creep response mode. The default value is `None`.

Return value
`None`.

Exceptions

If *dataSource*=TEST_DATA and *strainEnergyPotentials* contains POLY_N3, POLY_N4, POLY_N5, or POLY_N6:	MaterialEvaluationError: POLY_N3, POLY_N4, POLY_N5, or POLY_N6

	not allowed for *dataSource*=TEST_DATA.
If the material evaluation failed:	MaterialEvaluationError: material evaluation failed, see *path to data file*.
If the material type of the material to be evaluated is not hyperelastic:	MaterialEvaluationError: Material evaluation is currently supported only for hyperelastic materials.

Mdb commands

Mdb commands are used to create and upgrade an Abaqus model database that stores models and analysis controls.

In this section:

- *Mdb object*
- *Mdb commands*

Mdb object

The Mdb object is the high-level Abaqus model database. A model database stores models and analysis controls.

Access

```
mdb
```

Mdb(...)

This constructor creates an empty Mdb object.

Path

```
Mdb
```

Required arguments

None.

Optional arguments

pathName A String specifying the path to be used when the model database is saved to a file. If you do not provide a file extension, `.cae` is appended automatically to the path. The default value is an empty string.

Return value

A Mdb object.

Exceptions

None.

importDxf(...)

This method creates a *ConstrainedSketch* object from a file containing dxf-format (AutoCAD) geometry. Only a limited number of entities are supported. This format should be used only if no other formats are available.

Path

```
importDxf
```

Required arguments

fileName A String specifying the path to the dxf file to open.

Optional arguments

None.

Return value

A Mdb object.

Exceptions

None.

openMdb(...)

This method opens an existing model database file.

Path

```
openMdb
```

Required arguments

pathName A String specifying the path to the model database file to open. If you do not provide a file extension, Abaqus/CAE attempts to open the file with `.cae` appended to the path.

Optional arguments

None.

Return value

An Mdb object.

Exceptions

If the file is an invalid model database:	MdbError: invalid model database.
If the file contains a model database from an Abaqus release other than the Abaqus release you are currently running:	MdbError: incompatible release number, expected *<Abaqus release>*, got *<earlier or later Abaqus release>*
If the model database file is already opened in write mode:	MdbError: cannot open file: May be in use by another CAE session
If the command fails to open the model database file for reasons not mentioned above:	MdbError: cannot open file...

openAcis(...)

This method creates an *AcisFile* object from a file containing ACIS-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openAcis
```

Required arguments

fileName A String specifying the path to the ACIS file to open.

Optional arguments

scaleFromFile A Boolean specifying whether to scale, rotate, and translate the part using the transform read from the ACIS file. The default value is OFF.

Return value

An AcisFile object.

Exceptions

File is from a newer version of ACIS than the CAE kernel.	Texterror: ACIS File version exceeds Kernel.
The data in the ACIS file are corrupted.	Texterror: Failed to read ACIS file.

openCatia(...)

This method creates an *AcisFile* object from a file containing CATIA V5–format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openCatia
```

Required arguments

fileName A String specifying the path to the CATIA file to open.

Optional arguments

topology A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID, SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

convertUnits A SymbolicConstant specifying whether the original units should be retained. Possible values are ON and OFF. The default value is OFF.

combineBodies A Boolean specifying whether to combine the bodies in the CATPart file. If the bodies to be combined touch or overlap, invalid entities would result. For CATProduct files, this option will be ignored.

Return value

An AcisFile object.

Exceptions

None.

openEnf(...)

This method creates an*AcisFile* object from a file containing Elysium Neutral File-format geometry that was created by CATIA V5, I-DEAS, or Pro/ENGINEER. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openEnf
```

Required arguments

fileName
A String specifying the path to the Elysium Neutral File that was created by I-DEAS, Pro/ENGINEER, or CATIA V5.

fileType
A String specifying the type of CAD system that created the file. Possible values are “ideas”, “proe”, or “catiav5”.

Optional arguments

topology
A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID, SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

convertUnits
A Boolean specifying if the dimensions of the part should be converted to millimeters. The default value is OFF.

Return value

An AcisFile object.

Exceptions

None.

openIges(...)

This method creates an *AcisFile* object from a file containing IGES-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openIges
```

Required arguments

fileName
A String specifying the path to the IGES file to open.

Optional arguments

trimCurve

A SymbolicConstant specifying the method used to define the trim curves that bound parametric surfaces. Possible values are:

- DEFAULT, use either of the following as specified by the contents of the IGES file.
- PARAMETRIC_DATA, use the parameter space of the surface being trimmed.
- THREED_DATA, use real space—the coordinate system of the part along with an indication that the trim curve lies on the parametric surface.

The default value is DEFAULT.

scaleFromFile

A SymbolicConstant specifying whether the imported geometry needs to be scaled using the units information available in the IGES file. Possible values are ON and OFF. The default value is OFF. When the argument is set to ON, the geometry is scaled to millimeters with respect to the unit system specified in the IGES file.

msbo

A Boolean specifying if the IGES file contains MSBO (Manifold Solid B-Rep Object) entities. The default value is False.

includedLayers

A sequence of Ints specifying the levels or layers of entities that will be translated from the IGES file to build the part. The default is to include all the layers.

topology

A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID, SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

uniteWires

A SymbolicConstant specifying whether the imported wires need to be united or not. Possible values are ON andOFF. The default value is ON. When importing a sketch, this value is set to OFF.

Return value

An AcisFile object.

Exceptions

The data in the IGES file are corrupted.	Texterror: Failed to read IGES file.

openParasolid(...)

This method creates an*AcisFile* object from a file containing Parasolid-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openParasolid
```

Required arguments

fileName A String specifying the path to the Parasolid file to open.

Optional arguments

topology A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID, SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

Return value

An AcisFile object.

Exceptions

None.

openStep(...)

This method creates an *AcisFile* object from a file containing STEP-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openStep
```

Required arguments

fileName A String specifying the path to the STEP file to open.

Optional arguments

scale A Float specifying the scaling factor to apply to the imported geometric entities. The default value is 1.0.

Return value

An AcisFile object.

Exceptions

The data in the STEP file are corrupted. Texterror: Failed to read STEP file.

openVda(...)

This method creates an *AcisFile* object from a file containing VDA-FS-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openVda
```

Required arguments

fileName A String specifying the path to the VDA-FS file to open.

Optional arguments

None.

Return value

An AcisFile object.

Exceptions

The data in the VDA-FS file are corrupted. Texterror: Failed to read VDA file.

openSolidworks(...)

This method creates an *AcisFile* object from a file containing Solidworks format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

```
openSolidworks
```

Required arguments

fileName A String specifying the path to the Solidworks file to open.

Optional arguments

topology A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID, SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid entity. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity, not as a solid entity. The default value is SOLID.

Return value

An AcisFile object.

Exceptions

The data in the Solidworks file are corrupted. Texterror: Failed to read Solidworks file.

close()

This method closes an open Mdb object but does not save the Mdb object to disk. After closing the Mdb object, this method creates a new unnamed empty Mdb object.

Arguments

None.

Return value

`None.`

Exceptions

None.

save()

This method saves an Mdb object to disk at the location specified by *pathName* (*pathName* is a member of the Mdb object).

Arguments

None.

Return value

`None.`

Exceptions

If *pathName* is empty:	MdbError: cannot save file: pathname member is empty
If *pathName* is `abaqus.cae`:	MdbError: “abaqus.cae” is an invalid CAE filename.
If the command fails to save the Mdb object to disk for reasons not mentioned above:	MdbError: cannot save file...

saveAs(...)

This method saves an Mdb object to disk at the specified location.

Required arguments

pathName	A String specifying the path to be used when the model database is saved to a file. If you do not provide a file extension, `.cae` is appended automatically to the path.

Optional arguments

None.

Return value
`None.`

Exceptions

If *pathName* is `abaqus.cae`:	MdbError: “abaqus.cae” is an invalid CAE filename.
If the command fails to save the Mdb object to disk for reasons not mentioned above:	MdbError: cannot save file...

`openAuxMdb(...)`

This method opens an auxiliary Mdb object on the disk at the specified location. This enables models from the auxiliary Mdb object to be copied into the current Mdb.

Required arguments

pathName	A String specifying the path to the auxiliary Mdb which is to be opened. If you do not provide a file extension, `.cae` is appended automatically to the path.

Optional arguments

None.

Return value
`None.`

Exceptions

If the file is an invalid model database:	MdbError: invalid model database.
If the file contains a model database from an Abaqus release other than the Abaqus release you are currently running:	MdbError: incompatible release number, expected *<Abaqus release>*, got *<earlier or later Abaqus release>*.
If the command fails to open the model database file for reasons not mentioned above:	MdbError: cannot open file...

`closeAuxMdb()`

This method closes the auxiliary Mdb which had been opened earlier using the `openAuxMdb` command.

Arguments

None.

Return value
`None.`

Exceptions

If the auxiliary Mdb was not opened earlier.	MdbError: The auxiliary Mdb was not opened..

`getAuxMdbModelNames()`

This method returns a list of model names present in the auxiliary Mdb which had been opened earlier using the `openAuxMdb` command.

Arguments

None.

Return value

A list of model names present in the auxiliaryMdb

Exceptions

If the auxiliary Mdb was not opened earlier.	MdbError: The auxiliary Mdb was not opened..

`copyAuxMdbModel(...)`

This method copies a specified model from the auxiliary Mdb which had been opened earlier using the `openAuxMdb` command.

Required arguments

fromName	A String specifying the model name in the auxiliary Mdb which is to be copied.

Optional arguments

toName	A String specifying the name to be given to the model after it is copied into the Mdb. If this argument is not specified *toName* is assumed to be the same as *fromName*. If a model with name *toName* already exists in Mdb, it is overwritten.

Return value

`None.`

Exceptions

If the auxiliary Mdb was not opened earlier.	MdbError: The auxiliary Mdb was not opened.
If the model fromName does not exist in the auxiliary Mdb.	KeyError: fromName does not exist.

Members

The Mdb object has members with the same names and descriptions as the arguments to the *Mdb* method.

In addition, the Mdb object can have the following members:

version
An Int specifying the release number of the Mdb object in memory.

lastChangedCount
A Float specifying the value of a counter associated with the Mdb object. The counter indicates when the Mdb object was last changed.

jobs
A repository of *Job* objects.

adaptivityProcesses
A repository of *AdaptivityProcess* objects.

coexecutions
A repository of *Coexecution* objects.

optimizationProcesses
A repository of *OptimizationProcess* objects.

meshEditOptions
A *MeshEditOptions* object specifying the undo/redo behavior when editing meshes on parts or part instances.

models
A repository of *Model* objects.

customData
A *RepositorySupport* object.

annotations
A repository of *Annotation* objects.

Mdb commands

The following command upgrades a model database (`.cae`) to the current release and writes the upgraded model database to a new file.

upgradeMdb

This method upgrades an existing Mdb object to the current release and writes the upgraded version of the Mdb object to a file. In addition, Abaqus/CAE writes information about the status of the upgrade to the log file (*upgradedMdbPath*`.log`).

Path

upgradeMdb

Required arguments

existingMdbPath A String specifying the path to the file containing the model database to be upgraded.

upgradedMdbPath A String specifying the path to the file that will contain the upgraded model database.

Optional arguments

None.

Return value

`None.`

Exceptions

If the model database upgrade fails: MdbError: cannot convert file

CombineOptResults

This method combines the results from existing ODB files for each optimization cycle and writes a merged ODB file.

Path

CombineOptResults

Required arguments

optResultLocation A String specifying the path to the folder in which optimization results are present.

Optional arguments

optIter

A Symbolic Constant to specify the optimization cycles from which the results should be merged. The possible values are INITIAL_AND_LAST, NONE, ALL, LAST, EVERY_NCYCLES, SPECIFY.

The default value is INITIAL_AND_LAST.

nValues

An Int or a tuple of Ints specifying the optimization cycles from which the results should be merged. This argument is used only when EVERY_NCYCLES or SPECIFY is selected for *optIter*.

The default value is ALL.

models

A tuple of strings specifying the list of *models* for which the merging of results is performed.

The default value is ALL.

steps

A tuple of strings specifying the list of steps from the selected *models* to be included in the odb merge.

The default value is ALL.

analysisFieldVariables

A tuple of strings specifying the list of *analysisFieldVariables* to be included in the odb merge.

The default value is ALL.

includeResultsFrom

A Symbolic Constant to specify the target odb to which the results will be merged. The possible values are ORIGINAL_MODEL, FIRST or LAST.

The default value is FIRST.

originalModel

A String to specify the path of target odb if *includeResultsFrom* is set to ORIGINAL_MODEL.

Mesh commands

Mesh commands are used to mesh part instances and regions.
Mesh commands are also used to assign element sizes, element types, and mesh control parameters.

In this section:

- *Assembly object*
- *Part object*
- *ElemType object*
- *MeshEdge object*
- *MeshEdgeArray object*
- *MeshElement object*
- *MeshElementArray object*
- *MesherOptions object*
- *MeshFace object*
- *MeshFaceArray object*
- *MeshNode object*
- *MeshNodeArray object*
- *MeshStats object*

Assembly object

The following commands operate on Assembly objects. For more information about the Assembly object, see *Assembly object*.

Access

```
import mesh
```

assignStackDirection(...)

This method assigns a stack direction to geometric cells. The stack direction will be used to orient the elements during mesh generation.

Required arguments

cells
A sequence of *Cell* objects specifying regions where to assign the stack direction.

referenceRegion
A *Face* object specifying the top side of the stack direction.

Optional arguments

None.

Return value

None.

Exceptions

None.

associateMeshWithGeometry(...)

This method associates a geometric entity with mesh entities that are either orphan elements, bounds orphan elements, or were created using the bottom-up meshing technique.

Required arguments

geometricEntity
A *Cell* , a *Face*, an *Edge*, or a *Vertex* object specifying geometric entity to be associated with one or more mesh entities.

If the geometric entity is a *Cell* object then the argument *elements* must be specified.

If the geometric entity is a *Face* object then the argument *elemFaces* must be specified.

If the geometric entity is an *Edge* object then the argument *elemEdges* must be specified.

If the geometric entity is a *Vertex* object then the argument *node* must be specified.

Optional arguments

elements
A sequence of *MeshElement* objects specifying the elements to be associated with the geometric cell.

elemFaces
A sequence of *MeshFace* objects specifying the element faces to be associated with the geometric face.

elemEdges
A sequence of *MeshEdge* objects specifying the element edges to be associated with the geometric edge.

node
A *MeshNode* object specifying the mesh node to be associated with the geometric vertex.

Return value

`None.`

Exceptions

None.

`createVirtualTopology(...)`

This method creates a virtual topology feature by automatically merging faces and edges based on a set of geometric parameters. The edges and vertices that are being merged will be ignored during mesh generation.

Required arguments

regions
A sequence of *Face* objects or *PartInstance* objects specifying the domain to search for geometric entities that need to be merged. Entities identified as candidates to be merged may be merged with entities from outside the specified region.

Optional arguments

mergeShortEdges
A Boolean specifying whether to merge short edges. The default value is False.

shortEdgeThreshold
A Float specifying a threshold that determines which edges are considered to be short. These edges are the candidate entities to be merged. This argument is a required argument if the argument *mergeShortEdges* equals True and it is ignored if the argument *mergeShortEdges* equals False.

mergeSmallFaces
A Boolean specifying whether to merge faces with small area. The default value is False.

smallFaceAreaThreshold
A Float specifying a threshold that determines which faces are considered to have a small area. These faces are the candidate entities to be merged. This argument is a required argument if the argument *mergeSmallFaces* equals True and it is ignored if the argument *mergeSmallFaces* equals False.

mergeSliverFaces
A Boolean specifying whether to merge faces with high aspect ratio. The default value is False.

faceAspectRatioThreshold
A Float specifying a threshold that determines which faces are considered to have high aspect ratio. These faces are candidate entities to be merged. This argument is a required argument if the argument *mergeSliverFaces* equals True and it is ignored if the argument *mergeSliverFaces* equals False.

mergeSmallAngleFaces
A Boolean specifying whether to merge faces that have a sharp corner angle. The default value is False.

smallFaceCornerAngleThreshold
A Float specifying a threshold that determines which face corner angles are considered to be small. These faces will be candidate entities to be merged. This argument is a required argument if the argument *mergeSmallAngleFaces* equals True and it is ignored if the argument *mergeSmallAngleFaces* equals False.

mergeThinStairFaces
A Boolean specifying whether to merge faces that represent a thin stair-like feature. The default value is False.

thinStairFaceThreshold
A Float specifying a threshold that determines which faces representing small stair-like features are considered thin. These faces will be candidate entities to be merged. This argument is required if the argument *mergeThinStairFaces* is True and it is ignored if *mergeThinStairFaces* is False.

ignoreRedundantEntities
A Boolean specifying whether to abstract away redundant edges and vertices. The default value is False.

cornerAngleTolerance
A Float specifying the angle deviation from 180 degrees at a vertex or at an edge such that the two edges radiating from the vertex or the two faces bounded by the edge can be merged. The default value is 30.0 degrees.

applyBlendControls
A Boolean specifying whether to verify that blend faces can be merged with neighboring faces. If *applyBlendControls* is True then all faces that have angle larger than *blendSubtendedAngleTolerance* and a radius smaller than *blendRadiusTolerance* will not be merged with neighboring faces unless the neighboring faces are also blend faces with similar geometric characteristics. The default value is False.

blendSubtendedAngleTolerance
A Float specifying the largest subtended angle of blend faces that can be merged with neighboring faces. This argument is a required argument if the argument *applyBlendControls* equals True and it is ignored if the argument *applyBlendControls* equals False.

blendRadiusTolerance
A Float specifying the smallest radius of curvature of blend faces that can be merged with neighboring faces. This argument is a required argument if the argument *applyBlendControls* equals True and it is ignored if the argument *applyBlendControls* equals False.

Return value

A Feature object.

Exceptions

None.

deleteBoundaryLayerControls(...)

This method deletes the control parameters for boundary layer mesh for all the specified regions.

Required arguments

regions A sequence of *Cell* objects specifying the regions for which to set the boundary layer mesh control parameters.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteMesh(...)

This method deletes a subset of the mesh that contains the native elements from the given part instances or regions.

Required arguments

regions A sequence of *PartInstance* objects or *Region* objects specifying the part instances or regions from where the native mesh is to be deleted.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteMeshAssociationWithGeometry(...)

This method deletes the association of geometric entities with mesh entities.

Required arguments

geometricEntities
A sequence of *Cell* objects, *Face* objects, *Edge* objects, or *Vertex* objects specifying the geometric entities that will be disassociated from the mesh.

Optional arguments

addBoundingEntities
A Boolean specifying whether the mesh will also be disassociated from the geometric entities that bounds the given *geometricEntities*. For example, if the argument *geometricEntities* contains a face, this boolean indicates whether the edges and vertices that bound the face will also be disassociated from the mesh. The default value is False.

Return value

`None.`

Exceptions

None.

`deletePreviewMesh()`

This method deletes all boundary meshes in the assembly. See the *boundaryPreview* argument of `generateMesh` for information about generating boundary meshes.

Arguments

None.

Return value

`None.`

Exceptions

None.

`deleteSeeds(...)`

This method deletes the global edge seeds from the given part instances or deletes the local edge seeds from the given edges.

Required arguments

regions
A sequence of *PartInstance* objects or Edge objects specifying the part instances or edges from which the seeds are to be deleted.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

generateBottomUpExtrudedMesh(...)

This method generates solid elements by extruding a 2D mesh along a vector, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell
A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native part instances.

numberOfLayers
An Int specifying the number of layers to be generated along the extrusion vector.

extrudeVector
A sequence of sequences of Floats specifying the start point and end point of a vector. Each point is defined by a tuple of three coordinates indicating its position. The direction of the mesh extrusion operation is from the first point to the second point.

Optional arguments

At least one of the three optional "SourceSide" arguments must be specified.

geometrySourceSide
A *Region* of *Face* objects specifying the geometric domain to be used as the source for the extrude meshing operation.

elemFacesSourceSide
A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the extrude meshing operation.

elemSourceSide
A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the extrude meshing operation.

depth
A Float specifying the distance of the mesh extrusion. If unspecified, the vector length of the *extrudeVector* argument is assumed.

targetSide
A datum plane, a sequence of *Face* objects, a sequence of *MeshFace* objects, or a sequence of 2D *MeshElement* objects specifying the target of the extrude meshing operation. If specified, this argument overrides the *depth* argument, and all points on the source will be extruded in the direction of the extrusion vector until meeting the target.

biasRatio
A Float specifying a ratio of the element size in the extrusion direction between the source and the target sides of the extrusion. The default is 1.0, meaning no bias.

extendElementSets
A Boolean specifying whether existing element sets that include source elements will be extended to also include extruded elements. This argument is ignored for native part instances. The default value is False.

Return value

`None.`

Exceptions

None.

generateBottomUpSweptMesh(...)

This method generates solid elements by sweeping a 2D mesh, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell
A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native part instances.

Optional arguments

At least one of the following three arguments must be specified: *geometrySourceSide*, *elemFacesSourceSide*, or *elemSourceSide*.

In addition one of the following two sets of arguments must also be specified:

- One of the following three arguments: *geometryConnectingSides*, *elemFacesConnectingSides*, or *elemConnectingSides*. In this case the targetSide is an optional argument.
- *targetSide* and *numberOfLayers*.

geometrySourceSide
A *Region* of *Face* objects specifying the geometric domain to be used as the source for the sweep meshing operation.

elemFacesSourceSide
A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the sweep meshing operation.

elemSourceSide
A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the sweep meshing operation.

geometryConnectingSides
A *Region* of *Face* objects specifying the connecting sides of the sweep meshing operation.

elemFacesConnectingSides
A sequence of *MeshFace* objects specifying connecting sides of the sweep meshing operation.

elemConnectingSides
A sequence of 2D *MeshElement* objects specifying connecting sides of the sweep meshing operation.

targetSide
A *Face* object specifying the target side of the sweep meshing operation.

numberOfLayers
An Int specifying the number of layers to be generated along the sweep direction.

extendElementSets
A Boolean specifying whether existing element sets that include source elements will be extended to also include swept elements. This argument is ignored for native part instances. The default value is False.

Return value
None.

Exceptions
None.

generateBottomUpRevolvedMesh(...)

This method generates solid elements by revolving a 2D mesh around an axis, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native part instances.

numberOfLayers An Int specifying the number of layers of elements to be generated around the axis of revolution.

axisOfRevolution A sequence of sequences of Floats specifying the two points of the vector that describes the axis of revolution. Each point is defined by a tuple of three coordinates indicating its position. The direction of the axis of revolution is from the first point to the second point. The orientation of the revolution operation follows the right-hand-rule about the axis of revolution.

angleOfRevolution A Float specifying the angle of revolution.

Optional arguments

At least one of the three optional "SourceSide" arguments must be specified.

geometrySourceSide A *Region* of *Face* objects specifying the geometric domain to be used as the source for the revolve meshing operation.

elemFacesSourceSide A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the revolve meshing operation.

elemSourceSide A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the revolve meshing operation.

extendElementSets A Boolean specifying whether existing element sets that include source elements will be extended to also include extruded elements. This argument is ignored for native part instances. The default value is False.

Return value
None.

Exceptions
None.

generateMesh(...)

This method generates a mesh in the given part instances or regions.

Required arguments

None.

Optional arguments

regions A sequence of *PartInstance* objects or *Region* objects specifying the part instances or regions where the mesh is to be generated.

seedConstraintOverride A Boolean specifying whether mesh generation is allowed to modify seed constraints. The default value is OFF.

meshTechniqueOverride A Boolean specifying whether mesh generation is allowed to modify the existing mesh techniques so that a compatible mesh can be generated. The default value is OFF.

boundaryPreview A Boolean specifying whether the generated mesh should be a boundary mesh. This option will only have an effect if any of the specified regions are to be meshed with tetrahedral elements or using the bottom-up technique with hexahedral or wedge elements. The default value is OFF.

boundaryMeshOverride A Boolean specifying whether mesh generation is allowed to modify an existing boundary preview mesh. This option will only have an effect if any of the specified regions are to be meshed with tetrahedral elements and a boundary preview mesh already exists. The default value is OFF.

Return value

`None.`

Exceptions

None.

getEdgeSeeds(...)

This method returns an edge seed parameter for a specified edge of an assembly.

Required arguments

edge An *Edge* object specifying the edge to be queried.

attribute A SymbolicConstant specifying the type of edge seed attribute to return. Possible values are:

- EDGE_SEEDING_METHOD
- BIAS_METHOD
- NUMBER

- AVERAGE_SIZE
- DEVIATION_FACTOR
- MIN_SIZE_FACTOR
- BIAS_RATIO
- BIAS_MIN_SIZE
- BIAS_MAX_SIZE
- VERTEX_ADJ_TO_SMALLEST_ELEM
- SMALLEST_ELEM_LOCATION
- CONSTRAINT

The return value is dependent on the *attribute* argument.

- If *attribute*=EDGE_SEEDING_METHOD, the return value is a SymbolicConstant specifying the edge seeding method used to create the seeds along the edge. Possible values are:
  - UNIFORM_BY_NUMBER
  - UNIFORM_BY_SIZE
  - CURVATURE_BASED_BY_SIZE
  - BIASED
  - NONE
- If *attribute*=BIAS_METHOD, the return value is a SymbolicConstant specifying the bias type used to create the seeds along the edge. Possible values are:
  - SINGLE
  - DOUBLE
  - NONE
- If *attribute*=NUMBER, the return value is an Int specifying the number of element seeds along the edge.
- If *attribute*=AVERAGE_SIZE, the return value is a Float specifying the average element size along the edge.
- If *attribute*=DEVIATION_FACTOR, the return value is a Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length. If edge seeds are not defined, the return value is zero.
- If *attribute*=MIN_SIZE_FACTOR, the return value is a Float specifying the size of the smallest allowable element as a fraction of the specified global element size. If edge seeds are not defined, the return value is zero.
- If *attribute*=BIAS_RATIO, the return value is a Float specifying the length ratio of the largest element to the smallest element.
- If *attribute*=BIAS_MIN_SIZE, the return value is a Float specifying the length of the largest element; only applicable if the EDGE_SEEDING_METHOD is BIASED and seeds were specified by minimum and maximum sizes.
- If *attribute*=BIAS_MAX_SIZE, the return value is a Float specifying the length of the largest element; only applicable if the EDGE_SEEDING_METHOD is BIASED and seeds were specified by minimum and maximum sizes.

- If *attribute*=VERTEX_ADJ_TO_SMALLEST_ELEM, the return value is an Int specifying the ID of the vertex next to the smallest element; only applicable if the EDGE_SEEDING_METHOD is BIASED.
- If *attribute*=SMALLEST_ELEM_LOCATION, the return value is a SymbolicConstant specifying the location of smallest elements for double bias seeds; only applicable if the EDGE_SEEDING_METHOD is BIASED and BIAS_METHOD is DOUBLE. Possible values are:
  - SMALLEST_ELEM_AT_CENTER
  - SMALLEST_ELEM_AT_ENDS
  - NONE
- If *attribute*=CONSTRAINT, the return value is a SymbolicConstant specifying how close the seeds must be matched by the mesh. Possible values are:
  - FREE
  - FINER
  - FIXED
  - NONE

  A value of NONE indicates that the edge is not seeded.

Optional arguments

None.

Return value

The return value is a Float, an Int, or a SymbolicConstant depending on the value of the *attribute* argument.

Exceptions

None.

`getElementType(...)`

This method returns the *ElemType* object of a given element shape assigned to a region of the assembly.

Required arguments

region

A *Cell*, a *Face*, or an *Edge* object specifying the region to be queried.

elemShape

A SymbolicConstant specifying the shape of the element for which to return the element type. Possible values are:

- LINE
- QUAD
- TRI
- HEX
- WEDGE
- TET

Optional arguments

None.

Return value

ElemType object.

Exceptions

TypeError If the region cannot be associated with element types or if the *elemShape* is not consistent with the dimension of the *region*.

`getIncompatibleMeshInterfaces(...)`

This method returns a sequence of face objects that are meshed with incompatible elements.

Required arguments

None.

Optional arguments

cells A sequence of cell objects which will be used to search the incompatible faces.

Return value

A sequence of *Face* objects.

Exceptions

None.

`getMeshControl(...)`

This method returns a mesh control parameter for the specified region of the assembly.

Required arguments

region A *Cell*, a *Face*, or an *Edge* object specifying the region to be queried.

attribute A SymbolicConstant specifying the mesh control attribute to return. Possible values are:

- ELEM_SHAPE
- TECHNIQUE
- ALGORITHM
- MIN_TRANSITION

The return value is dependent on the *attribute* argument.

- If *attribute*=ELEM_SHAPE, the return value is a SymbolicConstant specifying the element shape used during meshing. Possible values are:
  - LINE
  - QUAD
  - TRI
  - QUAD_DOMINATED
  - HEX
  - TET
  - WEDGE
  - HEX_DOMINATED
- If *attribute*=TECHNIQUE, the return value is a SymbolicConstant specifying the meshing technique to be used during meshing. Possible values are:
  - FREE
  - STRUCTURED
  - SWEEP
  - UNMESHABLE

  Where UNMESHABLE indicates that no meshing technique is applicable with the currently assigned element shape.
- If *attribute*=ALGORITHM, the return value is a SymbolicConstant specifying the meshing algorithm to be used during meshing. Possible values are:
  - MEDIAL_AXIS
  - ADVANCING_FRONT
  - DEFAULT
  - NON_DEFAULT
  - NONE

  Where NONE indicates that no algorithm is applicable.
- If *attribute*=MIN_TRANSITION, the return value is a Boolean indicating whether minimum transition will be used during meshing. This option is applicable only to the following:
  - Free quadrilateral meshing or sweep hexahedral meshing with *algorithm*=MEDIAL_AXIS.
  - Structured quadrilateral meshing.

Optional arguments

None.

Return value

The return value is a SymbolicConstant or a Boolean depending on the value of the *attribute* argument.

Exceptions

TypeError	The region cannot carry mesh controls.

getMeshStats(...)

This method returns the mesh statistics for the given part instances or regions.

Required arguments

regions A sequence or tuple of *PartInstance* objects or Geometry regions for which mesh statistics should be returned.

Optional arguments

None.

Return value

A MeshStats object.

Exceptions

None.

getPartSeeds(...)

This method returns a part seed parameter for the specified instance.

Required arguments

region A *PartInstance* object specifying the part instance to be queried.

attribute A SymbolicConstant specifying the type of part seed attribute to return. Possible values are:

- SIZE
- DEFAULT_SIZE
- DEVIATION_FACTOR
- MIN_SIZE_FACTOR

The return value is dependent on the value of the *attribute* argument.

- If *attribute*=SIZE, the return value is a Float specifying the assigned global element size. If part seeds are not defined, the return value is zero.
- If *attribute*=DEFAULT_SIZE, the return value is a Float specifying a suggested default global element size based upon the part geometry.
- If *attribute*=DEVIATION_FACTOR, the return value is a Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length. If part seeds are not defined, the return value is zero.

- If *attribute*=MIN_SIZE_FACTOR, the return value is a Float specifying the size of the smallest allowable element as a fraction of the specified global element size. If part seeds are not defined, the return value is zero.

Optional arguments

None.

Return value

The return value is a Float, and its value is dependent on the *attribute* argument.

Exceptions

An exception occurs if the part instance does not contain native geometry.	Error: Part instance does not contain native geometry

`getUnmeshedRegions()`

This method returns all geometric regions in the assembly that require a mesh for submitting an analysis but are either unmeshed or are meshed incompletely.

Arguments

None.

Return value

A Region object, or None.

Exceptions

None.

`ignoreEntity(...)`

This method creates a virtual topology feature. Virtual topology allows unimportant entities to be ignored during mesh generation. You can combine two adjacent faces by specifying a common edge to ignore. Similarly, you can combine two adjacent edges by specifying a common vertex to ignore.

Required arguments

entities	A sequence of vertices and edges specifying the entities to be ignored during meshing.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

restoreIgnoredEntity(...)

This method restores vertices and edges that have been merged using a virtual topology feature.

Required arguments

entities

A sequence of *IgnoredVertex* objects and *IgnoredEdge* objects specifying the entities to be restored.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

seedEdgeByBias(...)

This method seeds the given edges nonuniformly using the specified number of elements and bias ratio or the specified minimum and maximum element sizes.

Required arguments

biasMethod

A SymbolicConstant specifying whether single- or double-biased seed distribution will be applied. If unspecified, single-biased seed distribution will be applied. Possible values are:

- SINGLE: Single-biased seed distribution will be applied.
- DOUBLE: Double-biased seed distribution will be applied.

end1Edges

A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near the end where the normalized curve parameter=0.0. You must provide either the *end1Edges* or the *end2Edges* argument or both when *biasMethod*=SINGLE and omit both of them when *biasMethod*=DOUBLE.



Note:

You can determine which end is which by the order of the vertex indices returned by *getVertices()*.

end2Edges

A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near the end where the normalized curve parameter=1.0.

centerEdges A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near edge center. You must provide either the *centerEdges* or the *endEdges* argument or both when *biasMethod*=DOUBLE and omit both of them when *biasMethod*=SINGLE.

endEdges A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near edge ends.

ratio A Float specifying the ratio of the largest element to the smallest element. Possible values are $1.0 \le ratio \le 10^6$.

number An Int specifying the number of elements along each edge. Possible values are $1 \le number \le 10^4$.

minSize A Float specifying the desired smallest element size.

maxSize A Float specifying the desired largest element size.



Note:

You must specify either the *ratio* and *number* or *minSize* and *maxSize* pair of arguments.

Optional arguments

constraint A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value

`None.`

Exceptions

None.

`seedEdgeByNumber(...)`

This method seeds the given edges uniformly based on the number of elements along the edges.

Required arguments

edges A sequence of *Edge* objects specifying the edges to seed.

number

An Int specifying the number of elements along each edge. Possible values are $1 \leq number \leq 10^4$.

Optional arguments

constraint

A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value

`None.`

Exceptions

None.

`seedEdgeBySize(...)`

This method seeds the given edges either uniformly or following edge curvature distribution, based on the desired element size.

Required arguments

edges

A sequence of *Edge* objects specifying the edges to seed.

size

A Float specifying the desired element size.

Optional arguments

deviationFactor

A Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length.

minSizeFactor

A Float specifying the size of the smallest allowable element as a fraction of the specified global element size.

constraint

A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value
`None.`

Exceptions
None.

seedPartInstance(...)

This method assigns global edge seeds to the given part instances.

Required arguments

regions A sequence of *PartInstance* objects specifying the part instances to seed.

size A Float specifying the desired global element size for the edges.

Optional arguments

deviationFactor A Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length.

minSizeFactor A Float specifying the size of the smallest allowable element as a fraction of the specified global element size.

constraint A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.

Return value
`None.`

Exceptions
None.

setBoundaryLayerControls(...)

This method sets the control parameters for boundary layer mesh for the specified regions.

Required arguments

regions A sequence of *Cell* objects specifying the regions for which to set the boundary layer mesh control parameters.

firstElemSize

A Float specifying the height of the first element layer off boundary. Possible values are $0.0 < firstElemSize \leq 10^6$.

growthFactor

A Float specifying the ratio of heights of any two consecutive element layers. Possible values are $1.0 \leq growthFactor \leq 10.0$.

numLayers

An Int specifying the number of element layers to be generated. Possible values are $1 \leq numLayers \leq 10^4$.

Optional arguments

inactiveFaces

A sequence of *Face* objects specifying the faces where boundary layer should not be generated. By default, boundary layer mesh will be generated on all faces of the selected regions.

setName

A String specifying a unique name for a set that will contain boundary layer elements.

Return value

`None.`

Exceptions

None.

`setElementType(...)`

This method assigns element types to the specified regions.

Required arguments

regions

A sequence of Geometry regions or *MeshElement* objects, or a Set object containing either geometry regions or elements, specifying the regions to which element types are to be assigned.

elemTypes

A sequence of ElemType objects, one for each element shape applicable to the regions.



Note:

If an ElemType object has an UNKNOWN_*xxx* value for *elemCode*, its order will be deduced from the order of other valid ElemType objects within the same `setElementType` command. If no valid ElemType objects can be found, the order will remain unchanged.

Optional arguments

None.

Return value

`None.`

Exceptions

As a result of the element assignment, a region must have the same library, family, and order for all its assigned element types. Otherwise, an exception will be thrown.

For example, suppose the Hex, Wedge, and Tet elements previously assigned to a cell are all linear. The user now constructs an ElemType object with a quadratic Hex element and includes only this object in the `setElementType` command. An exception will be thrown because the Wedge and Tet elements will remain linear (i.e., As Is) and become incompatible with the newly assigned quadratic Hex element.

`setLogicalCorners(...)`

This method sets the logical corners for a mappable face region.

Required arguments

region A Face region.

corners Three, four, or five Vertex objects defining the logical corners for a given mappable face region.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`setMeshControls(...)`

This method sets the mesh control parameters for the specified regions.

Required arguments

regions A sequence of Face or Cell regions specifying the regions for which to set the mesh control parameters.

Optional arguments

elemShape A SymbolicConstant specifying the element shape to be used for meshing. The default value is QUAD for Face regions and HEX for Cell regions. If unspecified, the existing element shape will remain unchanged. Possible values are:

- QUAD: Quadrilateral mesh.
- QUAD_DOMINATED: Quadrilateral-dominated mesh.
- TRI: Triangular mesh.
- HEX: Hexahedral mesh.

- HEX_DOMINATED: Hex-dominated mesh.
- TET: Tetrahedral mesh.
- WEDGE: Wedge mesh.

technique

A SymbolicConstant specifying the mesh technique to be used. The default value is FREE for Face regions. For Cell regions the initial value depends on the geometry of the regions and can be STRUCTURED, SWEEP, or "unmeshable." If unspecified, the existing mesh technique(s) will remain unchanged. Possible values are:

- FREE: Free mesh technique.
- STRUCTURED: Structured mesh technique.
- SWEEP: Sweep mesh technique.
- BOTTOM_UP: Bottom-up mesh technique. Only applicable for cell regions.
- SYSTEM_ASSIGN: Allow the system to assign a suitable technique. The actual technique assigned can be STRUCTURED, SWEEP, or "unmeshable."

algorithm

A SymbolicConstant specifying the algorithm used to generate the mesh for the specified regions. Possible values are MEDIAL_AXIS, ADVANCING_FRONT, and NON_DEFAULT. If unspecified, the existing value will remain unchanged. This option is applicable only to the following:

- Free quadrilateral or quadrilateral-dominated meshing. In this case the possible values are MEDIAL_AXIS and ADVANCING_FRONT.
- Sweep hexahedral or hexahedral-dominated meshing. In this case the possible values are MEDIAL_AXIS and ADVANCING_FRONT.
- Free tetrahedral meshing. In this case the only possible value is NON_DEFAULT, and it indicates that the free tetrahedral-meshing technique available in Abaqus 6.4 or earlier will be used. If algorithm is not specified, the default tetrahedral-meshing technique will be used.

minTransition

A Boolean specifying whether minimum transition is to be applied. The default value is ON. If unspecified, the existing value will remain unchanged. This option is applicable only in the following cases:

- Free quadrilateral meshing or hexahedral sweep meshing with *algorithm*=MEDIAL_AXIS.
- Structured quadrilateral meshing.

sizeGrowth

A SymbolicConstant specifying element size growth to be applied when generating the interior of a tetrahedral mesh. Possible values are MODERATE and MAXIMUM. If unspecified, the existing value will remain unchanged. This option only applies to the default tetrahedral mesher.

allowMapped

A Boolean specifying whether mapped meshing can be used to replace the selected mesh technique. The *allowMapped* argument is applicable only in the following cases:

- Free triangular meshing.
- Free quadrilateral or quadrilateral-dominated meshing with *algorithm*=ADVANCING_FRONT.
- Hexahedral or hexahedral-dominated sweep meshing with *algorithm*=ADVANCING_FRONT.
- Free tetrahedral meshing. *allowMapped*=True implies that mapped triangular meshing can be used on faces that bound three-dimensional *regions*.

Return value
`None.`

Exceptions
None.

setSweepPath(...)

This method sets the sweep path for a sweepable region or the revolve path for a revolvable region.

Required arguments

region A sweepable region.

edge An *Edge* object specifying the sweep or revolve path.

sense A SymbolicConstant specifying the sweep sense. The sense will affect only how gasket elements will be created; it will have no effect if gasket elements are not used. Possible values are FORWARD or REVERSE.

If *sense*=FORWARD, the sense of the given edge's underlying curve will be used.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

verifyMeshQuality(...)

This method tests the quality of part instance meshes and returns poor-quality elements.

Required arguments

criterion A SymbolicConstant specifying the criterion used for the quality check. Possible values are:

ANALYSIS_CHECKS

When this criterion is specified Abaqus/CAE will invoke the element quality checks included with the input file processor for Abaqus/Standard and Abaqus/Explicit.

ANGULAR_DEVIATION

The maximum amount (in degrees) that an element's face corner angles deviate from the ideal angle. The ideal angle is 90° for quadrilateral element faces and 60° for triangular element faces. Elements with an angular deviation larger than the specified threshold will fail this test.

ASPECT_RATIO

The ratio between the lengths of the longest and shortest edges of an element. Elements with an aspect ratio larger than the specified threshold will fail this test.

GEOM_DEVIATION_FACTOR

The largest geometric deviation factor evaluated along any of the element edges associated with geometric edges or faces. The geometric deviation factor along an element edge is calculated by dividing the maximum gap between the element edge and its associated geometry by the length of the element edge. Elements with a geometric deviation factor larger than the specified threshold will fail this test.

LARGE_ANGLE

The largest corner angle on any of an element's faces. Elements with face angles larger than the specified threshold (in degrees) will fail this test.

LONGEST_EDGE

The length of an element's longest edge. Elements with an edge longer than the specified threshold will fail this test.

MAX_FREQUENCY

An estimate of an element's contribution to the initial maximum allowable frequency for Abaqus/Standard analyses. This calculation requires appropriate section assignments and material definitions. Elements whose maximum allowable frequency is smaller than the given value will fail this test.

SHAPE_FACTOR

The shape factor for triangular and tetrahedral elements. This is the ratio between the element area or volume and the optimal element area or volume. Elements with a shape factor smaller than the specified threshold will fail this test.

SHORTEST_EDGE

The length of an element's shortest edge. Elements with an edge shorter than the specified threshold will fail this test.

SMALL_ANGLE

The smallest corner angle on any of an element's faces. Elements with face angles smaller than the given value (in degrees) will fail this test.

STABLE_TIME_INCREMENT

An estimate of an element's contribution to the initial maximum stable time increment for Abaqus/Explicit analyses. This calculation requires appropriate section assignments and material definitions. Elements that require a time increment smaller than the given value will fail this test.

Optional arguments

threshold

A Float value used to determine low quality elements according to the specified criterion. This argument is ignored when the ANALYSIS_CHECKS criterion is used. For other criterion, if this argument is unspecified then no list of failed elements will be returned.

elemShape

A SymbolicConstant specifying an element shape for limiting the query. Possible values are LINE, QUAD, TRI, HEX, WEDGE, and TET.

regions

A sequence of *Region* or *MeshElement* objects. If you do not specify the *regions* argument, all meshes in the assembly are considered.

Return value

A Dictionary object containing values for some number of the following keys: failedElements, warningElements, naElements (sequences of *MeshElement* objects); numElements (Int); average, worst (Float); worstElement (*MeshElement* object)

Exceptions

None.

Part object

The following commands operate on Part objects. For more information about the Part object, see *Part object*.

Access

```
import mesh
```

assignStackDirection(...)

This method assigns a stack direction to geometric cells. The stack direction will be used to orient the elements during mesh generation.

Required arguments

cells A sequence of *Cell* objects specifying regions where to assign the stack direction.

referenceRegion A *Face* object specifying the top side of the stack direction.

Optional arguments

None.

Return value

None.

Exceptions

None.

associateMeshWithGeometry(...)

This method associates a geometric entity with mesh entities that are either orphan elements, bounds orphan elements, or were created using the bottom-up meshing technique.

Required arguments

geometricEntity A *Cell*, a *Face*, an *Edge*, or a *Vertex* object specifying geometric entity to be associated with one or more mesh entities.

If the geometric entity is a *Cell* object then the argument *elements* must be specified.

If the geometric entity is a *Face* object then the argument *elemFaces* must be specified.

If the geometric entity is an *Edge* object then the argument *elemEdges* must be specified.

If the geometric entity is a *Vertex* object then the argument *node* must be specified.

Optional arguments

elements A sequence of *MeshElement* objects specifying the elements to be associated with the geometric cell.

elemFaces A sequence of *MeshFace* objects specifying the element faces to be associated with the geometric face.

elemEdges A sequence of *MeshEdge* objects specifying the element edges to be associated with the geometric edge.

node A *MeshNode* object specifying the mesh node to be associated with the geometric vertex.

Return value

`None.`

Exceptions

None.

`createVirtualTopology(...)`

This method creates a virtual topology feature by automatically merging faces and edges based on a set of geometric parameters. The edges and vertices that are being merged will be ignored during mesh generation.

Required arguments

None.

Optional arguments

regions A sequence of *Face* objects specifying the domain to search for geometric entities that need to be merged. Entities identified as candidates to be merged may be merged with entities from outside the specified region. If *regions* is not specified then the entire part is the domain for searching geometric entities that need to be merged.

mergeShortEdges A Boolean specifying whether to merge short edges. The default value is False.

shortEdgeThreshold A Float specifying a threshold that determines which edges are considered to be short. These edges are the candidate entities to be merged. This argument is a required argument if the argument*mergeShortEdges* equals True and it is ignored if the argument *mergeShortEdges* equals False.

mergeSmallFaces A Boolean specifying whether to merge faces with small area. The default value is False.

smallFaceAreaThreshold A Float specifying a threshold that determines which faces are considered to have a small area. These faces are the candidate entities to be merged. This argument is a required argument if the argument*mergeSmallFaces* equals True and it is ignored if the argument *mergeSmallFaces* equals False.

mergeSliverFaces A Boolean specifying whether to merge faces with high aspect ratio. The default value is False.

faceAspectRatioThreshold
A Float specifying a threshold that determines which faces are considered to have high aspect ratio. These faces are the candidate entities to be merged. This argument is a required argument if the argument*mergeSliverFaces* equals True and it is ignored if the argument *mergeSliverFaces* equals False.

mergeSmallAngleFaces
A Boolean specifying whether to merge faces that have a sharp corner angle. The default value is False.

smallFaceCornerAngleThreshold
A Float specifying a threshold that determines which face corner angles are considered to be small. These faces will be candidate entities to be merged. This argument is a required argument if the argument*mergeSmallAngleFaces* equals True and it is ignored if the argument *mergeSmallAngleFaces* equals False.

mergeThinStairFaces
A Boolean specifying whether to merge faces that represent a thin stair-like feature. The default value is False.

thinStairFaceThreshold
A Float specifying a threshold that determines which faces representing small stair-like features are considered thin. These faces will be candidate entities to be merged. This argument is required if the argument *mergeThinStairFaces* is True and it is ignored if *mergeThinStairFaces* is False.

ignoreRedundantEntities
A Boolean specifying whether to abstract away redundant edges and vertices. The default value is False.

cornerAngleTolerance
A Float specifying the angle deviation from 180 degrees at a vertex or at an edge such that the two edges radiating from the vertex or the two faces bounded by the edge can be merged. The default value is 30.0 degrees.

applyBlendControls
A Boolean specifying whether to verify that blend faces can be merged with neighboring faces. If *applyBlendControls* is true then all faces that have angle larger than *blendSubtendedAngleTolerance* and a radius smaller than *blendRadiusTolerance* will not be merged with neighboring faces unless the neighboring faces are also blend faces with similar geometric characteristics. The default value is False.

blendSubtendedAngleTolerance
A Float specifying the largest subtended angle of blend faces that can be merged with neighboring faces. This argument is a required argument if the argument *applyBlendControls* equals True and it is ignored if the argument *applyBlendControls* equals False.

blendRadiusTolerance
A Float specifying the smallest radius of curvature of blend faces that can be merged with neighboring faces. This argument is a required argument if the argument *applyBlendControls* equals True and it is ignored if the argument *applyBlendControls* equals False.

Return value

A Feature object.

Exceptions

None.

deleteBoundaryLayerControls(...)

This method deletes the control parameters for boundary layer mesh for all the specified regions.

Required arguments

regions A sequence of *Cell* objects specifying the regions for which to set the boundary layer mesh control parameters.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteMesh(...)

This method deletes a subset of the mesh that contains the native elements from the given parts or regions.

Required arguments

regions A sequence of Part objects or *Region* objects specifying the parts or regions from which the native mesh is to be deleted.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteMeshAssociationWithGeometry(...)

This method deletes the association of geometric entities with mesh entities.

Required arguments

geometricEntities A sequence of *Cell* objects, *Face* objects, *Edge* objects, or *Vertex* objects specifying the geometric entities that will be disassociated from the mesh.

Optional arguments

addBoundingEntities A Boolean specifying whether the mesh will also be disassociated from the geometric entities that bounds the given *geometricEntities*. For example, if the argument *geometricEntities* contains a face, this boolean indicates whether the edges and vertices that bound the face will also be disassociated from the mesh. The default value is False.

Return value

`None.`

Exceptions

None.

`deletePreviewMesh()`

This method deletes all boundary meshes in the parts. See the *boundaryPreview* argument of `generateMesh` for information about generating boundary meshes.

Arguments

None.

Return value

`None.`

Exceptions

None.

`deleteSeeds(...)`

This method deletes the global edge seeds from the given parts or deletes the local edge seeds from the given edges.

Required arguments

regions A sequence of Part objects or Edge objects specifying the parts or edges from which the seeds are to be deleted.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

generateMesh(...)

This method generates a mesh in the given parts or regions.

Required arguments

None.

Optional arguments

regions
A sequence of Part objects or *Region* objects specifying the parts or regions where the mesh is to be generated.

seedConstraintOverride
A Boolean specifying whether mesh generation is allowed to modify seed constraints. The default value is OFF.

meshTechniqueOverride
A Boolean specifying whether mesh generation is allowed to modify the existing mesh techniques so that a compatible mesh can be generated. The default value is OFF.

boundaryPreview
A Boolean specifying whether the generated mesh should be a boundary preview mesh. This option will only have an effect if any of the specified regions are to be meshed with tetrahedral elements or using the bottom-up technique with hexahedral or wedge elements. The default value is OFF.

boundaryMeshOverride
A Boolean specifying whether mesh generation is allowed to modify an existing boundary preview mesh. This option will only have an effect if any of the specified regions are to be meshed with tetrahedral elements and a boundary preview mesh already exists. The default value is OFF.

Return value

`None.`

Exceptions

None.

generateBottomUpExtrudedMesh(...)

This method generates solid elements by extruding a 2D mesh along a vector, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell
A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native parts.

numberOfLayers
An Int specifying the number of layers to be generated along the extrusion vector.

extrudeVector
A sequence of sequences of Floats specifying the start point and end point of a vector. Each point is defined by a tuple of three coordinates indicating its position. The direction of the mesh extrusion operation is from the first point to the second point.

Optional arguments

At least one of the three optional "SourceSide" arguments must be specified.

geometrySourceSide
A *Region* of *Face* objects specifying the geometric domain to be used as the source for the extrude meshing operation.

elemFacesSourceSide
A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the extrude meshing operation.

elemSourceSide
A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the extrude meshing operation.

depth
A Float specifying the distance of the mesh extrusion. If unspecified, the vector length of the *extrudeVector* argument is assumed.

targetSide
A datum plane, a sequence of *Face* objects, a sequence of *MeshFace* objects, or a sequence of 2D *MeshElement* objects specifying the target of the extrude meshing operation. If specified, this argument overrides the *depth* argument, and all points on the source will be extruded in the direction of the extrusion vector until meeting the target.

biasRatio
A Float specifying a ratio of the element size in the extrusion direction between the source and the target sides of the extrusion. The default is 1.0, meaning no bias.

extendElementSets
A Boolean specifying whether existing element sets that include source elements will be extended to also include extruded elements. This argument is ignored for native parts. The default value is False.

Return value

`None.`

Exceptions

None.

`generateBottomUpSweptMesh(...)`

This method generates solid elements by sweeping a 2D mesh, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell
A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native parts.

Optional arguments

At least one of the following three arguments must be specified: *geometrySourceSide*, *elemFacesSourceSide*, or *elemSourceSide*.

In addition one of the following two sets of arguments must also be specified:

- One of the following three arguments: *geometryConnectingSides*, *elemFacesConnectingSides*, or *elemConnectingSides*. In this case the targetSide is an optional argument.
- *targetSide* and *numberOfLayers*.

geometrySourceSide
A *Region* of *Face* objects specifying the geometric domain to be used as the source for the sweep meshing operation.

elemFacesSourceSide
A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the sweep meshing operation.

elemSourceSide
A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the sweep meshing operation.

geometryConnectingSides
A *Region* of *Face* objects specifying connecting sides of the sweep meshing operation.

elemFacesConnectingSides
A sequence of *MeshFace* objects specifying connecting sides of the sweep meshing operation.

elemConnectingSides
A sequence of 2D *MeshElement* objects specifying connecting sides of the sweep meshing operation.

targetSide
A *Face* object specifying the target side of the sweep meshing operation.

numberOfLayers
An Int specifying the number of layers to be generated along the sweep direction.

extendElementSets
A Boolean specifying whether existing element sets that include source elements will be extended to also include swept elements. This argument is ignored for native parts. The default value is False.

Return value

`None.`

Exceptions

None.

`generateBottomUpRevolvedMesh(...)`

This method generates solid elements by revolving a 2D mesh around an axis, either on an orphan mesh or within a cell region using a bottom-up technique.

Required arguments

cell A *Cell* object specifying the geometric region where the mesh is to be generated. This argument is valid only for native parts.

numberOfLayers An Int specifying the number of layers of elements to be generated around the axis of revolution.

axisOfRevolution A sequence of sequences of Floats specifying the two points of the vector that describes the axis of revolution. Each point is defined by a tuple of three coordinates indicating its position. The direction of the axis of revolution is from the first point to the second point. The orientation of the revolution operation follows the right-hand-rule about the axis of revolution.

angleOfRevolution A Float specifying the angle of revolution.

Optional arguments

At least one of the three optional "SourceSide" arguments must be specified.

geometrySourceSide A *Region* of *Face* objects specifying the geometric domain to be used as the source for the revolve meshing operation.

elemFacesSourceSide A sequence of *MeshFace* objects specifying the faces of 3D elements to be used as the source for the revolve meshing operation.

elemSourceSide A sequence of 2D *MeshElement* objects specifying the elements to be used as the source for the revolve meshing operation.

extendElementSets A Boolean specifying whether existing element sets that include source elements will be extended to also include extruded elements. This argument is ignored for native parts. The default value is False.

Return value

`None.`

Exceptions

None.

`getEdgeSeeds(...)`

This method returns an edge seed parameter for a specified edge of a part.

Required arguments

edge An *Edge* object specifying the edge to be queried.

attribute A SymbolicConstant specifying the type of edge seed attribute to return. Possible values are:

- EDGE_SEEDING_METHOD
- BIAS_METHOD
- NUMBER
- AVERAGE_SIZE
- DEVIATION_FACTOR
- MIN_SIZE_FACTOR
- BIAS_RATIO
- BIAS_MIN_SIZE
- BIAS_MAX_SIZE
- VERTEX_ADJ_TO_SMALLEST_ELEM
- SMALLEST_ELEM_LOCATION
- CONSTRAINT

The return value is dependent on the *attribute* argument.

- If *attribute*=EDGE_SEEDING_METHOD, the return value is a SymbolicConstant specifying the edge seeding method used to create the seeds along the edge. Possible values are:
  - UNIFORM_BY_NUMBER
  - UNIFORM_BY_SIZE
  - CURVATURE_BASED_BY_SIZE
  - BIASED
  - NONE
- If *attribute*=BIAS_METHOD, the return value is a SymbolicConstant specifying the bias type used to create the seeds along the edge. Possible values are:
  - SINGLE
  - DOUBLE
  - NONE
- If *attribute*=NUMBER, the return value is an Int specifying the number of element seeds along the edge.
- If *attribute*=AVERAGE_SIZE, the return value is a Float specifying the average element size along the edge.
- If *attribute*=DEVIATION_FACTOR, the return value is a Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length. If edge seeds are not defined, the return value is zero.
- If *attribute*=MIN_SIZE_FACTOR, the return value is a Float specifying the size of the smallest allowable element as a fraction of the specified global element size. If edge seeds are not defined, the return value is zero.
- If *attribute*=BIAS_RATIO, the return value is a Float specifying the length ratio of the largest element to the smallest element.
- If *attribute*=BIAS_MIN_SIZE, the return value is a Float specifying the length of the largest element; only applicable if the EDGE_SEEDING_METHOD is BIASED and seeds were specified by minimum and maximum sizes.

- If *attribute*=BIAS_MAX_SIZE, the return value is a Float specifying the length of the largest element; only applicable if the EDGE_SEEDING_METHOD is BIASED and seeds were specified by minimum and maximum sizes.
- If *attribute*=VERTEX_ADJ_TO_SMALLEST_ELEM, the return value is an Int specifying the ID of the vertex next to the smallest element; only applicable if the EDGE_SEEDING_METHOD is BIASED.
- If *attribute*=SMALLEST_ELEM_LOCATION, the return value is a SymbolicConstant specifying the location of smallest elements for double bias seeds; only applicable if the EDGE_SEEDING_METHOD is BIASED and BIAS_METHOD is DOUBLE. Possible values are:
  - SMALLEST_ELEM_AT_CENTER
  - SMALLEST_ELEM_AT_ENDS
  - NONE
- If *attribute*=CONSTRAINT, the return value is a SymbolicConstant specifying how close the seeds must be matched by the mesh. Possible values are:
  - FREE
  - FINER
  - FIXED
  - NONE

  A value of NONE indicates that the edge is not seeded.

Optional arguments

None.

Return value

The return value is a Float, an Int, or a SymbolicConstant depending on the value of the *attribute* argument.

Exceptions

None.

`getElementType(...)`

This method returns the *ElemType* object of a given element shape assigned to a region of a part.

Required arguments

region

A *Cell*, a *Face*, or an *Edge* object specifying the region to be queried.

elemShape

A SymbolicConstant specifying the shape of the element for which to return the element type. Possible values are:

- LINE
- QUAD
- TRI

- HEX
- WEDGE
- TET

Optional arguments

None.

Return value

ElemType object.

Exceptions

TypeError The region cannot be associated with element types or the *elemShape* is not consistent with the dimension of the *region*.

`getIncompatibleMeshInterfaces(...)`

This method returns a sequence of face objects that are meshed with incompatible elements.

Required arguments

None.

Optional arguments

cells A sequence of cell objects which will be used to search the incompatible faces.

Return value

A sequence of *Face* objects.

Exceptions

None.

`getMeshControl(...)`

This method returns a mesh control parameter for the specified region of a part.

Required arguments

region A *Cell*, a *Face*, or an *Edge* object specifying the region to be queried.

attribute A SymbolicConstant specifying the mesh control attribute to return. Possible values are:

- ELEM_SHAPE
- TECHNIQUE

- ALGORITHM
- MIN_TRANSITION

The return value depends on the value of the *attribute* argument.

- If *attribute*=ELEM_SHAPE, the return value is a SymbolicConstant specifying the element shape used during meshing. Possible values are:
  - LINE
  - QUAD
  - TRI
  - QUAD_DOMINATED
  - HEX
  - TET
  - WEDGE
  - HEX_DOMINATED
- If *attribute*=TECHNIQUE, the return value is a SymbolicConstant specifying the meshing technique to be used during meshing. Possible values are:
  - FREE
  - STRUCTURED
  - SWEEP
  - UNMESHABLE

  Where UNMESHABLE indicates that no meshing technique is applicable with the currently assigned element shape.
- If *attribute*=ALGORITHM, the return value is a SymbolicConstant specifying the meshing algorithm to be used during meshing. Possible values are:
  - MEDIAL_AXIS
  - ADVANCING_FRONT
  - DEFAULT
  - NON_DEFAULT
  - NONE

  Where NONE indicates that no algorithm is applicable.
- If *attribute*=MIN_TRANSITION, the return value is a Boolean indicating whether minimum transition will be used during meshing. This option is applicable only to the following:
  - Free quadrilateral meshing or sweep hexahedral meshing with *algorithm*=MEDIAL_AXIS.
  - Structured quadrilateral meshing.

Optional arguments

None.

Return value

The return value is a SymbolicConstant or a Boolean depending on the value of the *attribute* argument.

Exceptions

TypeError The region cannot carry mesh controls.

getMeshStats(...)

This method returns the mesh statistics for the given regions.

Required arguments

regions A sequence or tuple of Geometry regions for which mesh statistics should be returned.

Optional arguments

None.

Return value

A MeshStats object.

Exceptions

None.

getPartSeeds(...)

This method returns a part seed parameter for the part.

Required arguments

attribute A SymbolicConstant specifying the type of part seed attribute to return. Possible values are:

- SIZE
- DEFAULT_SIZE
- DEVIATION_FACTOR
- MIN_SIZE_FACTOR

The return value depends on the value of the *attribute* argument.

- If *attribute*=SIZE, the return value is a Float specifying the assigned global element size. If part seeds are not defined, the return value is zero.
- If *attribute*=DEFAULT_SIZE, the return value is a Float specifying a suggested default global element size based upon the part geometry.
- If *attribute*=DEVIATION_FACTOR, the return value is a Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length. If part seeds are not defined, the return value is zero.

- If *attribute*=MIN_SIZE_FACTOR, the return value is a Float specifying the size of the smallest allowable element as a fraction of the specified global element size. If part seeds are not defined, the return value is zero.

Optional arguments

None.

Return value

The return value is a Float that depends on the value of the *attribute* argument.

Exceptions

An exception occurs if the part does not contain native geometry.	Error: Part does not contain native geometry

`getUnmeshedRegions()`

This method returns all geometric regions in the part that require a mesh for submitting an analysis but are either unmeshed or are meshed incompletely.

Arguments

None.

Return value

A Region object, or None.

Exceptions

None.

`ignoreEntity(...)`

This method creates a virtual topology feature. Virtual topology allows unimportant entities to be ignored during mesh generation. You can combine two adjacent faces by specifying a common edge to ignore. Similarly, you can combine two adjacent edges by specifying a common vertex to ignore.

Required arguments

entities	A sequence of vertices and edges specifying the entities to be ignored during meshing.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

restoreIgnoredEntity(...)

This method restores vertices and edges that have been merged using a virtual topology feature.

Required arguments

entities

A sequence of *IgnoredVertex* objects and *IgnoredEdge* objects specifying the entities to be restored.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

seedEdgeByBias(...)

This method seeds the given edges nonuniformly using the specified number of elements and bias ratio or the specified minimum and maximum element sizes.

Required arguments

biasMethod

A SymbolicConstant specifying whether single- or double-biased seed distribution will be applied. If unspecified, single-biased seed distribution will be applied. Possible values are:

- SINGLE: Single-biased seed distribution will be applied.
- DOUBLE: Double-biased seed distribution will be applied.

end1Edges

A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near the end where the normalized curve parameter=0.0. You must provide either the *end1Edges* or the *end2Edges* argument or both when *biasMethod*=SINGLE and omit both of them when *biasMethod*=DOUBLE.



Note:

You can determine which end is which by the order of the vertex indices returned by *getVertices()*.

end2Edges

A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near the end where the normalized curve parameter=1.0.

centerEdges A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near edge center. You must provide either the *centerEdges* or the *endEdges* argument or both when *biasMethod*=DOUBLE and omit both of them when *biasMethod*=SINGLE.

endEdges A sequence of *Edge* objects specifying the edges to seed. The smallest elements will be positioned near edge ends.

ratio A Float specifying the ratio of the largest element to the smallest element. Possible values are $1.0 \leq ratio \leq 10^6$.

number An Int specifying the number of elements along each edge. Possible values are $1 \leq number \leq 10^4$.

minSize A Float specifying the desired smallest element size.

maxSize A Float specifying the desired largest element size.



Note:

You must specify either the *ratio* and *number* or *minSize* and *maxSize* pair of arguments.

Optional arguments

constraint A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value

`None.`

Exceptions

None.

`seedEdgeByNumber(...)`

This method seeds the given edges uniformly based on the number of elements along the edges.

Required arguments

edges A sequence of *Edge* objects specifying the edges to seed.

number

An Int specifying the number of elements along each edge. Possible values are $1 \leq number \leq 10^4$.

Optional arguments

constraint

A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value

`None.`

Exceptions

None.

`seedEdgeBySize(...)`

This method seeds the given edges either uniformly or following edge curvature distribution, based on the desired element size.

Required arguments

edges

A sequence of *Edge* objects specifying the edges to seed.

size

A Float specifying the desired element size.

Optional arguments

deviationFactor

A Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length.

minSizeFactor

A Float specifying the size of the smallest allowable element as a fraction of the specified global element size.

constraint

A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.
- FIXED: The seeds must be exactly matched by the mesh (only with respect to the number of elements, not to the nodal positioning).

Return value
`None.`

Exceptions
None.

`seedPart(...)`

This method assigns global edge seeds to the given parts.

Required arguments

size A Float specifying the desired global element size for the edges.

Optional arguments

deviationFactor A Float specifying the deviation factor h/L, where h is the chordal deviation and L is the element length.

minSizeFactor A Float specifying the size of the smallest allowable element as a fraction of the specified global element size.

constraint A SymbolicConstant specifying how closely the seeds must be matched by the mesh. The default value is FREE. If unspecified, the existing constraint will remain unchanged. Possible values are:

- FREE: The resulting mesh can be finer or coarser than the specified seeds.
- FINER: The resulting mesh can be finer than the specified seeds.

Return value
`None.`

Exceptions
None.

`setBoundaryLayerControls(...)`

This method sets the control parameters for boundary layer mesh for the specified regions.

Required arguments

regions A sequence of *Cell* objects specifying the regions for which to set the boundary layer mesh control parameters.

firstElemSize A Float specifying the height of the first element layer off boundary. Possible values are $0.0 < firstElemSize \leq 10^6$.

growthFactor A Float specifying the ratio of heights of any two consecutive element layers. Possible values are $1.0 \leq growthFactor \leq 10.0$.

numLayers An Int specifying the number of element layers to be generated. Possible values are $1 \leq numLayers \leq 10^4$.

Optional arguments

inactiveFaces A sequence of *Face* objects specifying the faces where boundary layer should not be generated. By default, boundary layer mesh will be generated on all faces of the selected regions.

setName A String specifying a unique name for a set that will contain boundary layer elements.

Return value

`None.`

Exceptions

None.

`setElementType(...)`

This method assigns element types to the specified regions.

Required arguments

regions A sequence of Geometry regions or *MeshElement* objects, or a Set object containing either geometry regions or elements, specifying the regions to which element types are to be assigned.

elemTypes A sequence of ElemType objects, one for each element shape applicable to the regions.



Note:

If an ElemType object has an UNKNOWN_*xxx* value for *elemCode*, its order will be deduced from the order of other valid ElemType objects within the same `setElementType` command. If no valid ElemType objects can be found, the order will remain unchanged.

Optional arguments

None.

Return value

`None.`

Exceptions

As a result of the element assignment, a region must have the same library, family, and order for all its assigned element types. Otherwise, an exception will be thrown.

For example, suppose the Hex, Wedge, and Tet elements previously assigned to a cell are all linear. The user now constructs an ElemType object with a quadratic Hex element and includes only this object in the `setElementType` command. An exception will be thrown because the Wedge and Tet elements will remain linear (i.e., As Is) and become incompatible with the newly assigned quadratic Hex element.

`setLogicalCorners(...)`

This method sets the logical corners for a mappable face region.

Required arguments

region A Face region.

corners Three, four, or five Vertex objects defining the logical corners for a given mappable face region.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`setMeshControls(...)`

This method sets the mesh control parameters for the specified regions.

Required arguments

regions A sequence of Face or Cell regions specifying the regions for which to set the mesh control parameters.

Optional arguments

elemShape A SymbolicConstant specifying the element shape to be used for meshing. The default value is QUAD for Face regions and HEX for Cell regions. If unspecified, the existing element shape will remain unchanged. Possible values are:

- QUAD: Quadrilateral mesh.
- QUAD_DOMINATED: Quadrilateral-dominated mesh.
- TRI: Triangular mesh.
- HEX: Hexahedral mesh.
- HEX_DOMINATED: Hex-dominated mesh.
- TET: Tetrahedral mesh.
- WEDGE: Wedge mesh.

technique
A SymbolicConstant specifying the mesh technique to be used. The default value is FREE for Face regions. For Cell regions the initial value depends on the geometry of the regions and can be STRUCTURED, SWEEP, or "unmeshable." If unspecified, the existing mesh technique(s) will remain unchanged. Possible values are:

- FREE: Free mesh technique.
- STRUCTURED: Structured mesh technique.
- SWEEP: Sweep mesh technique.
- BOTTOM_UP: Bottom-up mesh technique. Only applicable for cell regions.
- SYSTEM_ASSIGN: Allow the system to assign a suitable technique. The actual technique assigned can be STRUCTURED, SWEEP, or "unmeshable."

algorithm
A SymbolicConstant specifying the algorithm used to generate the mesh for the specified regions. Possible values are MEDIAL_AXIS, ADVANCING_FRONT, and NON_DEFAULT. If unspecified, the existing value will remain unchanged. This option is applicable only to the following:

- Free quadrilateral or quadrilateral-dominated meshing. In this case the possible values are MEDIAL_AXIS and ADVANCING_FRONT.
- Sweep hexahedral or hexahedral-dominated meshing. In this case the possible values are MEDIAL_AXIS and ADVANCING_FRONT.
- Free tetrahedral meshing. In this case the only possible value is NON_DEFAULT, and it indicates that the free tetrahedral-meshing technique available in Abaqus 6.4 or earlier will be used. If algorithm is not specified, the default tetrahedral-meshing technique will be used.

minTransition
A Boolean specifying whether minimum transition is to be applied. The default value is ON. If unspecified, the existing value will remain unchanged. This option is applicable only in the following cases:

- Free quadrilateral meshing or hexahedral sweep meshing with *algorithm*=MEDIAL_AXIS.
- Structured quadrilateral meshing.

sizeGrowth
A SymbolicConstant specifying element size growth to be applied when generating the interior of a tetrahedral mesh. Possible values are MODERATE and MAXIMUM. If unspecified, the existing value will remain unchanged. This option only applies to the default tetrahedral mesher.

allowMapped
A Boolean specifying whether mapped meshing can be used to replace the selected mesh technique. The *allowMapped* argument is applicable only in the following cases:

- Free triangular meshing.
- Free quadrilateral or quadrilateral-dominated meshing with *algorithm*=ADVANCING_FRONT.
- Hexahedral or hexahedral-dominated sweep meshing with *algorithm*=ADVANCING_FRONT.
- Free tetrahedral meshing. *allowMapped*=True implies that mapped triangular meshing can be used on faces that bound three-dimensional *regions*.

Return value
`None.`

Exceptions

None.

`setSweepPath(...)`

This method sets the sweep path for a sweepable region or the revolve path for a revolvable region.

Required arguments

region A sweepable region.

edge An *Edge* object specifying the sweep or revolve path.

sense A SymbolicConstant specifying the sweep sense. The sense will affect only how gasket elements will be created; it will have no effect if gasket elements are not used. Possible values are FORWARD or REVERSE.

If *sense*=FORWARD, the sense of the given edge's underlying curve will be used.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`verifyMeshQuality(...)`

This method tests the mesh quality of a part and returns poor-quality elements.

Required arguments

criterion A SymbolicConstant specifying the criterion used for the quality check. Possible values are:

ANALYSIS_CHECKS

When this criterion is specified Abaqus/CAE will invoke the element quality checks included with the input file processor for Abaqus/Standard and Abaqus/Explicit.

ANGULAR_DEVIATION

The maximum amount (in degrees) that an element's face corner angles deviate from the ideal angle. The ideal angle is 90° for quadrilateral element faces and 60° for triangular element faces. Elements with an angular deviation larger than the specified threshold will fail this test.

ASPECT_RATIO

The ratio between the lengths of the longest and shortest edges of an element. Elements with an aspect ratio larger than the specified threshold will fail this test.

GEOM_DEVIATION_FACTOR

The largest geometric deviation factor evaluated along any of the element edges associated with geometric edges or faces. The geometric deviation factor along an element edge is calculated by dividing the maximum gap between the element edge and its associated geometry by the length of the element edge. Elements with a geometric deviation factor larger than the specified threshold will fail this test.

LARGE_ANGLE

The largest corner angle on any of an element's faces. Elements with face angles larger than the specified threshold (in degrees) will fail this test.

LONGEST_EDGE

The length of an element's longest edge. Elements with an edge longer than the specified threshold will fail this test.

MAX_FREQUENCY

An estimate of an element's contribution to the initial maximum allowable frequency for Abaqus/Standard analyses. This calculation requires appropriate section assignments and material definitions. Elements whose maximum allowable frequency is smaller than the given value will fail this test.

SHAPE_FACTOR

The shape factor for triangular and tetrahedral elements. This is the ratio between the element area or volume and the optimal element area or volume. Elements with a shape factor smaller than the specified threshold will fail this test.

SHORTEST_EDGE

The length of an element's shortest edge. Elements with an edge shorter than the specified threshold will fail this test.

SMALL_ANGLE

The smallest corner angle on any of an element's faces. Elements with face angles smaller than the given value (in degrees) will fail this test.

STABLE_TIME_INCREMENT

An estimate of an element's contribution to the initial maximum stable time increment for Abaqus/Explicit analyses. This calculation requires appropriate section assignments and material definitions. Elements that require a time increment smaller than the given value will fail this test.

Optional arguments

threshold

A Float value used to determine low quality elements according to the specified criterion. This argument is ignored when the ANALYSIS_CHECKS criterion is used. For other criterion, if this argument is unspecified then no list of failed elements will be returned.

elemShape

A SymbolicConstant specifying an element shape for limiting the query. Possible values are LINE, QUAD, TRI, HEX, WEDGE, and TET.

regions

A sequence of *Region* or *MeshElement* objects. If you do not specify the *regions* argument, the entire part mesh is considered.

Return value

A Dictionary object containing values for some number of the following keys: failedElements, warningElements, naElements (sequences of *MeshElement* objects); numElements (Int); average, worst (Float); worstElement (*MeshElement* object)

Exceptions

None.

ElemType object

The ElemType object is an argument object used as an argument in the `setElementType` command.

Access

```
import mesh
```

ElemType(...)

This method creates an ElemType object.

Path

```
ElemType
```

Required arguments

elemCode A SymbolicConstant specifying the Abaqus element code or just the element shape. Possible values are:

- C3D8R, specifying a 8-node linear brick, reduced integration with hourglass control.
- CODE, specifying add more codes.
- UNKNOWN_TRI, specifying an unknown element type associated with a triangular shape.
- UNKNOWN_QUAD, specifying an unknown element type associated with a quadrilateral shape.
- UNKNOWN_HEX, specifying an unknown element type associated with a hexahedral shape.
- UNKNOWN_WEDGE, specifying an unknown element type associated with a wedge shape.
- UNKNOWN_TET, specifying an unknown element type associated with a tetrahedral shape.

Optional arguments

elemLibrary A SymbolicConstant specifying the Abaqus element library to use. Possible values are STANDARD and EXPLICIT. The default value is STANDARD.

hourglassStiffness A Float specifying the hourglass stiffness. (For shell elements this is the membrane hourglass stiffness.) A value of zero indicates the default value should be used. The default value will be used where appropriate. The default value is 0.0.

This argument is applicable only to some Abaqus/Standard elements.

bendingHourglass A Float specifying the bending hourglass stiffness. A value of zero indicates the default value should be used. The default value will be used where appropriate. The default value is 0.0.

This argument is applicable only to some Abaqus/Standard elements.

drillingHourglass A Float specifying the drilling hourglass scaling factor. A value of zero indicates the default value should be used. The default value will be used where appropriate. The default value is 0.0.

This argument is applicable only to some Abaqus/Standard elements.

kinematicSplit

A SymbolicConstant specifying the kinematic split control. Possible values are AVERAGE_STRAIN, ORTHOGONAL, and CENTROID. The default value is AVERAGE_STRAIN.

This argument is applicable only to some Abaqus/Explicit elements.

distortionControl

A Boolean specifying whether to prevent negative element volumes or other excessive distortions in crushable materials. The default value is OFF.

This argument is applicable only to some Abaqus/Explicit elements.

lengthRatio

A Float specifying the length ratio for distortion control in crushable materials. Possible values are $0.0 \le$ *lengthRatio* ≤ 1.0. The default value is *lengthRatio*=**0.1**

This argument is applicable only when *distortionControl* is ON.

secondOrderAccuracy

A Boolean specifying the second-order accuracy option. The default value is OFF.

This argument is applicable only to some Abaqus/Explicit elements.

hourglassControl

A SymbolicConstant specifying the hourglass control. Possible values are RELAX_STIFFNESS, STIFFNESS, VISCOUS, ENHANCED, and COMBINED. The default value is ENHANCED.

This argument is applicable only to some Abaqus/Explicit elements.

weightFactor

A Float specifying a weight factor when *hourglassControl*=COMBINED. The default value is 0.5.

This argument is applicable only to some Abaqus/Explicit elements.

displacementHourglass

A Float specifying the displacement hourglass scaling factor. The default value will be used where appropriate. The default value is 1.0.

This argument is applicable only to some Abaqus/Explicit elements.

rotationalHourglass

A Float specifying the rotational hourglass scaling factor. The default value will be used where appropriate. The default value is 1.0.

This argument is applicable only to some Abaqus/Explicit elements.

outOfPlaneDisplacementHourglass

A Float specifying the out-of-plane displacement hourglass scaling factor. The default value will be used where appropriate. The default value is 1.0.

This argument is applicable only to some Abaqus/Explicit elements.

elemDeletion

A SymbolicConstant specifying the element deletion option. Possible values are DEFAULT, ON, and OFF. The default value is DEFAULT.

particleConversion

A SymbolicConstant specifying the particle conversion option for smoothed particle hydrodynamics. When not OFF or DEFAULT this argument refers to the criterion used for conversion of elements to particles. Possible values are

DEFAULT, OFF, TIME, STRAIN, and STRESS. The default value is DEFAULT.

This argument is applicable only to some Abaqus/Explicit elements.

particleConversionThreshold

A Float specifying the threshold value for the particle conversion criterion specified by *particleConversion*. The default value is 0.0.

This argument is applicable only to some Abaqus/Explicit elements.

particleConversionPPD

An Int specifying the number of particles per direction for element conversion when *particleConversion* is specified. The default value is 1.

This argument is applicable only to some Abaqus/Explicit elements.

particleConversionKernel

A SymbolicConstant specifying the interpolation function for particle conversion when *particleConversion* is specified. Possible values are CUBIC, QUADRATIC, and QUINTIC. The default value is CUBIC.

This argument is applicable only to some Abaqus/Explicit elements.

maxDegradation

A Float specifying the maximum degradation option for damage control. The default value is −1.0.

viscosity

A Float specifying the viscosity option. The default value is 0.0.

This argument is applicable only to some Abaqus/Standard elements.

linearBulkViscosity

A Float specifying the linear bulk viscosity scaling factor option for Abaqus/Explicit. The default value is 1.0.

quadraticBulkViscosity

A Float specifying the quadratic bulk viscosity scaling factor option for Abaqus/Explicit. The default value is 1.0.

numFourierModes

An Int specifying the number of Fourier modes. Possible values are 1, 2, 3, and 4. The default value is 1.

This argument is applicable only for axisymmetric elements with nonlinear asymmetric deformation.

nodeOffset

An Int specifying the positive offset number for specifying the additional nodes needed in the connectivity.

This argument is applicable only for axisymmetric elements with nonlinear asymmetric deformation.

linearKinematicCtrl

A Float specifying the linear kinematic conversion value.

This argument is applicable only to some Abaqus/Explicit elements.

initialGapOpening

A Float specifying the initial gap opening.

This parameter is applicable only to some Abaqus/Standard elements.

Return value

An ElemType object.

Exceptions

None.

Members

The ElemType object has members with the same names and descriptions as the arguments to the *ElemType* method.

MeshEdge object

The MeshEdge object refers to an element edge. It has no constructor or members. A MeshEdge object can be accessed via a *MeshEdgeArray* or a repository on a part or part instance.

Access

```
import part
mdb.models[name].parts[name].elemEdges[i]
mdb.models[name].parts[name].elementEdges[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].elemEdges[i]
mdb.models[name].rootAssembly.allInstances[name].elementEdges[i]
mdb.models[name].rootAssembly.instances[name].elemEdges[i]
mdb.models[name].rootAssembly.instances[name].elementEdges[i]
```

`getElements()`

This method returns a tuple of elements that share the element edge.

Arguments

None.

Return value

A tuple of *MeshElement* objects.

Exceptions

None.

`getElementsViaTopology()`

This method returns an array of *MeshElement* objects that are obtained by recursively finding adjacent elements via topology.

Optional arguments

domain A MeshElementArray object specifying the domain to include in the search. By default, all elements in the mesh are included.

Return value

A *MeshElementArray* object, which is a sequence of *MeshElement* objects.

Exceptions

None.

getNodesViaTopology()

This method returns an array of *MeshNode* objects that lie along element edges topologically in line with the element edge.

Optional arguments

domain A MeshElementArray object specifying the domain to include in the search. By default, all elements in the mesh are included.

Return value

A *MeshNodeArray* object, which is a sequence of *MeshNode* objects.

Exceptions

None.

getElemFaces()

This method returns a tuple of unique *MeshFace* objects that share the element edge.

Arguments

None.

Return value

A tuple of *MeshFace* objects.

Exceptions

None.

getNodes()

This method returns a tuple of nodes on the element edge.

Arguments

None.

Return value

A tuple of *MeshNode* objects.

Exceptions

None.

Members

The MeshEdge object has no members.

MeshEdgeArray object

The MeshEdgeArray is a sequence of *MeshEdge* objects.

Access

```
import part
mdb.models[name].parts[name].elementEdges
import assembly
mdb.models[name].rootAssembly.allInstances[name].elementEdges
mdb.models[name].rootAssembly.instances[name].elementEdges
```

MeshEdgeArray(...)

This method creates a MeshEdgeArray object.

Path

mesh.MeshEdgeArray

Required arguments

elemEdges A list of *MeshEdge* objects.

Optional arguments

None.

Return value

A MeshEdgeArray object.

Exceptions

None.

getSequenceFromMask(...)

This method returns the objects in the MeshEdgeArray identified using the specified *mask*. When large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A MeshEdgeArray object.

Exceptions

An exception occurs if the resulting sequence is empty.	Error: The mask results in an empty sequence

`getMask()`

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

Members

The MeshEdgeArray object has no members.

MeshElement object

The MeshElement object refers to an element of a native mesh or an orphan mesh. A MeshElement object can be accessed via a part or part instance using an index that refers to the internal numbering of the element repository. The index does not refer to the element label.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].elements[i]
mdb.models[name].parts[name].allInternalSurfaces[name].elements[i]
mdb.models[name].parts[name].allSets[name].elements[i]
mdb.models[name].parts[name].allSurfaces[name].elements[i]
mdb.models[name].parts[name].elements[i]
mdb.models[name].parts[name].sets[name].elements[i]
mdb.models[name].parts[name].surfaces[name].elements[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].elements[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name].elements[i]
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].elements[i]
mdb.models[name].rootAssembly.allInternalSets[name].elements[i]
mdb.models[name].rootAssembly.allInternalSurfaces[name].elements[i]
mdb.models[name].rootAssembly.allSets[name].elements[i]
mdb.models[name].rootAssembly.allSurfaces[name].elements[i]
mdb.models[name].rootAssembly.elements[i]
mdb.models[name].rootAssembly.instances[name].elements[i]
mdb.models[name].rootAssembly.instances[name].sets[name].elements[i]
mdb.models[name].rootAssembly.instances[name].surfaces[name]\
.elements[i]
mdb.models[name].rootAssembly.modelInstances[i].elements[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].elements[i]
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]\
.elements[i]
mdb.models[name].rootAssembly.sets[name].elements[i]
mdb.models[name].rootAssembly.surfaces[name].elements[i]
```

Element(...)

This method creates an element on an orphan mesh part from a sequence of nodes.

Path

mdb.models[*name*].parts[*name*].Element

Required arguments

nodes

A sequence of *MeshNode* objects.

elemShape

A SymbolicConstant specifying the shape of the new element. Possible values are LINE2, LINE3, TRI3, TRI6, QUAD4, QUAD8, TET4, TET10, WEDGE6, WEDGE15, HEX8, and HEX20.

Optional arguments

label An Int specifying the element label.

Return value

A MeshElement object.

Exceptions

None.

`getNodes()`

This method returns a tuple of node objects of the element.

Arguments

None.

Return value

A tuple of *MeshNode* objects.

Exceptions

None.

`getElemEdges()`

This method returns a tuple of unique element edge objects on the element.

Arguments

None.

Return value

A tuple of *MeshEdge* objects.

Exceptions

None.

`getElemFaces()`

This method returns a tuple of unique element face objects on the element.

Arguments

None.

Return value

A tuple of *MeshFace* objects.

Exceptions

None.

getAdjacentElements()

This method returns an array of element objects adjacent to the mesh element.

Arguments

None.

Return value

A *MeshElementArray* object which is a sequence of MeshElement objects.

Exceptions

None.

getElementsByFeatureEdge(...)

This method returns an array of mesh element objects that are obtained by recursively finding adjacent elements along a feature edge with a face angle of less than or equal to the specified angle.

Required arguments

angle A float specifying the value of the face angle in degrees.

Optional arguments

None.

Return value

A *MeshElementArray* object, which is a sequence of MeshElement objects.

Exceptions

None.

setValues(...)

This method modifies the MeshElement object.

Required arguments

None.

Optional arguments

label
An Int specifying the element label. This member may only be edited if the element belongs to an orphan mesh part. The specified label must be non-negative and must not be in use by any other element of the same part.

Return value

`None.`

Exceptions

None.

Members

The MeshElement object has the following members:

label
An Int specifying the element label.

type
A SymbolicConstant specifying the Abaqus element code.

instanceName
A String specifying the name of the part instance that owns this element.

connectivity
A tuple of Ints specifying the internal node indices that define the nodal connectivity. It is important to note the difference with OdbMeshElement object of ODB where the connectivity is node labels instead of node indices.

MeshElementArray object

The MeshElementArray is a sequence of *MeshElement* objects.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].elements
mdb.models[name].parts[name].allInternalSurfaces[name].elements
mdb.models[name].parts[name].allSets[name].elements
mdb.models[name].parts[name].allSurfaces[name].elements
mdb.models[name].parts[name].elements
mdb.models[name].parts[name].sets[name].elements
mdb.models[name].parts[name].surfaces[name].elements
import assembly
mdb.models[name].rootAssembly.allInstances[name].elements
mdb.models[name].rootAssembly.allInstances[name].sets[name].elements
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].elements
mdb.models[name].rootAssembly.allInternalSets[name].elements
mdb.models[name].rootAssembly.allInternalSurfaces[name].elements
mdb.models[name].rootAssembly.allSets[name].elements
mdb.models[name].rootAssembly.allSurfaces[name].elements
mdb.models[name].rootAssembly.elements
mdb.models[name].rootAssembly.instances[name].elements
mdb.models[name].rootAssembly.instances[name].sets[name].elements
mdb.models[name].rootAssembly.instances[name].surfaces[name].elements
mdb.models[name].rootAssembly.modelInstances[i].elements
mdb.models[name].rootAssembly.modelInstances[i].sets[name].elements
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]\
.elements
mdb.models[name].rootAssembly.sets[name].elements
mdb.models[name].rootAssembly.surfaces[name].elements
```

MeshElementArray(...)

This method creates a MeshElementArray object.

Path

mesh.MeshElementArray

Required arguments

elements	A list of *MeshElement* objects.

Optional arguments

None.

Return value

A MeshElementArray object.

Exceptions

None.

`getFromLabel(...)`

This method returns the object in the MeshElementArray with the given label.

Required arguments

label An Int specifying the label of the object.

Optional arguments

None.

Return value

A *MeshElement* object.

Exceptions

None.

`getSequenceFromMask(...)`

This method returns the objects in the MeshElementArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When a large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A MeshElementArray object.

Exceptions

None.

`getMask()`

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

getExteriorEdges()

This method returns the exterior element edges for 2D/shell elements in the MeshElementArray. These are edges referenced by exactly one element in the sequence. Nothing is returned if the sequence contains no topologically 2D/shell elements.

Arguments

None.

Return value

A MeshEdgeArray object specifying the element edges on the exterior.

Exceptions

None.

getExteriorFaces()

This method returns the exterior element faces for solid elements in the MeshElementArray. These are faces referenced by exactly one element in the sequence. Nothing is returned if the sequence contains no topologically solid elements.

Arguments

None.

Return value

A MeshFaceArray object specifying the element faces on the exterior.

Exceptions

None.

getByBoundingBox(...)

This method returns an array of element objects that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin

A float specifying the minimum X boundary of the bounding box.

yMin A float specifying the minimum Y boundary of the bounding box.

zMin A float specifying the minimum Z boundary of the bounding box.

xMax A float specifying the maximum X boundary of the bounding box.

yMax A float specifying the maximum Y boundary of the bounding box.

zMax A float specifying the maximum Z boundary of the bounding box.

Return value

A MeshElementArray object, which is a sequence of *MeshElement* objects.

Exceptions

None.

`getByBoundingCylinder(...)`

This method returns an array of element objects that lie within the specified bounding cylinder.

Required arguments

center1 A tuple of the X-, Y-, and Z-coordinates of the center of the first end of the cylinder.

center2 A tuple of the X-, Y-, and Z-coordinates of the center of the second end of the cylinder.

radius A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

A MeshElementArray object, which is a sequence of *MeshElement* objects.

Exceptions

None.

`getByBoundingSphere(...)`

This method returns an array of element objects that lie within the specified bounding sphere.

Required arguments

center A tuple of the X-, Y-, and Z-coordinates of the center of the sphere.

radius A float specifying the radius of the sphere.

Optional arguments

None.

Return value

A MeshElementArray object, which is a sequence of *MeshElement* objects.

Exceptions

None.

`getBoundingBox()`

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the element sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum x, y, and z boundary values of the bounding box.

high: a tuple of three floats representing the maximum x, y, and z boundary values of the bounding box.

Exceptions

None.

`getMemberFromAll(...)`

This method returns a tuple containing a member or members for each MeshElement in the MeshElementArray.

Required arguments

member A String specifying a MeshElement member name, or a tuple of member names.

Optional arguments

None.

Return value

A tuple of member values, one for each MeshElement, or a tuple of tuples of member values, if you specified a tuple of member names in the *member* argument.

Exceptions

An exception occurs if no such member exists.	AttributeError: 'MeshElement' object has no attribute ''

`sequenceFromLabels(...)`

This method returns the objects in the MeshElementArray identified using the specified labels.

Required arguments

labels	A sequence of Ints specifying the labels.

Optional arguments

None.

Return value

A MeshElementArray object.

Exceptions

An exception occurs if the resulting sequence is empty.	Error: The mask results in an empty sequence

Members

The MeshElementArray object has no members.

MesherOptions object

The MesherOptions object controls the default settings that Abaqus uses for all meshing methods. The MesherOptions object has no constructor. Abaqus creates the *MesherOptions* member when a session is started.

MesherOptions commands are intended for use at the beginning of scripts and in the `abaqus_v6.env` file only; they should not be used during an Abaqus/CAE session.

Access

```
session.defaultMesherOptions
```

`setValues(...)`

This method modifies the MesherOptions object.

Required arguments

None.

Optional arguments

elemShape2D
A SymbolicConstant specifying the default element shape for meshing two-dimensional objects. Possible values are QUAD, QUAD_DOMINATED, and TRI. The default value is QUAD_DOMINATED.

elemShape3D
A SymbolicConstant specifying the default element shape for meshing three-dimensional objects. Possible values are HEX, HEX_DOMINATED, WEDGE, and TET. The default value is HEX.

quadAlgorithm
A SymbolicConstant specifying the default algorithm for meshing an object with quad- or quad-dominated elements. Possible values are ADVANCING_FRONT and MEDIAL_AXIS. The default value is ADVANCING_FRONT.

allowMapped
A Boolean specifying whether Abaqus/CAE should allow mapped meshing, where appropriate. The default value is OFF.

minTransition
A Boolean specifying whether Abaqus/CAE should attempt to minimize the mesh transition when it moves from a coarse mesh to a fine mesh. The default value is ON.

guiPreferredElements
A list of SymbolicConstants specifying preferred Abaqus element types. This setting is relevant only when Abaqus/CAE is run interactively. When a part or part instance that has never been assigned an element type is meshed, this list is consulted. If an element type appropriate to the geometry is found in the list, it is assigned to the geometry. Multiple element types representing different shapes (for example, triangles and quadrilaterals) can be assigned in combination, but only element types that are compatible with each other are used. When more than one appropriate element type is found in the list, the first element type encountered takes precedence. This list is also consulted when populating the element type dialog; preferred types are selected by default for a region not previously assigned any element types. The default value is an empty list.

Return value

`None.`

Exceptions

None.

Members

The MesherOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

MeshFace object

The MeshFace object refers to an element face. It has no constructor or members. A MeshFace object can be accessed via a *MeshFaceArray* or a repository on a part or part instance.

Access

```
import part
mdb.models[name].parts[name].elementFaces[i]
mdb.models[name].parts[name].elemFaces[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].elementFaces[i]
mdb.models[name].rootAssembly.allInstances[name].elemFaces[i]
mdb.models[name].rootAssembly.instances[name].elementFaces[i]
mdb.models[name].rootAssembly.instances[name].elemFaces[i]
```

`getElemEdges()`

This method returns a tuple of unique element edges on the element face.

Arguments

None.

Return value

A tuple of *MeshEdge* objects.

Exceptions

None.

`getElements()`

This method returns a tuple of elements that share the element face.

Arguments

None.

Return value

A tuple of *MeshElement* objects.

Exceptions

None.

`getNodes()`

This method returns a tuple of nodes on the element face.

Arguments

None.

Return value

A tuple of *MeshNode* objects.

Exceptions

None.

getNodesByFaceAngle(...)

This method returns an array of mesh node objects that are obtained by recursively finding adjacent element faces that are at an angle of less than or equal to the specified angle.

Required arguments

angle A float specifying the value of the face angle.

Optional arguments

None.

Return value

A *MeshNodeArray* object, which is a sequence of *MeshNode* objects.

Exceptions

None.

getNormal(...)

This method returns the normal direction for the element face.

Required arguments

None.

Optional arguments

None.

Return value

A tuple of 3 floats representing the unit normal vector. If the element face is collapsed such that a normal cannot be computed, a zero-length vector is returned.

Exceptions

None.

getElemFacesByFaceAngle(...)

This method returns an array of element face objects that are obtained by recursively finding adjacent element faces that are at an angle of less than or equal to the specified angle.

Required arguments

angle
A float specifying the value of the face angle.

Optional arguments

None.

Return value

A *MeshFaceArray* object, which is a sequence of MeshFace objects.

Exceptions

None.

getElemEdgesByFaceAngle(...)

This method returns an array of element edge objects that are obtained by recursively finding adjacent element edges that are at an angle of less than or equal to the specified face angle.

Required arguments

angle
A float specifying the value of the face angle in degrees.

Optional arguments

None.

Return value

A *MeshEdgeArray* object, which is a sequence of *MeshEdge* objects.

Exceptions

None.

getElementsByFaceAngle(...)

This method returns an array of mesh Element objects that are obtained by recursively finding adjacent element faces that are at an angle of less than or equal to the specified angle.

Required arguments

angle
A float specifying the value of the face angle.

Optional arguments

None.

Return value

A *MeshElementArray* object, which is a sequence of *MeshElement* objects.

Exceptions

None.

getElemFacesByLimitingAngle(...)

This method returns an array of element edge objects that are obtained by recursively finding adjacent element faces that are at an angle of less than or equal to the specified face angle with the seed face.

Required arguments

angle A float specifying the value of the face angle in degrees.

Optional arguments

None.

Return value

A *MeshFaceArray* object, which is a sequence of MeshFace objects.

Exceptions

None.

getElementsViaTopology()

This method returns an array of mesh Element objects that are obtained by recursively finding adjacent elements via topology.

Arguments

None.

Return value

A *MeshElementArray* object, which is a sequence of *MeshElement* objects.

Exceptions

None.

getElemFacesByLayer()

This method returns an array of element face objects, obtained by traversing shell elements or the exterior of a solid mesh, and recursively finding adjacent element faces by layer.

Required arguments

numLayers A int specifying the value of the number of layers.

Optional arguments

None.

Return value

A *MeshFaceArray* object, which is a sequence of MeshFace objects.

Exceptions

None.

Members

The MeshFace object has the following members:

label An Int specifying the element label.

face A symbolic constant specifying the side of the element.

MeshFaceArray object

The MeshFaceArray is a sequence of *MeshFace* objects.

Access

```
import part
mdb.models[name].parts[name].elementFaces
import assembly
mdb.models[name].rootAssembly.allInstances[name].elementFaces
mdb.models[name].rootAssembly.instances[name].elementFaces
```

MeshFaceArray(...)

This method creates a MeshFaceArray object.

Path

mesh.MeshFaceArray

Required arguments

elemFaces A list of *MeshFace* objects.

Optional arguments

None.

Return value

A MeshFaceArray object.

Exceptions

None.

getSequenceFromMask(...)

This method returns the objects in the MeshFaceArray identified using the specified *mask*. When large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A MeshFaceArray object.

Exceptions

An exception occurs if the resulting sequence is empty.	Error: The mask results in an empty sequence.

getMask()

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

getMemberFromAll(...)

This method returns a tuple containing a member or members for each MeshFace in the MeshFaceArray.

Required arguments

member	A String specifying a MeshFace member name or a tuple of member names.

Optional arguments

None.

Return value

A tuple of member values (one for each MeshFace) or a tuple of tuples of member values (if you specified a tuple of member names in the *member* argument).

Exceptions

An exception occurs if no such member exists.	AttributeError: 'MeshFace' object has no attribute ''

Members

The MeshFaceArray object has no members.

MeshNode object

The MeshNode object refers to a node of a native mesh or an orphan mesh. A MeshNode object can be accessed via a part or part instance using an index that refers to the internal numbering of the node repository. The index does not refer to the node label.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].nodes[i]
mdb.models[name].parts[name].allInternalSurfaces[name].nodes[i]
mdb.models[name].parts[name].allSets[name].nodes[i]
mdb.models[name].parts[name].allSurfaces[name].nodes[i]
mdb.models[name].parts[name].nodes[i]
mdb.models[name].parts[name].retainedNodes[i]
mdb.models[name].parts[name].sets[name].nodes[i]
mdb.models[name].parts[name].surfaces[name].nodes[i]
import assembly
mdb.models[name].rootAssembly.allInstances[name].nodes[i]
mdb.models[name].rootAssembly.allInstances[name].sets[name].nodes[i]
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].nodes[i]
mdb.models[name].rootAssembly.allInternalSets[name].nodes[i]
mdb.models[name].rootAssembly.allInternalSurfaces[name].nodes[i]
mdb.models[name].rootAssembly.allSets[name].nodes[i]
mdb.models[name].rootAssembly.allSurfaces[name].nodes[i]
mdb.models[name].rootAssembly.instances[name].nodes[i]
mdb.models[name].rootAssembly.instances[name].sets[name].nodes[i]
mdb.models[name].rootAssembly.instances[name].surfaces[name].nodes[i]
mdb.models[name].rootAssembly.modelInstances[i].nodes[i]
mdb.models[name].rootAssembly.modelInstances[i].sets[name].nodes[i]
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]\
.nodes[i]
mdb.models[name].rootAssembly.nodes[i]
mdb.models[name].rootAssembly.sets[name].nodes[i]
mdb.models[name].rootAssembly.surfaces[name].nodes[i]
```

Node(...)

This method creates a node on an orphan mesh part.

Path

mdb.models[*name*].parts[*name*].Node

Required arguments

coordinates
A sequence of three Floats specifying the coordinates of the new node.

Optional arguments

localCsys
A DatumCsys object specifying the local coordinate system. If unspecified, the global coordinate system will be used.

label An Int specifying the node label.

Return value

A MeshNode object.

Exceptions

None.

`getElemEdges()`

This method returns a tuple of element edge objects that share the node.

Arguments

None.

Return value

A tuple of *MeshEdge* objects.

Exceptions

None.

`getElemFaces()`

This method returns a tuple of element face objects that share the node.

Arguments

None.

Return value

A tuple of *MeshFace* objects.

Exceptions

None.

`getElements()`

This method returns a tuple of element objects that share the node.

Arguments

None.

Return value

A tuple of *MeshElement* objects.

Exceptions

None.

getNodesByFeatureEdge(...)

This method returns an array of mesh node objects that are obtained by recursively finding adjacent nodes along a feature edge that are at an angle of less than or equal to the specified face angle.

Required arguments

angle A float specifying the value of the face angle in degrees.

Optional arguments

None.

Return value

A *MeshNodeArray* object, which is a sequence of MeshNode objects.

Exceptions

None.

setValues(...)

This method modifies the MeshNode object.

Required arguments

None.

Optional arguments

label An Int specifying the node label. This member may only be edited if the node belongs to an orphan mesh part. The specified label must be non-negative and must not be in use by any other node of the same part.

Return value

None.

Exceptions

None.

Members

The MeshNode object has the following members:

label An Int specifying the node label.

instanceName

A String specifying the name of the part instance that owns this node.

coordinates

A tuple of three Floats specifying the coordinates of the new node.

MeshNodeArray object

The MeshNodeArray is a sequence of *MeshNode* objects.

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name].nodes
mdb.models[name].parts[name].allInternalSurfaces[name].nodes
mdb.models[name].parts[name].allSets[name].nodes
mdb.models[name].parts[name].allSurfaces[name].nodes
mdb.models[name].parts[name].nodes
mdb.models[name].parts[name].retainedNodes
mdb.models[name].parts[name].sets[name].nodes
mdb.models[name].parts[name].surfaces[name].nodes
import assembly
mdb.models[name].rootAssembly.allInstances[name].nodes
mdb.models[name].rootAssembly.allInstances[name].sets[name].nodes
mdb.models[name].rootAssembly.allInstances[name].surfaces[name].nodes
mdb.models[name].rootAssembly.allInternalSets[name].nodes
mdb.models[name].rootAssembly.allInternalSurfaces[name].nodes
mdb.models[name].rootAssembly.allSets[name].nodes
mdb.models[name].rootAssembly.allSurfaces[name].nodes
mdb.models[name].rootAssembly.instances[name].nodes
mdb.models[name].rootAssembly.instances[name].sets[name].nodes
mdb.models[name].rootAssembly.instances[name].surfaces[name].nodes
mdb.models[name].rootAssembly.modelInstances[i].nodes
mdb.models[name].rootAssembly.modelInstances[i].sets[name].nodes
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name].nodes
mdb.models[name].rootAssembly.nodes
mdb.models[name].rootAssembly.sets[name].nodes
mdb.models[name].rootAssembly.surfaces[name].nodes
```

MeshNodeArray(...)

This method creates a MeshNodeArray object.

Path

mesh.MeshNodeArray

Required arguments

nodes A list of *MeshNode* objects.

Optional arguments

None.

Return value

A MeshNodeArray object.

Exceptions

None.

getFromLabel(...)

This method returns the object in the MeshNodeArray with the given label.

Required arguments

label An Int specifying the label of the object.

Optional arguments

None.

Return value

A *MeshNode* object.

Exceptions

None.

getSequenceFromMask(...)

This method returns the objects in the MeshNodeArray identified using the specified *mask*. This command is generated when the *JournalOptions* are set to COMPRESSEDINDEX. When a large number of objects are involved, this method is highly efficient.

Required arguments

mask A String specifying the object or objects.

Optional arguments

None.

Return value

A MeshNodeArray object.

Exceptions

None.

getMask()

This method returns a string specifying the object or objects.

Arguments

None.

Return value

A String specifying the object or objects.

Exceptions

None.

`getByBoundingBox(...)`

This method returns an array of nodes that lie within the specified bounding box.

Required arguments

None.

Optional arguments

xMin A float specifying the minimum X boundary of the bounding box.

yMin A float specifying the minimum Y boundary of the bounding box.

zMin A float specifying the minimum Z boundary of the bounding box.

xMax A float specifying the maximum X boundary of the bounding box.

yMax A float specifying the maximum Y boundary of the bounding box.

zMax A float specifying the maximum Z boundary of the bounding box.

Return value

A MeshNodeArray object, which is a sequence of *MeshNode* objects.

Exceptions

None.

`getByBoundingCylinder(...)`

This method returns an array of node objects that lie within the specified bounding cylinder.

Required arguments

center1 A tuple of the X-, Y-, and Z-coordinates of the center of the first end of the cylinder.

center2 A tuple of the X-, Y-, and Z-coordinates of the center of the second end of the cylinder.

radius A float specifying the radius of the cylinder.

Optional arguments

None.

Return value

A MeshNodeArray object, which is a sequence of *MeshNode* objects.

Exceptions

None.

`getByBoundingSphere(...)`

This method returns an array of node objects that lie within the specified bounding sphere.

Required arguments

center	A tuple of the X-, Y-, and Z-coordinates of the center of the sphere.
radius	A float specifying the radius of the sphere.

Optional arguments

None.

Return value

A MeshNodeArray object, which is a sequence of *MeshNode* objects.

Exceptions

None.

`getBoundingBox()`

This method returns a dictionary of two tuples representing minimum and maximum boundary values of the bounding box of the minimum size containing the node sequence.

Arguments

None.

Return value

A Dictionary object with the following items:

low: a tuple of three floats representing the minimum x, y, and z boundary values of the bounding box.

high: a tuple of three floats representing the maximum x, y, and z boundary values of the bounding box.

`getClosest()`

This method returns the node or nodes closest to the given point or set of points.

Required arguments

coordinates A point defined by x, y, and z values or a list of such points.

Optional arguments

numToFind The number of nodes to find for each given point. For example, if *numToFind* is 2, then the 2 closest points, if available and within *searchTolerance*, will be returned in order of proximity for each input point. The default is 1.

searchTolerance A float specifying a search radius for each point. By default, no search radius is defined, and all nodes in the sequence will be searched.

Return value

A MeshNode, or a list of MeshNode objects, or a list of lists of MeshNode objects, depending on the number of points given and the number of nodes requested.

`getMemberFromAll(...)`

This method returns a tuple containing a member or members for each MeshNode in the MeshNodeArray.

Required arguments

member A String specifying a MeshNode member name or a tuple of member names.

Optional arguments

None.

Return value

A tuple of member values (one for each MeshNode) or a tuple of tuples of member values (if you specified a tuple of member names in the *member* argument).

Exceptions

An exception occurs if no such member exists. AttributeError: 'MeshNode' object has no attribute "

`sequenceFromLabels(...)`

This method returns the objects in the MeshNodeArray identified using the specified labels.

Required arguments

labels A sequence of Ints specifying the labels.

Optional arguments

None.

Return value

A MeshNodeArray object.

Exceptions

An exception occurs if the resulting sequence is empty.	Error: The mask results in an empty sequence.

Members

The MeshNodeArray object has no members.

MeshStats object

The MeshStats object is a query object for holding mesh statistics and is returned by the `getMeshStats` command. The object does not have any methods.

Access

```
import mesh
```

Members

The MeshStats object has the following members:

numPointElems
An Int specifying the number of point elements.

numLineElems
An Int specifying the number of line elements.

numQuadElems
An Int specifying the number of quadrilateral elements.

numTriElems
An Int specifying the number of triangular elements.

numHexElems
An Int specifying the number of hexahedral elements.

numWedgeElems
An Int specifying the number of wedge elements.

numTetElems
An Int specifying the number of tetrahedral elements.

numPyramidElems
An Int specifying the number of pyramid elements.

numNodes
An Int specifying the number of nodes.

numMeshedRegions
An Int specifying the number of regions that contain a mesh.

Messaging commands

Messaging commands are available only if Abaqus/CAE is run interactively using the GUI.

In this section:

- *DataObject object*
- *MonitorMgr object*

DataObject object

An instance of the DataObject object is passed to each callback. The DataObject object has no methods. The members of a DataObject object depend on the type of the object. All DataObject instances have the following members, regardless of type:

- *clientHost*
- *clientName*
- *phase*
- *processId*
- *threadId*
- *timeStamp*

The possible DataObject types and the additional members for each type are as follows:

ABORTED

- *message*

COMPLETED

- *message*

END_STEP

- *stepId*

ERROR

- *message*

HEADING

- *heading*

MONITOR_DATA

- *dof*
- *node*
- *nset*
- *procedure*
- *time*
- *value*

ODB_FILE

- *file*

STARTED

- No additional members.

STATUS

- *attempts*
- *equilibrium*
- *increment*
- *iterations*
- *severe*
- *step*
- *stepTime*
- *timeIncrement*
- *totalTime*

STEP

- *stepId*
- *stepName*

WARNING

- *message*

Members

The DataObject object has the following members:

phase
A SymbolicConstant specifying the phase of the analysis. Possible values are BATCHPRE_PHASE, PACKAGER_PHASE, STANDARD_PHASE, EXPLICIT_PHASE, CALCULATOR_PHASE, and UNKNOWN_PHASE.

processId
An Int specifying the process ID of the analysis product.

threadId
An Int specifying the thread ID of the analysis product. Threads are used for parallel or multiprocessing; in most cases *threadId* is set to zero.

timeStamp
An Int specifying the time the message was sent in seconds since 00:00:00 UTC, January 1, 1970.

attempts An Int specifying the number of attempts made to reach equilibrium during this step.

dof An Int specifying the degree of freedom requested for monitoring the output.

equilibrium An Int specifying the number of equilibrium iterations made during this increment.

increment An Int specifying the increment of the analysis.

iterations An Int specifying the number of iterations in the step.

node An Int specifying the node number requested for monitoring output.

severe An Int specifying the number of severe discontinuity iterations completed during this increment.

step An Int specifying the current step number. Step number 1 corresponds to the first step.

stepId An Int specifying the ID of the step.

stepTime A Float specifying the step time corresponding to the current increment.

time A Float specifying the total time corresponding to the monitor data.

timeIncrement A Float specifying the time increment used in the current step.

totalTime A Float specifying the total time completed in the analysis.

value A Float specifying the current value of the degree of freedom requested for monitoring.

clientHost A String specifying the host name of the machine that is running the analysis.

clientName A String specifying the name of the client that responded to the callback function. Possible values are “BatchPre”, “Packager”, “Standard”, “Explicit”, and “Calculator”.

file A String specifying the full path of the output database.

heading A String specifying the job heading.

message A String specifying the job heading.

nset A String specifying the node set specified for monitoring output.

stepName A String specifying the name of the step.

MonitorMgr object

An instance of the MonitorMgr object is created when you import the abaqus module. No other instance of the MonitorMgr object is required. (This MonitorMgr object is not to be confused with the degree of freedom (DOF) monitor that is constructed from the *Step* object.)

Access

```
monitorManager
```

addMessageCallback(...)

This method specifies a callback function that will be called when the specified message is received from the analysis product.

For more information, see *An example of a callback function*.

Required arguments

jobName A String specifying the name of the job to be monitored or the SymbolicConstant ANY_JOB.

messageType A SymbolicConstant specifying which message type will call this callback. Possible values are:

- ABORTED
- ANY_JOB
- ANY_MESSAGE_TYPE
- COMPLETED
- END_STEP
- ERROR
- HEADING
- HEALER_JOB
- HEALER_TYPE
- INTERRUPTED
- ITERATION
- JOB_ABORTED
- JOB_COMPLETED
- JOB_INTERRUPTED
- JOB_SUBMITTED
- MONITOR_DATA
- ODB_FILE
- ODB_FRAME
- SIMULATION_ABORTED
- SIMULATION_COMPLETED
- SIMULATION_INTERRUPTED
- SIMULATION_SUBMITTED

- STARTED
- STATUS
- STEP
- WARNING

callback

A Python function to be called. The interface definition of the callback function is :

```
def onMessage(jobName, messageType, data, userData)
```

- *jobName* is a String.
- *messageType* is a SymbolicConstant with possible values as listed previously for the `addMessageCallback` method.
- *data* is a *DataObject* object.
- *userData* is the object passed as the *userData* argument to the `addMessageCallback` method.

Optional arguments

userData

Any Python object or `None`. This object is passed to the callback function.

Return value

`None`.

Exceptions

None.

`removeMessageCallback(...)`

This method removes a callback function. You specify the callback function to remove using the same arguments you used to add the callback.

Required arguments

jobName

A String specifying the name of the job to be monitored or the SymbolicConstant ANY_JOB.

messageType

A SymbolicConstant specifying which message type will call this callback. Possible values are:

- ABORTED
- ANY_JOB
- ANY_MESSAGE_TYPE
- COMPLETED
- END_STEP
- ERROR
- HEADING
- HEALER_JOB

- HEALER_TYPE
- INTERRUPTED
- ITERATION
- JOB_ABORTED
- JOB_COMPLETED
- JOB_INTERRUPTED
- JOB_SUBMITTED
- MONITOR_DATA
- ODB_FILE
- ODB_FRAME
- SIMULATION_ABORTED
- SIMULATION_COMPLETED
- SIMULATION_INTERRUPTED
- SIMULATION_SUBMITTED
- STARTED
- STATUS
- STEP
- WARNING

callback
A Python function to be called; it must be the same as the *callback* argument specified in the original call to `addMessageCallback`.

userData
Any Python object or `None`; it must be the same as the *userData* argument specified in the original call to `addMessageCallback`.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`checkMonitorStatus()`

This method raises a MonitorError exception if the monitoring status is not ENABLED.

Arguments

None.

Return value

None.

Exceptions

MonitorError:	Status is not ENABLED

Members

The MonitorMgr object has no members.

Model commands

Model commands are used to create Abaqus/CAE models.
A finished model contains all the data that Abaqus/CAE needs to create and submit an analysis to Abaqus/Standard or Abaqus/Explicit. Models are stored in a model database.

In this section:

- *Model object*
- *KeywordBlock object*

Model object

Abaqus creates a Model object named `Model-1` when a session is started.

Access

```
mdb.models[name]
```

Model(...)

This method creates a Model object.

Path

```
mdb.Model
```

Required arguments

name A String specifying the repository key.

Optional arguments

description A String specifying the purpose and contents of the Model object. The default value is an empty string.

stefanBoltzmann `None` or a Float specifying the Stefan-Boltzmann constant. The default value is `None`.

absoluteZero `None` or a Float specifying the absolute zero constant. The default value is `None`.

waveFormulation A SymbolicConstant specifying the type of incident wave formulation to be used in acoustic problems. Possible values are NOT_SET, SCATTERED, and TOTAL. The default value is NOT_SET.

modelType A SymbolicConstant specifying the analysis model type. Possible values are STANDARD_EXPLICIT and ELECTROMAGNETIC. The default is STANDARD_EXPLICIT.

universalGas `None` or a Float specifying the universal gas constant. The default value is `None`.

copyConstraints A boolean specifying whether to copy the constraints created in the model to the model that instances this model. The default value is ON.

copyConnectors A boolean specifying whether to copy the connectors created in the model to the model that instances this model. The default value is ON.

copyInteractions A boolean specifying whether to copy the interactions created in the model to the model that instances this model. The default value is ON.

Return value

A Model object.

Exceptions

None.

ModelFromInputFile(...)

This method creates a Model object by reading the keywords in an input file and creating the corresponding Abaqus/CAE objects.

Path

```
mdb.ModelFromInputFile
```

Required arguments

name
A String specifying the repository key.

inputFileName
A String specifying the name of the input file (including the `.inp` extension) to be parsed into the new model. This String can also be the full path to the input file if it is located in another directory.

Optional arguments

None.

Return value

A Model object.

Exceptions

None.

ModelFromOdbFile(...)

This method creates a Model object by reading an output database and creating any corresponding Abaqus/CAE objects.

Path

```
mdb.ModelFromOdbFile
```

Required arguments

name
A String specifying the repository key.

odbFileName
A String specifying the name of the output database file (including the `.odb` extension) to be read into the new model. This String can also be the full path to the output database file if it is located in another directory.

Optional arguments

None.

Return value

A Model object.

Exceptions

None.

ModelFromNastranFile(...)

This method creates a Model object by reading the keywords in a Nastran bulk data file or Nastran input file and creating any corresponding Abaqus/CAE objects. The default values is discussed in following and can be defined alternatively in the Abaqus environment file as the one used for the translator from Nastran to Abaqus. For more information, see *Translating Nastran Bulk Data in Text Files to Partial Abaqus Input Files*.

Path

```
mdb.ModelFromNastranFile
```

Required arguments

modelName
A String specifying the repository key.

inputFileName
A String specifying the name of the Nastran input file (including the `.bdf, .dat, .nas, .nastran, .blk, .bulk` extension) to be read into the new model. This String can also be the full path to the Nastran input file if it is located in another directory.

Optional arguments

sectionConsolidation
A SymbolicConstant specifying the method used to create shell section. Possible values are PRESERVE_SECTION, GROUP_BY_MATERIAL, and NONE. If PRESERVE_SECTION is used, an Abaqus section is created corresponding to each shell property ID. If GROUP_BY_MATERIAL is used, a single Abaqus section is created for all homogeneous elements referencing the same material. In both cases, material orientations and offsets are created using discrete fields. If NONE is used, a separate shell section is created for each combination of orientation, material offset, and/or thickness. The default is PRESERVE_SECTION.

preIntegratedShell
A Boolean specifying whether the pre-integrated shell section is created in default for shell element. The default value is OFF.

weightMassScaling
A Boolean specifying whether the value on the Nastran data line PARAM, WTMASS is used as a multiplier for all density, mass, and rotary inertia values created in the Abaqus input file. The default value is ON.

loadCases
A Boolean specifying whether each SUBCASE for linear static analyses is translated to a **LOAD CASE* option, and all such **LOAD CASE* options are grouped in a single **STEP* option. The default value is ON.

coupleBeamOffsets A Boolean specifying whether to translate the beam element connectivity to newly created nodes at the offset location and rigidly coupling the new and original nodes. If not, beam element offsets are translated to the **CENTROID* and **SHEAR CENTER* options, which are suboptions of the **BEAM GENERAL SECTION* option. The default value is ON. When the beam element references a PBARL or PBEAML property or if the beam offset has a significant component in the direction of the beam axis, the setting for this argument is always ON.

cbar A String specifying the 2-node beam that is created from CBAR and CBEAM elements. Possible values are B31 and B33. The default is B31.

cquad4 A String specifying the 4-node shell that is created from CQUAD4 elements. Possible values are S4 and S4R. The default is S4. If a reduced-integration element is chosen, the enhanced hourglass formulation is applied automatically.

chexa A String specifying the 8-node brick that is created from CHEXA elements. Possible values are C3D8I, C3D8 and C3D8R. The default is C3D8I. If a reduced-integration element is chosen, the enhanced hourglass formulation is applied automatically.

ctetra A String specifying the 10-node tetrahedron that is created from CTETRA elements. Possible values are C3D10 and C3D10M. The default is C3D10.

keepTranslatedFiles A Boolean specifying whether to keep the generated Abaqus input file after the model is created from the Nastran input file. The default value is ON.

Return value

A Model object.

Exceptions

None.

`setValues(...)`

This method modifies the Model object.

Required arguments

None.

Optional arguments

description A String specifying the purpose and contents of the Model object. The default value is an empty string.

noPartsInputFile A Boolean specifying whether an input file should be written without parts and assemblies. The default value is OFF.

absoluteZero `None` or a Float specifying the absolute zero constant. The default value is `None`.

stefanBoltzmann `None` or a Float specifying the Stefan-Boltzmann constant. The default value is `None`.

waveFormulation A SymbolicConstant specifying the type of incident wave formulation to be used in acoustic problems. Possible values are NOT_SET, SCATTERED, and TOTAL. The default value is NOT_SET.

universalGas `None` or a Float specifying the universal gas constant. The default value is `None`.

restartJob A String specifying the name of the job that generated the restart data.

restartStep A String specifying the name of the step where the restart analysis will start.

restartIncrement An Int specifying the increment, interval, iteration or cycle where the restart analysis will start. To select the end of the step use the SymbolicConstant STEP_END.

endRestartStep A Boolean specifying that the step specified by *restartStep* should be terminated at the increment specified by *restartIncrement*.

globalJob A String specifying the name of the job that generated the results for the global model.

shellToSolid A Boolean specifying that a shell global model drives a solid submodel.

copyConstraints A Boolean specifying whether to copy the constraints created in the model to the model that instances this model.

copyConnectors A Boolean specifying whether to copy the connectors created in the model to the model that instances this model

copyInteractions A Boolean specifying whether to copy the interactions created in the model to the model that instances this model.

Return value

`None`.

Exceptions

None.

Members

The Model object can have the following members:

name A String specifying the repository key.

stefanBoltzmann `None` or a Float specifying the Stefan-Boltzmann constant. The default value is `None`.

absoluteZero

`None` or a Float specifying the absolute zero constant. The default value is `None`.

waveFormulation

A SymbolicConstant specifying the type of incident wave formulation to be used in acoustic problems. Possible values are NOT_SET, SCATTERED, and TOTAL. The default value is NOT_SET.

universalGas

`None` or a Float specifying the universal gas constant. The default value is `None`.

noPartsInputFile

A Boolean specifying whether an input file should be written without parts and assemblies. The default value is OFF.

restartIncrement

An Int specifying the increment, interval, iteration or cycle where the restart analysis will start. To select the end of the step use the SymbolicConstant STEP_END.

endRestartStep

A Boolean specifying that the step specified by *restartStep* should be terminated at the increment specified by *restartIncrement*.

shellToSolid

A Boolean specifying that a shell global model drives a solid submodel.

lastChangedCount

A Float specifying the time stamp that indicates when the model was last changed.

description

A String specifying the purpose and contents of the Model object. The default value is an empty string.

restartJob

A String specifying the name of the job that generated the restart data.

restartStep

A String specifying the name of the step where the restart analysis will start.

globalJob

A String specifying the name of the job that generated the results for the global model.

copyConstraints

A boolean specifying the status of constraints created in a model, in the model which instances this model.

copyConnectors

A boolean specifying the status of connectors created in a model, in the model which instances this model.

copyInteractions

A boolean specifying the status of interactions created in a model, in the model which instances this model.

keywordBlock

A *KeywordBlock* object.

rootAssembly

An *Assembly* object.

amplitudes A repository of *Amplitude* objects.

profiles A repository of *Profile* objects.

boundaryConditions A repository of *BoundaryCondition* objects.

constraints A repository of *Constraint* objects.

analyticalFields A repository of *AnalyticalField* objects.

discreteFields A repository of *DiscreteField* objects.

predefinedFields A repository of *PredefinedField* objects.

interactions A repository of *Interaction* objects.

interactionProperties A repository of *InteractionProperty* objects.

contactControls A repository of *ContactControl* objects.

contactInitializations A repository of *ContactInitialization* objects.

contactStabilizations A repository of *ContactStabilization* objects.

linkedInstances A tuple of tuples of Strings specifying the linked child PartInstance name in the current model to the corresponding parent PartInstance name in a different model.

linkedParts A tuple of tuples of Strings specifying the linked child Part name in the current model to the corresponding parent Part name in a different model.

loads A repository of *Load* objects.

materials A repository of *Material* objects.

calibrations A repository of *Calibration* objects.

sections A repository of *Section* objects.

remeshingRules A repository of *RemeshingRule* objects.

sketches A repository of *ConstrainedSketch* objects.

parts A repository of *Part* objects.

steps A repository of *Step* objects.

featureOptions	A *FeatureOptions* object.
adaptiveMeshConstraints	A repository of *AdaptiveMeshConstraint* objects.
adaptiveMeshControls	A repository of *AdaptiveMeshControl* objects.
timePoints	A repository of *TimePoint* objects.
filters	A repository of *Filter* objects.
integratedOutputSections	A repository of *IntegratedOutputSection* objects.
fieldOutputRequests	A repository of *FieldOutputRequest* objects.
historyOutputRequests	A repository of *HistoryOutputRequest* objects.
optimizationTasks	A repository of *OptimizationTask* objects.
tableCollections	A repository of *TableCollection* objects.
eventSeriesTypes	A repository of *EventSeriesType* objects.
eventSeriesDatas	A repository of *EventSeriesData* objects.

Corresponding analysis keywords

- **PHYSICAL CONSTANTS*

KeywordBlock object

The KeywordBlock object contains a representation of its model in the Abaqus input file format. You may edit the contents of the KeywordBlock to add solver functionality that is not supported by Abaqus/CAE. As a general rule, edits to the KeywordBlock object should be made as the last step prior to writing the actual Abaqus input file, thus avoiding possible conflicts with changes made using other MDB commands. The KeywordBlock object has no constructor. A KeywordBlock object is created when you create a model object. A model object contains only one KeywordBlock object.

Access

```
mdb.models[name].keywordBlock
```

`setValues(...)`

This method modifies the KeywordBlock object.

Required arguments

None.

Optional arguments

edited A Boolean specifying whether this objects *sieBlocks* member has been edited. Setting edited=False will set the *sieBlocks* member to an empty tuple, thereby discarding all previous edits.

Return value

`None.`

Exceptions

None.

`insert(...)`

This method inserts a String at a specified position in the *sieBlocks* member.

Required arguments

position An Int specifying the position in the *sieBlocks* member after which the new string will be inserted.

text A String specifying the text to be inserted. The text represents an Abaqus input file keyword and its associated data

Optional arguments

None.

Return value
`None.`

Exceptions
IndexError.

`replace(...)`

This method replaces a String at a specified position in the *sieBlocks* member.

Required arguments

position An Int specifying the position of the String to be replaced in the *sieBlocks* member.

text A String specifying the text to be replaced. The text represents an Abaqus input file keyword and its associated data.

Optional arguments
None.

Return value
`None.`

Exceptions
IndexError.

`synchVersions(...)`

This method synchronizes, or merges, the edits made in this object with those made in the model using other scripting commands or the user interface. The `synchVersions` method updates the *sieBlocks* member. The *sieBlocks* member is empty prior to the first call to `synchVersions`. As a side effect, `synchVersions` sets *lastSynchCount* to the current value of the counter associated with the *Mdb* object, which is used to determine if synchronization is necessary.

Required arguments

storeNodesAndElements A Boolean specifying whether the nodal coordinates and element connectivities (i.e. the data lines for the *NODE and *ELEMENT keyword blocks) are to be stored in the *sieBlocks* member. All other keywords and their data lines are always stored. The default value is True. If *storeNodesAndElements* is True, the size of the keywordBlock data will be similar to that of the input file. Since the KeywordBlock is stored in the Abaqus/CAE database, this will result in a larger database. It will also result in a slower execution of the `synchVersions` command. If *storeNodesAndElements* is False, the data lines are not stored in *sieBlocks*. Consequently, only set *storeNodesAndElements*=True if you wish to make changes to the *NODE or *ELEMENT data lines themselves. If your task is limited to reading nodal coordinates and element connectivities (i.e. not editing this information) then it is generally better to access this information from other parts of the *Mdb*.

Optional arguments

None.

Return value

`None.`

Exceptions

IndexError.

Members

The KeywordBlock object has the following members:

edited

A Boolean specifying whether the Keywords Editor has been used to change the model.

lastSynchCount

A Float specifying the value of the counter associated with the *Mdb* object at the most recent synchronization.

sieBlocks

A tuple of Strings specifying a sequence of Strings that is identical to the information written to the Abaqus input file. Each String in the sequence represents an Abaqus input file keyword along with the parameters and data lines associated with the keyword. A String can also be a comment in the input file. You initialize this data member by calling `synchVersions`. After you initialize the data member, you use calls to `replace` and `insert` to record your edits in the correct location. If the last call to `synchVersions` used the argument *storeNodesAndElements*=False, the entry for the keywords **NODE* and **ELEMENT* will contain only the keyword and its parameters, not the data lines.

Odb commands

The Python ODB API commands are used to read and write data from an output database (`.odb`) file. The path to the Odb object can be via the session.odbs repository or via a variable. In this chapter the `Access` and `Path` statements refer to a variable called `odb` that represents an existing Odb object.

In this section:

- *Odb object*
- *AnalyticSurface object*
- *AnalyticSurfaceSegment object*
- *BeamOrientation object*
- *FieldBulkData object*
- *FieldLocation object*
- *FieldOutput object*
- *FieldValue object*
- *HistoryOutput object*
- *HistoryPoint object*
- *HistoryRegion object*
- *JobData object*
- *OdbAssembly object*
- *OdbDatumCsys object*
- *OdbFrame object*
- *OdbInstance object*
- *OdbLoadCase object*
- *OdbMeshElementArray object*
- *OdbMeshElement object*
- *OdbMeshNodeArray object*
- *OdbMeshNode object*
- *OdbPart object*
- *OdbPretensionSection object*
- *OdbRigidBody object*
- *OdbSequenceAnalyticSurfaceSegment object*
- *OdbSet object*
- *OdbStep object*
- *RebarOrientation object*
- *ScratchOdb object*
- *SectionCategory object*
- *SectionPoint object*
- *SectorDefinition object*
- *UserData object*
- *Odb commands*

Odb object

The Odb object is the in-memory representation of an output database (ODB) file.

Access

```
import odbAccess
session.odbs[name]
```

Odb(...)

This method creates a new Odb object.

Path

```
session.Odb
```

Required arguments

name A String specifying the repository key.

Optional arguments

analysisTitle A String specifying the title of the output database. The default value is an empty string.

description A String specifying the description of the output database. The default value is an empty string.

path A String specifying the path to the file where the new output database (`.odb`) file will be written. The default value is an empty string.

Return value

An Odb object.

Exceptions

None.

`close()`

This method closes an output database.

Arguments

None.

Return value

`None.`

Exceptions

None.

getFrame(...)

This method returns the frame at the specified time, frequency, or mode. It will not interpolate values between frames. The method is not applicable to an Odb object containing steps with different domains or to an Odb object containing a step with load case specific data.

Required arguments

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the total time or frequency.

Optional arguments

match A SymbolicConstant specifying which frame to return if there is no frame at the exact frame value. Possible values are CLOSEST, BEFORE, AFTER, and EXACT. The default value is CLOSEST.

When *match*=CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the exact frame is not found: OdbError: Frame not found.

save()

This method saves output to an output database (`.odb`) file.

Arguments

None.

Return value

`None.`

Exceptions

OdbError Database save failed. The database was opened as read-only. Modification of data is not permitted.

update()

This method is used to update an Odb object in memory while an Abaqus analysis writes data to the associated output database. `update` checks if additional steps have been written to the output database since it was opened or last updated. If additional steps have been written to the output database, `update` adds them to the Odb object.

Arguments

None.

Return value

A Boolean specifying whether additional steps or frames were added to the Odb object.

Exceptions

None.

Members

The Odb object has members with the same names and descriptions as the arguments to the *Odb* method.

In addition, the Odb object can have the following members:

isReadOnly	A Boolean specifying whether the output database was opened with read-only access.
amplitudes	A repository of *Amplitude* objects.
filters	A repository of *Filter* objects.
rootAssembly	An *OdbAssembly* object.
jobData	A *JobData* object.
parts	A repository of *OdbPart* objects.
materials	A repository of *Material* objects.
steps	A repository of *OdbStep* objects.
sections	A repository of *Section* objects.
sectionCategories	A repository of *SectionCategory* objects.
sectorDefinition	A *SectorDefinition* object.
userData	A *UserData* object.
customData	A *RepositorySupport* object.

profiles

A repository of *Profile* objects.

AnalyticSurface object

The AnalyticSurface object is a geometric surface that can be described with straight and/or curved line segments.

Access

```
import odbAccess
session.odbs[name].parts[name].analyticSurface
session.odbs[name].rootAssembly.instances[name].analyticSurface
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.analyticSurface
```

Members

The AnalyticSurface object has the following members:

name

A String specifying the name of the analytic surface.

type

A SymbolicConstant specifying the type of AnalyticSurface object. Possible values are SEGMENTS, CYLINDER, and REVOLUTION.

filletRadius

A Float specifying radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

segments

An *OdbSequenceAnalyticSurfaceSegment* object specifying the profile associated with the surface.

localCoordData

A tuple of tuples of Floats specifying the global coordinates of points representing the local coordinate system, if used.

AnalyticSurfaceSegment object

An individual segment of the analytic surface.

Access

```
import odbAccess
session.odbs[name].parts[name].analyticSurface.segments[i]
session.odbs[name].rootAssembly.instances[name].analyticSurface\
.segments[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.analyticSurface.segments[i]
```

AnalyticSurfaceSegment(...)

This method creates an AnalyticSurfaceSegment object.

Path

odbAccess.AnalyticSurfaceSegment

Required arguments

type A SymbolicConstant specifying the type of AnalyticSurfaceSegment. Possible values are START, LINE, CIRCLE, and PARABOLA.

data A sequence of sequences of Floats specifying the coordinates of point/s representing the segment of the *AnalyticSurface* object. If *type*=CIRCLE, the first row contains coordinates of the end point and the second row contains coordinates of the center point. If *type*=PARABOLA, the first row contains coordinates of the middle point and the second row contains coordinates of the end point. If *type*=START or *type*=LINE, a single row contains coordinates of the start/end point.

Optional arguments

None.

Return value

An AnalyticSurfaceSegment object.

Exceptions

None.

Members

The AnalyticSurfaceSegment object has members with the same names and descriptions as the arguments to the *AnalyticSurfaceSegment* method.

BeamOrientation object

The BeamOrientation object represents the direction of the first beam section axis $\mathbf{n_1}$. Specifying the beam orientation using an additional node in the element connectivity list is not supported.

Access

```
import odbAccess
session.odbs[name].parts[name].beamOrientations[i]
session.odbs[name].rootAssembly.instances[name].beamOrientations[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.beamOrientations[i]
```

Members

The BeamOrientation object can have the following members:

method

A SymbolicConstant specifying the orientation assignment method. Possible values are N1_COSINES, CSYS, and VECT.

region

An *OdbSet* object specifying a region for which the beam orientation is defined.

vector

A tuple of Floats specifying direction cosines of the $\mathbf{n_1}$-direction of the beam cross-section.

FieldBulkData object

The FieldBulkData object represents the entire field data for a class of elements or nodes. All elements in a class correspond to the same element type and material.

Access

```
import odbAccess
session.odbs[name].steps[name].frames[i].fieldOutputs[name]\
.bulkDataBlocks[i]
```

Members

The FieldBulkData object can have the following members:

position

A SymbolicConstant specifying the position of the output in the element. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- ELEMENT_FACE.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

type

A SymbolicConstant specifying the output type. Possible values are SCALAR, VECTOR, TENSOR_3D_FULL, TENSOR_3D_PLANAR, TENSOR_3D_SURFACE, TENSOR_2D_PLANAR, and TENSOR_2D_SURFACE.

instance

An *OdbInstance* object specifying the part to which the labels belong.

sectionPoint

A *SectionPoint* object specifying the section point number of the current block of data.

elementLabels

A sequence of Ints specifying the element labels of the elements in the block. *elementLabels* is valid only if *position*=INTEGRATION_POINT, CENTROID, ELEMENT_NODAL, or ELEMENT_FACE.

nodeLabels

A sequence of Ints specifying the node labels of the nodes in the block. *nodelabels* is valid only if *position*=ELEMENT_NODAL or NODAL.

componentLabels

A sequence of Strings specifying the component labels.

integrationPoints

A sequence of Ints specifying the integration points in the elements in the block. *integrationPoints* is available only if *position*=INTEGRATION_POINT.

data

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *data* is a sequence containing the components for each element or node in the block. If the underlying data are in double precision, an exception will be thrown.

conjugateData

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *conjugateData* is a sequence containing the imaginary part of the components for each element or node in the block. If the underlying data are in double precision, an exception will be thrown.

mises

A sequence of Floats specifying the calculated von Mises stress at each output location in the block of element data, or NULL. The value is valid only when the *validInvariants* member includes MISES; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

localCoordSystem

A pointer to an array of Floats specifying the quaternion representing the local coordinate system (the rotation from global to local) at each output location. The quaternion is returned in the form

$q = (\mathbf{q}, q_0)$,, which is the reverse of that shown in *Rotation variables*. *localCoordSystem* is available for TENSOR data written in a local coordinate system. It is also available for VECTOR data for connector element outputs. For connector element outputs the quaternion form is

$q = (q_0, \mathbf{q})$, which represents the rotation from local to global. If the underlying data are in double precision, an exception will be thrown.

FieldLocation object

The FieldLocation object specifies locations for which data are available in the field. For example, a displacement field will have a FieldLocation object with a *position* member value of NODAL. The FieldLocation object has no constructor; it is created automatically as an element of the *location* member of a *FieldOutput* object by the `addData` method of a *FieldOutput* object.

Access

```
import odbAccess
session.odbs[name].steps[name].frames[i].fieldOutputs[name]\
.locations[i]
```

Members

The FieldLocation object can have the following members:

position

A SymbolicConstant specifying the position of the output in the element. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- ELEMENT_FACE.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

sectionPoints

A *SectionPointArray* object.

FieldOutput object

A FieldOutput object contains field data for a specific output variable.

Access

```
import
odbAccess
session.odbs[name].steps[name].frames[i].fieldOutputs[name]
```

FieldOutput(...)

This method creates a FieldOutput object.

Path

```
session.odbs[name].steps[name].frames[i].FieldOutput
```

Required arguments

name
A String specifying the output variable name.

description
A String specifying the output variable. Colon (:) should not be used as a part of the field output description.

type
A SymbolicConstant specifying the output type. Possible values are SCALAR, VECTOR, TENSOR_3D_FULL, TENSOR_3D_PLANAR, TENSOR_3D_SURFACE, TENSOR_2D_PLANAR, and TENSOR_2D_SURFACE.

Optional arguments

componentLabels
A sequence of Strings specifying the labels for each component of the value. The length of the sequence must match the type. If *type*=TENSOR, the default value is *name* with the suffixes ('11', '22', '33', '12', '13', '23'). If *type*=VECTOR, the default value is *name* with the suffixes ('1', '2', '3'). If *type*=SCALAR, the default value is an empty sequence.

validInvariants
A sequence of SymbolicConstants specifying which invariants should be calculated for this field. An empty sequence indicates that no invariants are valid for this field. Possible values are:

- MAGNITUDE
- MISES
- TRESCA
- PRESS
- INV3
- MAX_PRINCIPAL
- MID_PRINCIPAL
- MIN_PRINCIPAL

- MAX_INPLANE_PRINCIPAL
- MIN_INPLANE_PRINCIPAL
- OUTOFPLANE_PRINCIPAL

The default value is an empty sequence.

isEngineeringTensor A Boolean specifying whether the field is an engineering tensor or not. Setting isEngineeringTensor to true makes a tensor field behave as a strain like quantity where the off-diagonal components of tensor are halved for invariants computation. This parameter applies only to tensor field outputs. The default value is OFF.

Return value

A FieldOutput object.

Exceptions

None.

FieldOutput(...)

This method creates a FieldOutput object from an existing FieldOutput object of the same output database.

Path

```
session.odbs[name].steps[name].frames[i].FieldOutput
```

Required arguments

field A FieldOutput object.

Optional arguments

name A String specifying the name of the FieldOutput object.

description A String specifying the output variable. Colon (:) should not be used as a part of the field output description.

Return value

A FieldOutput object.

Exceptions

None.

addData(...)

This method adds data to a FieldOutput object.

Required arguments

position A SymbolicConstant specifying the position of the output. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

instance An *OdbInstance* object specifying the namespace for labels.

labels A sequence of Ints specifying the labels of the nodes or elements where the values in *data* are located. For better performance, the node or element labels are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data A sequence of sequences of Floats specifying the data values for the specified *position*, *instance*, and *labels*. The values must be given in the correct order. Element nodal data follow the order of nodal connectivity defined in the Abaqus documentation. Integration point data follow the order defined in the Abaqus documentation. Section point data for beams and shells follow the convention given in the Abaqus documentation. For more information, see the *Abaqus Elements Guide*. These data create *FieldValue* objects internally.

Optional arguments

sectionPoint A *SectionPoint* object specifying the location in the section. Although *sectionPoint* is an optional argument to the `addData` method, omitting the argument does have consequences for visualization. If you omit the argument when you are writing field output data for a shell or a beam, you cannot subsequently select the section point to display when you are displaying the field output data using the Visualization module.

localCoordSystem The *localCoordSystem* parameter can be specified using either of the following:

- A sequence of sequences of Floats specifying the 3 × 3 matrix of direction cosines of the local coordinate system. This argument is available only for fields with type=TENSOR or VECTOR.
- A sequence of matrices of floats specifying the direction cosines of the local coordinates systems, where the sequence is the same length as *data*. If *localCoordSystem* is a matrix, a different local coordinate system applies to each data value.

User supplied values of localCoordSystem are transposed before storing in the database.

Return value

`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds the data from a field created using the `getSubset` method and mathematical operators to the database. The user must create a field to contain the new data and then use the `addData` method to assign the data from the fields.

Required arguments

field A FieldOutput object specifying the data to add.

Optional arguments

None.

Return value

`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds data to a FieldOutput object.

Required arguments

position A SymbolicConstant specifying the position of the output. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.
- ELEMENT_FACE_INTEGRATION_POINT, specifying the values calculated at the element face integration points.
- SURFACE_INTEGRATION_POINT, specifying the values calculated at the surface integration points. Selecting this value prompts the Visualization module to calculate the sum of the values at the ELEMENT_FACE_INTEGRATION_POINT position from multiple surfaces.

set

An *OdbSet* object specifying the instance-level set defining the region for `addData`. The set must be defined in the same output database as the output database into which the new field output data is being written. For better performance, the node or element labels in the set are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data

A sequence of sequences of Floats specifying the data values for the specified position and labels in the set. Each row of data provides the value at one unique position. The width of each row should match the number of required components for the data. The values must be given in the order that matches the ordering of labels in the set.

The order of the element nodal data, integration point data, and section point data for beams and shells follows the conventions defined in the *Abaqus Elements Guide*.

Optional arguments

sectionPoint

A *SectionPoint* object specifying the location in the section. Although*sectionPoint* is an optional argument to the`addData` method, omitting the argument does have consequences for visualization. If you omit the argument when you are writing field output data for a shell or a beam, you cannot subsequently select the section point to display when you are displaying the field output data using the Visualization module.

conjugateData

An odb_SequenceSequenceFloat object specifying the imaginary data values for the specified position, instance, and labels. You must provide this data when you add complex fields to the output database. The order of the values follows the conventions defined in the *Abaqus Elements Guide*.

Return value

`None.`

Exceptions

If you specify an odbSet containing entities from multiple instances:	odbException: Entities from multiple instances present in set.
The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`getScalarField(...)`

This method generates a scalar field containing the extracted component or calculated invariant values. The new field will hold values for the same nodes or elements as the parent field. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Required arguments

invariant

A SymbolicConstant specifying the invariant. Possible values areMAGNITUDE, MISES, TRESCA, PRESS, INV3, MAX_PRINCIPAL, MID_PRINCIPAL, MIN_PRINCIPAL, MAX_INPLANE_PRINCIPAL, MIN_INPLANE_PRINCIPAL, and OUTOFPLANE_PRINCIPAL.

Optional arguments

None.

Return value

AFieldOutput object.

Exceptions

None.

getScalarField(...)

This method generates a scalar field containing the extracted component or calculated invariant values. The new field will hold values for the same nodes or elements as the parent field. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Required arguments

componentLabel A String specifying the component label, such as “S11”.

Optional arguments

None.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

position A SymbolicConstant specifying the position of the output in the element. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

If the requested field values are not found in the output database at the specified ELEMENT_NODAL or CENTROID positions, they are extrapolated from the field data at the INTEGRATION_POINT position for the entire field region. If the field values are found at the specified positions, only these field values are returned without any extrapolation. This could potentially be only for a subset of the field region, depending on the output request.

readOnly

A Boolean specifying whether the extrapolated data returned by this call is written to the output database. The default value is OFF.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

region

An *OdbSet* specifying the region for which to extract values. For better performance, the node or element labels in the sets are preferred to be sorted in ascending order.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

localCoordSystem

A sequence of sequences of Floats specifying the 3 × 3 matrix of direction cosines. Field values associated with the supplied coordinate system will be extracted.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

sectionPoint A *SectionPoint* object.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

location A *FieldLocation* object.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

region An *OdbMeshElement* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

region An *OdbMeshNode* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

region An *OdbInstance* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Required arguments

None.

Optional arguments

elementType A String specifying the element type for which to extract values. The string must correspond to a valid Abaqus element type.

Return value

A FieldOutput object.

Exceptions

None.

getTransformedField(...)

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *DatumCsys* object. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Required arguments

datumCsys A valid *DatumCsys* object designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

projected22Axis An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.

projectionTol A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes.	odbException: Cannot apply transformation to field containing assembly level nodes.

`getTransformedField(...)`

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *DatumCsys* object. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Required arguments

datumCsys	A valid *DatumCsys* object designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

deformationField	A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.
projected22Axis	An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.
projectionTol	A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes.	odbException: Cannot apply transformation to field containing assembly level nodes.

`getTransformedField(...)`

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *DatumCsys* object. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Required arguments

datumCsys A valid *DatumCsys* object designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

deformationField A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.

rotationField A FieldOutput object specifying the nodal rotational displacement vectors required by moving coordinate systems that follow a 6-dof node, to determine instantaneous configurations.

projected22Axis An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.

projectionTol A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes.	odbException: Cannot apply transformation to field containing assembly level nodes.

`getConnectorFieldXformedToNodeA(...)`

This method generates a new vector field containing the transformed component values of the parent connector field to the node A coordinate system. The new field will hold values for the same connector elements as the parent field. Some connection types such as Axial, Link, Slip Ring, and Radial Thrust require that the deformationField be specified.

Required arguments

None.

Optional arguments

deformationField A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.

Return value

A FieldOutput object.

Exceptions

The `getConnectorFieldXformedToNodeA` method throws an exception if the field requires a deformationField and the argument is not supplied.	odbException: Deformation field is required for transforming this connector field.

`setComponentLabels(...)`

This method sets the component labels for the FieldOutput object.

Required arguments

componentLabels A sequence of Strings specifying the labels for each component of the value. The length of the sequence must match the type. If *type*=TENSOR, the default value is *name* with the suffixes ('11', '22', '33', '12', '13', '23'). If *type*=VECTOR, the default value is *name* with the suffixes ('1', '2', '3'). If *type*=SCALAR, the default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`setDataType(...)`

This method sets the data type of a FieldOutput object.

Required arguments

type A SymbolicConstant specifying the output type. Possible values are SCALAR, VECTOR, TENSOR_3D_FULL, TENSOR_3D_PLANAR, TENSOR_3D_SURFACE, TENSOR_2D_PLANAR, and TENSOR_2D_SURFACE.

Return value

`None.`

Exceptions

None.

`setValidInvariants(...)`

This method sets the invariants valid for the FieldOutput object.

Required arguments

validInvariants A sequence of SymbolicConstants specifying which invariants should be calculated for this field. An empty sequence indicates that no invariants are valid for this field. Possible values are:

- MAGNITUDE

- MISES
- TRESCA
- PRESS
- INV3
- MAX_PRINCIPAL
- MID_PRINCIPAL
- MIN_PRINCIPAL
- MAX_INPLANE_PRINCIPAL
- MIN_INPLANE_PRINCIPAL
- OUTOFPLANE_PRINCIPAL

The default value is an empty sequence.

Return value

`None.`

Exceptions

None.

Members

The FieldOutput object has members with the same names and descriptions as the arguments to the *FieldOutput* method.

In addition, the FieldOutput object can have the following members:

dim

An Int specifying the dimension of vector or the first dimension (number of rows) of matrix.

dim2

An Int specifying the second dimension (number of columns) of matrix.

isComplex

A Boolean specifying whether the data are complex.

locations

A *FieldLocationArray* object.

values

A *FieldValueArray* object specifying the order of the objects in the array is determined by the Abaqus Scripting Interface; see the *data* argument to the `addData` method for a description of the order.

FieldValue object

The FieldValue object represents the field data at a point. The FieldValue object has no constructor; it is created by the *Odb* object when data are added to the *FieldOutput* object using the `addData` method. For faster, bulk-data access, see *Using bulk data access to an output database*.

Access

```
import odbAccess
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]
```

Members

The FieldValue object has the following members:

position

A SymbolicConstant specifying the position of the output in the element. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- ELEMENT_FACE, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element.
- CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

precision

A SymbolicConstant specifying the precision of the output in the element. Possible values are:

- SINGLE_PRECISION, specifying that the output values are in single precision.
- DOUBLE_PRECISION, specifying that the output values are in double precision.

elementLabel

An Int specifying the element label of the element containing the location. *elementLabel* is available only if *position*=INTEGRATION_POINT, CENTROID, ELEMENT_NODAL, or ELEMENT_FACE.

nodeLabel

An Int specifying the node label of the node containing the location. *nodelabel* is available only if *position*=ELEMENT_NODAL or NODAL.

integrationPoint

An Int specifying the integration point in the element. *integrationPoint* is available only if *position*=INTEGRATION_POINT.

face

A SymbolicConstant specifying the face of the element. *face* is available only if *position*=ELEMENT_FACE.

type
A SymbolicConstant specifying the output type. Possible values are SCALAR, VECTOR, TENSOR_3D_FULL, TENSOR_3D_PLANAR, TENSOR_3D_SURFACE, TENSOR_2D_PLANAR, and TENSOR_2D_SURFACE.

magnitude
A Float specifying the length or magnitude of the vector. *magnitude* is valid only when *type*=VECTOR.

mises
A Float specifying the calculated von Mises stress. The value is valid only when the *validInvariants* member includes MISES; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

tresca
A Float specifying the calculated Tresca stress. The value is valid only when the *validInvariants* member includes TRESCA; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

press
A Float specifying the calculated pressure stress. The value is valid only when the *validInvariants* member includes PRESS; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

inv3
A Float specifying the calculated third stress invariant. The value is valid only when the *validInvariants* member includes INV3; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

maxPrincipal
A Float specifying the calculated maximum principal stress. The value is valid only when the *validInvariants* member includes MAX_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

midPrincipal
A Float specifying the calculated intermediate principal stress. The value is valid only when the *validInvariants* member includes MID_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

minPrincipal
A Float specifying the minimum principal stress. The value is valid only when the *validInvariants* member includes MIN_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

maxInPlanePrincipal
A Float specifying the maximum principal in-plane stress. The value is valid only when the *validInvariants* member includes MAX_INPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

minInPlanePrincipal
A Float specifying the calculated minimum principal in-plane stress. The value is valid only when the *validInvariants* member includes MIN_INPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

outOfPlanePrincipal
A Float specifying the calculated principal out-of-plane stress. The value is valid only when the *validInvariants* member includes OUTOFPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

instance
An *OdbInstance* object specifying the part to which the labels belong.

sectionPoint

A *SectionPoint* object.

localCoordSystem

A tuple of tuples of Floats specifying the 3 × 3 matrix of Floats specifying the direction cosines of the local coordinate system (the rotation from global to local). Each sequence represents a row in the direction cosine matrix. *localCoordSystem* is available for TENSOR data written in a local coordinate system. It is also available for VECTOR data for connector element outputs. For connector element outputs the rotation is from local to global. If the underlying data are in double precision, an exception will be thrown.

localCoordSystemDouble

A tuple of tuples of Floats specifying the 3 × 3 matrix of Doubles specifying the direction cosines of the local coordinate system (the rotation from global to local). Each sequence represents a row in the direction cosine matrix. *localCoordSystemDouble* is available for TENSOR data written in a local coordinate system. It is also available for VECTOR data for connector element outputs. For connector element outputs the rotation is from local to global. If the underlying data are in single precision, an exception will be thrown.

data

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *data* is a sequence containing the components. If the underlying data are in double precision an exception will be thrown.

dataDouble

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *data* is a sequence containing the components. If the underlying data are in single precision, an exception will be thrown.

conjugateData

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *conjugateData* is a sequence containing the components. If the underlying data are in double precision, an exception will be thrown.

conjugateDataDouble

A tuple of Floats specifying data in the form described by *type*. If *type*=TENSOR or VECTOR, *conjugateData* is a sequence containing the components. If the underlying data are in single precision, an exception will be thrown.

HistoryOutput object

The HistoryOutput object contains the history output at a point for the specified variable.

Access

```
import odbAccess
session.odbs[name].steps[name].historyRegions[name]\
.historyOutputs[name]
```

HistoryOutput(...)

This method creates a HistoryOutput object.

Path

```
session.odbs[name].steps[name].historyRegions[name].HistoryOutput
```

Required arguments

name A String specifying the output variable name.

description A String specifying the output variable.

type A SymbolicConstant specifying the output type. Only SCALAR is currently supported.

Optional arguments

validInvariants A sequence of SymbolicConstants specifying which invariants should be calculated for this field. Possible values are MAGNITUDE, MISES, TRESCA, PRESS, INV3, MAX_PRINCIPAL, MID_PRINCIPAL, and MIN_PRINCIPAL. The default value is an empty sequence.

Return value

A HistoryOutput object.

Exceptions

None.

addData(...)

This method adds data to the *data* member of the HistoryOutput object.

Required arguments

frame A Double specifying the frame value. *frame* can be specified in step time, frequency, or mode number.

value A Double specifying the value of the variable at the frame value specified in *frame*.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`addData(...)`

This method adds data to the *data* member of the HistoryOutput object.

Required arguments

frame A sequence of Floats specifying the frame values. *frame* can be specified in step time, frequency, or mode number.

value A sequence of Floats specifying the value of the variable at the frame values specified in *frame*.

Optional arguments

None.

Return value

`None.`

Exceptions

If the length of *frame* is not the same as the length of *value* a ValueError is raised.

`addData(...)`

This method adds data to the *data* member of the HistoryOutput object.

Required arguments

data A sequence of pairs of Floats specifying the pairs (*frameValue*, *value*) where *frameValue* is either time, frequency, or mode and *value* is the value of the specified variable at *frameValue*. (This value depends on the type of the variable.)

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The HistoryOutput object has members with the same names and descriptions as the arguments to the *HistoryOutput* method.

In addition, the HistoryOutput object has the following members:

data

A tuple of pairs of Floats specifying the pairs (*frameValue*, *value*) where *frameValue* is either time, frequency, or mode and *value* is the value of the specified variable at *frameValue*. (This value depends on the type of the variable.)

conjugateData

A tuple of pairs of Floats specifying the imaginary portion of a specified complex variable at each frame value (time, frequency, or mode). The pairs have the form (*frameValue*, *value*).

HistoryPoint object

The HistoryPoint object specifies the point at which history data will be collected.
The HistoryPoint object is a temporary object used as an argument to the `HistoryRegion` method.

Access

```
import odbAccess
session.odbs[name].steps[name].historyRegions[name].point
```

HistoryPoint(...)

This method creates a HistoryPoint object for a node.

Path

```
odbAccess.HistoryPoint
```

Required arguments

node An *OdbMeshNode* object specifying the node for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for an element.

Path

```
odbAccess.HistoryPoint
```

Required arguments

element An *OdbMeshElement* object specifying the element for which the data are to be collected.

Optional arguments

ipNumber An Int specifying the integration point. This argument is used to define a history output position of INTEGRATION_POINT or ELEMENT_FACE_INTEGRATION_POINT. The default value is 0.

sectionPoint A *SectionPoint* object.

face

A SymbolicConstant specifying the element face. This argument is used to define a history output position of ELEMENT_FACE or ELEMENT_FACE_INTEGRATION_POINT. Possible values are:

- FACE_UNKOWN, specifying this value indicates that no value has been specified.
- FACE1, specifying this value indicates that element face 1 has been specified.
- FACE2, specifying this value indicates that element face 2 has been specified.
- FACE3, specifying this value indicates that element face 3 has been specified.
- FACE4, specifying this value indicates that element face 4 has been specified.
- FACE5, specifying this value indicates that element face 5 has been specified.
- FACE6, specifying this value indicates that element face 6 has been specified.
- SIDE1, specifying this value indicates that element side 1 has been specified.
- SIDE2, specifying this value indicates element side 2 has been specified.
- END1, specifying this value indicates that element end 1 has been specified.
- END2, specifying this value indicates that element end 2 has been specified.
- END3, specifying this value indicates that element end 3 has been specified.

The default value is FACE_UNKNOWN.

node

An *OdbMeshNode* object specifying the node for which the data are to be collected.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for a region.

Path

```
odbAccess.HistoryPoint
```

Required arguments

region

An *OdbSet* object specifying the region for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for the *OdbAssembly* object.

Path

```
odbAccess.HistoryPoint
```

Required arguments

assembly An *OdbAssembly* object specifying the assembly for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for the *OdbInstance* object.

Path

```
odbAccess.HistoryPoint
```

Required arguments

instance An *OdbInstance* object specifying the instance for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

Members

The HistoryPoint object has members with the same names and descriptions as the arguments to the *HistoryPoint* method.

In addition, the HistoryPoint object can have the following members:

ipNumber

An Int specifying the integration point. This argument is used to define a history output position of INTEGRATION_POINT or ELEMENT_FACE_INTEGRATION_POINT. The default value is 0.

face

A SymbolicConstant specifying the element face. This argument is used to define a history output position of ELEMENT_FACE or ELEMENT_FACE_INTEGRATION_POINT. Possible values are:

- FACE_UNKOWN, specifying this value indicates that no value has been specified.
- FACE1, specifying this value indicates that element face 1 has been specified.
- FACE2, specifying this value indicates that element face 2 has been specified.
- FACE3, specifying this value indicates that element face 3 has been specified.
- FACE4, specifying this value indicates that element face 4 has been specified.
- FACE5, specifying this value indicates that element face 5 has been specified.
- FACE6, specifying this value indicates that element face 6 has been specified.
- SIDE1, specifying this value indicates that element side 1 has been specified.
- SIDE2, specifying this value indicates element side 2 has been specified.
- END1, specifying this value indicates that element end 1 has been specified.
- END2, specifying this value indicates that element end 2 has been specified.
- END3, specifying this value indicates that element end 3 has been specified.

The default value is FACE_UNKNOWN.

position

A SymbolicConstant specifying the result position of the history point. Possible values are:

- NODAL, specifying the values calculated at the nodes.
- ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- INTEGRATION_POINT, specifying the values calculated at the integration points.
- ELEMENT_FACE, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element.
- ELEMENT_FACE_INTEGRATION_POINT, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element when the surface facets have integration points.
- WHOLE_ELEMENT, specifying the results obtained for whole element variables.
- WHOLE_REGION, specifying the results for an entire region of the model.
- WHOLE_PART_INSTANCE, specifying the results for an entire part instance of the model.
- WHOLE_MODEL, specifying the results for the entire model.

element

An *OdbMeshElement* object specifying the element for which the data are to be collected.

sectionPoint A *SectionPoint* object.

region An *OdbSet* object specifying the region for which the data are to be collected.

assembly An *OdbAssembly* object specifying the assembly for which the data are to be collected.

instance An *OdbInstance* object specifying the instance for which the data are to be collected.

HistoryRegion object

The HistoryRegion object contains history data for a single location in the model.

Access

```
import odbAccess
session.odbs[name].steps[name].historyRegions[name]
```

HistoryRegion(...)

This method creates a HistoryRegion object.

Path

```
session.odbs[name].steps[name].HistoryRegion
```

Required arguments

name A String specifying the name of the HistoryRegion object.

description A String specifying the description of the HistoryRegion object.

point A *HistoryPoint* object specifying the point to which the history data refer.

Optional arguments

loadCase `None` or an *OdbLoadCase* object specifying the load case associated with the HistoryRegion object. The default value is `None`.

Return value

A HistoryRegion object.

Exceptions

None.

getSubset(...)

This method returns a subset of the data in the HistoryRegion object.

Required arguments

variableName A String specifying the name of the output variable to return.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

`getSubset(...)`

This method returns a subset of the data in the HistoryRegion object.

Required arguments

start A Float specifying the start of the subset. This is the same as the first item in the data array member of the *HistoryOutput* object.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

`getSubset(...)`

This method returns a subset of the data in the HistoryRegion object.

Required arguments

start A Float specifying the start of the subset. This is the same as the first item in the data array member of the *HistoryOutput* object.

end A Float specifying the end of the subset.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

Members

The HistoryRegion object has members with the same names and descriptions as the arguments to the *HistoryRegion* method.

In addition, the HistoryRegion object can have the following members:

position

A SymbolicConstant specifying the position of the history output. Possible values are NODAL, INTEGRATION_POINT, WHOLE_ELEMENT, WHOLE_REGION, and WHOLE_MODEL.

historyOutputs

A repository of *HistoryOutput* objects.

JobData object

The JobData object describes the context in which the analysis was run.

Access

```
import odbAccess
session.odbs[name].jobData
```

Members

The JobData object has the following members:

name
A String specifying the name of the job.

analysisCode
A SymbolicConstant specifying the analysis code. Possible values are ABAQUS_STANDARD, ABAQUS_EXPLICIT, and UNKNOWN_ANALYSIS_CODE.

precision
A SymbolicConstant specifying the precision. Only SINGLE_PRECISION is currently supported. Possible values are DOUBLE_PRECISION and SINGLE_PRECISION.

version
A String specifying the release of the analysis code.

creationTime
A String specifying the date and time at which the analysis was run.

modificationTime
A String specifying the date and time at which the database was last modified.

machineName
A String specifying the name of the machine on which the analysis was run.

productAddOns
A String specifying an odb_Sequence of productAddOns. Possible values are: Possible values are AQUA, DESIGN, BIORID, CEL, SOLITER, and CAVPARALLEL.

OdbAssembly object

The OdbAssembly object has no constructor; it is created automatically when an *Odb* object is created. Abaqus creates the *rootAssembly* member when an *Odb* object is created.

Access

```
import odbAccess
session.odbs[name].rootAssembly
```

`ConnectorOrientation(...)`

This method assigns a connector orientation to a connector region.

Required arguments

region An *OdbSet* specifying a region.

Optional arguments

localCsys1 An *OdbDatumCsys* object specifying the first connector node local coordinate system or `None`, indicating the global coordinate system.

axis1 A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation of the first connector node is applied. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle1 A Float specifying the angle of the additional rotation about the first connector node axis. The default value is 0.0.

orient2sameAs1 A Boolean specifying whether the same orientation settings should be used for the second node of the connector. The default value is OFF.

localCsys2 An *OdbDatumCsys* object specifying the second connector node local coordinate system or `None`, indicating the global coordinate system.

axis2 A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation of the second connector node is applied. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle2 A Float specifying the angle of the additional rotation about the second connector node axis. The default value is 0.0.

Return value

`None`.

Exceptions

If *region* is not an element set:	OdbError: Connector orientation assignment requires element set.

`SectionAssignment(...)`

This method is used to assign a section on an assembly or part. Section assignment on the assembly is limited to the connector elements only.

Required arguments

region	An *OdbSet* specifying a region.
section	A *Section* object.

Optional arguments

None.

Return value

`None.`

Exceptions

If *region* is not an element set:	OdbError: Section assignment requires element set.

`addElements(...)`

This method is used to define elements using nodes defined at the OdbAssembly and/or *OdbInstance* level. For connector elements connected to ground, specify the lone node in the connectivity. The position of the ground node cannot be specified. This is a limitation.



Warning:

Adding elements not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

labels	A sequence of Ints specifying the element labels.
connectivity	A sequence of sequences of Ints specifying the nodal connectivity.
instanceNames	A sequence of Strings specifying the instanceNames of each node in the nodal connectivity array. If the node is defined at the assembly level, the instance name should be an empty string
type	A String specifying the element type.

Optional arguments

elementSetName	A String specifying a name for this element set. The default value is the empty string.
sectionCategory	A *SectionCategory* object for this element set.

Return value

`None.`

Exceptions

Only certain element types are permitted at the assembly level. e.g., connector elements.	OdbError: Addition of this element type is not permitted at the assembly level
If length of label array does not match connectivity data length:	OdbError: Connectivity array must be provided for all elements

`addNodes(...)`

This method adds nodes to the OdbAssembly object using node labels and coordinates.



Warning:

Adding nodes not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

labels	A sequence of Ints specifying the node labels.
coordinates	A sequence of sequences of Floats specifying the nodal coordinates.

Optional arguments

nodeSetName	A String specifying a name for this node set. The default value is None.

Return value

`None.`

Exceptions

If length of labels does not match length of coordinates:	OdbError: Number of node labels and coordinates does not match
If width of coordinate array does not match assembly dimension:	OdbError: Node location specification does not correspond to part dimensions

RigidBody(...)

This method defines an *OdbRigidBody* on the assembly.

Required arguments

referenceNode	An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position	A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are INPUT and CENTER_OF_MASS. The default value is INPUT.
isothermal	A Boolean specifying an isothermal rigid body. The default value is OFF. This parameter is used only for a fully coupled thermal stress analysis.
elset	An *OdbSet* specifying an element set assigned to the rigid body.
pinNodes	An *OdbSet* specifying pin-type nodes assigned to the rigid body.
tieNodes	An *OdbSet* specifying tie-type nodes assigned to the rigid body.
analyticSurface	An *AnalyticSurface* specifying the Analytic Rigid Surface assigned to the rigid body.

Return value

`None.`

Exceptions

If *referenceNode* is not a node set:	OdbError: Rigid body definition requires a node set.

getMassProperties(...)

This method returns the mass properties of the element set. Only beams, trusses, shells, solids, point, nonstructural mass, and rotary inertia elements are supported.

Required arguments

None.

Optional arguments

elementSet	An *OdbSet* object specifying element sets or a string containing a name of the element set. The default element set present in odb named "ALL ELEMENTS" is queried by default.

specifyDensity A Boolean specifying whether a user-specified density should be used in elements with density errors such as undefined material density. The default value is False.

density A double value specifying the user-specified density value to be used in elements with density errors. The user-specified density should be greater than 0.

specifyThickness A Boolean specifying whether a user-specified thickness should be used in elements with thickness errors such as undefined thickness. The default value is False.

thickness A double value specifying the user-specified thickness value to be used in elements with thickness errors. The user-specified thickness should be greater than 0.

miAboutCenterOfMass A Boolean specifying if the moments of inertia should be evaluated about the center of mass. The default value is True.

miAboutPoint A tuple of three floats specifying the coordinates of the point about which to evaluate the moment of inertia. By default, if the moments of inertia are not evaluated about the center of mass, they are evaluated about the origin.

Return value

A Dictionary object with the following items:

area: None or a Float specifying the sum of the area of the specified faces. The area is computed only for one side for shells.

areaCentroid: None or a tuple of three Floats representing the coordinates of the area centroid.

volume: None or a Float specifying the volume of the specified elements.

volumeCentroid: None or a tuple of three Floats representing the coordinates of the volume centroid.

massFromMassPerUnitSurfaceArea: None or a Float specifying the mass due to mass per unit surface area.

mass: None or a Float specifying the mass of the specified elements. It is the total mass and includes mass from quantities such as mass per unit surface area.

centerOfMass: None or a tuple of three Floats representing the coordinates of the center of mass.

momentOfInertia: None or a tuple of six Floats representing the moments of inertia about the center of mass or about the point specified.

warnings: A tuple of SymbolicConstants representing the problems encountered while computing the mass properties. Possible SymbolicConstants are:

UNSUPPORTED_ENTITIES: Some unsupported entities exist in the specified elements. The mass properties are computed only for beams, trusses, shells, solids, point and nonstructural mass elements, and rotary inertia elements. The mass properties are not computed for axisymmetric elements, springs, connectors, gaskets, or any other elements.

MISSING_THICKNESS: For some elements, the section definitions are missing thickness values.

ZERO_THICKNESS: For some elements, the section definitions have a zero thickness value.

VARIABLE_THICKNESS: The nodal thickness or field thickness specified for some elements has been ignored.

NON_APPLICABLE_THICKNESS: For some elements, the thickness value is not applicable to the corresponding sections specified on the element.

MISSING_DENSITY: For some elements, the section definitions are missing material density values.

MISSING_MATERIAL_DEFINITION: For some elements, the material definition is missing.

ZERO_DENSITY: For some elements, the section definitions have a zero material density value.

UNSUPPORTED_DENSITY: For some elements, either a negative material density or a temperature-dependent density has been specified, or the material value is missing for one or more plies in the composite section.

SHELL_OFFSETS: For shells, this method does not account for any offsets specified.

MISSING_SECTION_DEFINITION: For some elements, the section definition is missing.

UNSUPPORTED_SECTION_DEFINITION: The section definition provided for some elements is not supported.

REINFORCEMENTS: This method does not account for any reinforcements specified on the model.

SMEARED_PROPERTIES: For elements with composite section assignments, the density is smeared across the thickness. The volume centroid and center of mass computations for a composite shell use a lumped mass approach where the volume and mass is assumed to be lumped in the plane of the shell. As a result of these approximations the volume centroid, center of mass and moments of inertia might be slightly inaccurate for elements with composite section assignments.

UNSUPPORTED_NON_STRUCTURAL_MASS_ENTITIES: This method does not account for any nonstructural mass on wires.

INCORRECT_MOMENT_OF_INERTIA: For elements with nonstructural mass per volume, the nonstructural mass is assumed to be a point mass at the centroid of the regions. Thus, the moments of inertia might be inaccurate as the distribution of the nonstructural mass is not accounted for. Use the mesh for accurately computing the moments of inertia.

MISSING_BEAM_ORIENTATIONS: For some elements with beam section assignments, the beam section orientations are missing.

UNSUPPORTED_BEAM_PROFILES: This method supports the Box, Pipe, Circular, Rectangular, Hexagonal, Trapezoidal, I, L, T, Arbitrary, and Tapered beam profiles. Any other beam profile is not supported.

TAPERED_BEAM_MI: Moment of inertia calculations for tapered beams are not accurate.

SUBSTRUCTURE_INCORRECT_PROPERTIES: The user-assigned density and thickness is not considered for substructures.

Exceptions

None.

Members

The OdbAssembly object can have the following members:

instances A repository of *OdbInstance* objects.

nodeSets A repository of *OdbSet* objects specifying node sets.

elementSets A repository of *OdbSet* objects specifying element sets.

surfaces A repository of *OdbSet* objects specifying surfaces.

nodes An *OdbMeshNodeArray* object.

elements

An *OdbMeshElementArray* object.

datumCsyses

A repository of *OdbDatumCsys* objects.

sectionAssignments

A *SectionAssignmentArray* object.

rigidBodies

An *OdbRigidBodyArray* object.

pretensionSections

An *OdbPretensionSectionArray* object.

connectorOrientations

A *ConnectorOrientationArray* object.

OdbDatumCsys object

The OdbDatumCsys object contains a coordinate system that can be stored in an output database. You can create the datum coordinate system in the Visualization module during an Abaqus/CAE session and save the datum coordinate system to the output database before you exit Abaqus/CAE. Alternatively, the analysis code can write the datum coordinate system to the output database.

Access

```
import odbAccess
session.odbs[name].rootAssembly.datumCsyses[name]
```

DatumCsysByThreePoints(...)

This method creates an OdbDatumCsys object using three points. A datum coordinate system created with this method results in a fixed system.

Path

```
session.odbs[name].rootAssembly.DatumCsysByThreePoints
```

Required arguments

name	A String specifying the repository key.
coordSysType	A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.
origin	A sequence of Floats specifying the coordinates of the origin of the datum coordinate system.
point1	A sequence of Floats specifying the coordinates of a point on the local 1- or r-axis.
point2	A sequence of Floats specifying the coordinates of a point in the 1–2 or r–θ plane.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsysByThreeNodes(...)

This method creates an OdbDatumCsys object using the coordinates of three *OdbMeshNode* objects. A datum coordinate system created with this method results in a system that follows the position of the three nodes. Results, such as those

for displacement, are resolved into the orientation of the datum coordinate system without regard to the position of its origin. The last three arguments are given in the form of an *OdbMeshNode* object.

Path

```
session.odbs[name].rootAssembly.DatumCsysByThreeNodes
```

Required arguments

name A String specifying the repository key.

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

origin An *OdbMeshNode* object specifying a node at the origin of the datum coordinate system.

point1 An *OdbMeshNode* object specifying a node on the local 1- or r-axis.

point2 An *OdbMeshNode* object specifying a node in the 1–2 or r–θ plane.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsysByThreeCircNodes(...)

This method is convenient to use where there are no nodes along the axis of a hollow cylinder or at the center of a hollow sphere. The three nodes that you provide as arguments determine a circle in space. The center of the circle is the origin of the datum coordinate system. The normal to the circle is parallel to the z-axis of a cylindrical coordinate system or to the ϕ-axis of a spherical coordinate system. The line from the origin to the first node defines the r-axis.

Path

```
session.odbs[name].rootAssembly.DatumCsysByThreeCircNodes
```

Required arguments

name A String specifying the repository key.

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

node1Arc An *OdbMeshNode* object that lies on the circular arc.

node2Arc An *OdbMeshNode* object that lies on the circular arc.

node3Arc An *OdbMeshNode* object that lies on the circular arc.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsysBy6dofNode(...)

A datum coordinate system created with this method results in a system that follows the position of a node. The node location defines the origin of the datum coordinate system. The rotational displacement (UR1, UR2, UR3) of the node defines the orientation of the coordinate system axes. Results, such as those for displacement, are resolved into the orientation of the datum coordinate system without regard to the position of its origin. The last argument is given in the form of an *OdbMeshNode* object.

Path

```
session.odbs[name].rootAssembly.DatumCsysBy6dofNode
```

Required arguments

name A String specifying the repository key.

coordSysType A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

origin An *OdbMeshNode* object specifying the origin of the datum coordinate system.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsys(...)

This method copies oneOdbDatumCsys object to a new OdbDatumCsys object.

Path

```
session.odbs[name].rootAssembly.DatumCsys
```

Required arguments

name A String specifying the repository key.

datumCsys An OdbDatumCsys object specifying the object to be copied.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

globalToLocal(...)

This method transforms specified coordinates in the global coordinate system into this local coordinate system.

Required arguments

coordinates A tuple of three Floats representing the coordinates in the global coordinate system.

Optional arguments

None.

Return value

A tuple of three Floats representing the coordinates in this local coordinate system.

Exceptions

None.

localToGlobal(...)

This method transforms specified coordinates in this local coordinate system into the global coordinate system.

Required arguments

coordinates A tuple of three Floats representing the coordinates in this local coordinate system.

Optional arguments

None.

Return value

A tuple of three Floats representing the coordinates in the global coordinate system.

Exceptions

None.

Members

The OdbDatumCsys object has the following members:

name
A String specifying the repository key.

coordSysType
A SymbolicConstant specifying the type of coordinate system. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

origin
A tuple of Floats specifying the coordinates of the origin of the datum coordinate system.

xAxis
A tuple of Floats specifying a point on the *X*-axis.

yAxis
A tuple of Floats specifying a point on the *Y*-axis.

zAxis
A tuple of Floats specifying a point on the *Z*-axis.

OdbFrame object

The domain of the OdbFrame object is taken from the parent step.

Access

```
import odbAccess
session.odbs[name].steps[name].frames[i]
```

Frame(...)

This method creates an OdbFrame object and appends it to the frame sequence.

Path

```
session.odbs[name].steps[name].Frame
```

Required arguments

incrementNumber An Int specifying the frame increment number within the step. The base frame has normally increment number 0, and the results run from 1. In case of multiple load cases, the same increment number is duplicated for each load case.

frameValue A Float specifying the value in units determined by the *domain* member of the *Step* object. The equivalent in the time domain is *stepTime*; in the frequency domain the equivalent is *frequency*; and in the modal domain the equivalent is *mode*.

Optional arguments

description A String specifying the contents of the frame. The default value is an empty string.

Return value

An OdbFrame object.

Exceptions

None.

Frame(...)

This constructor creates an OdbFrame object in the frequency domain and appends it to the frame sequence. The arguments to the constructor are valid only when *domain*=FREQUENCY or *domain*=MODAL.

Path

```
session.odbs[name].steps[name].Frame
```

Required arguments

mode An Int specifying the eigenmode. This member is valid only if *domain*=MODAL.

frequency A Float specifying the frequency. This member is valid only if *domain*=FREQUENCY or if the *procedure* member of the *OdbStep* object="*FREQUENCY". The default value is 0.0.

Optional arguments

description A String specifying the contents of the frame. The default value is an empty string.

Return value

An OdbFrame object.

Exceptions

None.

Frame(...)

This constructor creates an OdbFrame object for a specific load case and appends it to the frame sequence.

Path

```
session.odbs[name].steps[name].Frame
```

Required arguments

loadCase An *OdbLoadCase* object specifying the load case for the frame.

Optional arguments

description A String specifying the contents of the frame. The default value is an empty string.

frequency A Float specifying the frequency. This member is valid only if *domain*=FREQUENCY or if the *procedure* member of the *OdbStep* object="*FREQUENCY". The default value is 0.0.

Return value

An OdbFrame object.

Exceptions

None.

Members

The OdbFrame object has members with the same names and descriptions as the arguments to the *Frame* method.

In addition, the OdbFrame object can have the following members:

cyclicModeNumber

An Int specifying the cyclic mode number associated with the data stored on this frame. Only frequency analyses of cyclic symmetry models possess cyclic mode numbers.

domain

A SymbolicConstant specifying the domain of the step of which the frame is a member. Possible values are TIME, FREQUENCY, and MODAL.

frequency

A Float specifying the frequency. This member is valid only if *domain*=FREQUENCY or if the *procedure* member of the *OdbStep* object="*FREQUENCY". The default value is 0.0.

mode

An Int specifying the eigenmode. This member is valid only if *domain*=MODAL.

associatedFrame

An OdbFrame object specifying the real or imaginary portion of the data corresponding to this cyclic symmetry mode.

fieldOutputs

A repository of *FieldOutput* objects specifying the key to the *fieldOutputs*repository is a String representing an output variable.

loadCase

An *OdbLoadCase* object specifying the load case for the frame.

OdbInstance object

A part instance is the usage of a part within an assembly.

Access

```
import odbAccess
session.odbs[name].rootAssembly.instances[name]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance
```

Instance(...)

This method creates an OdbInstance object from an *OdbPart* object.

Path

session.odbs[*name*].rootAssembly.Instance

Required arguments

name A String specifying the instance name.

object An *OdbPart* object.

Optional arguments

localCoordSystem A sequence of sequences of three Floats specifying the rotation and translation of the part instance in the global Cartesian coordinate system. The first three sequences specify the new local coordinate system with its center at the origin.

- The first sequence specifies a point on the 1-axis.
- The second sequence specifies a point on the 2-axis.
- The third sequence specifies a point on the 3-axis.

The fourth sequence specifies the translation of the local coordinate system from the origin to its intended location.

For example, the following sequence moves a part 10 units in the *X*-direction with no rotation:

```
localCoordSystem = ((1, 0, 0), (0, 1, 0),
                                  (0, 0, 1), (10, 0, 0))
```

The following sequence moves a part 5 units in the *X*-direction with rotation:

```
localCoordSystem = ((0, 1, 0), (1, 0, 0),
                                  (0, 0, 1), (5, 0, 0))
```

transforms a part containing the two points

```
                                  Pt1= (1,0,0)
                                  Pt2= (2,0,0)
```

to

```
Pt1 = (0, 6, 0)
Pt2 = (0, 7, 0)
```

Return value

An OdbInstance object.

Exceptions

InvalidNameError.

assignBeamOrientation(...)

This method assigns a beam section orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

method A SymbolicConstant specifying the assignment method. Only a value of N1_COSINES is currently supported.

vector A sequence of three Floats specifying the approximate local $\mathbf{n_1}$-direction of the beam cross-section.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignMaterialOrientation(...)

This method assigns a material orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

localCsys An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

stackDirection A SymbolicConstant specifying the stack or thickness direction of the material. Possible values are STACK_1, STACK_2, STACK_3, and STACK_ORIENTATION. The default value is STACK_3.

Return value

`None.`

Exceptions

None.

`assignRebarOrientation(...)`

This method assigns a rebar reference orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

localCsys An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

`None.`

Exceptions

None.

`getElementFromLabel(...)`

This method is used to retrieved an element with a specific label from an instance object.

Required arguments

label An Int specifying the element label.

Optional arguments

None.

Return value

An *OdbMeshElement* object.

Exceptions

If no element with the specified label exists: OdbError: Invalid element label

`getNodeFromLabel(...)`

This method is used to retrieved a node with a specific label from an instance object.

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An*OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

`assignSection(...)`

This method is used to assign a section to a region on an instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

section A *Section* object.

Optional arguments

None.

Return value
None.

Exceptions

If *region* is not an element set:	OdbError: Section assignment requires element set.
If the element set is not from the current instance:	OdbError: Section assignment requires element set from this part instance.

AnalyticRigidSurf2DPlanar(...)

This method is used to define a two-dimensional *AnalyticSurface* object on the instance.

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius	A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value
None.

Exceptions

If *OdbPart* associated with the part instance is of type THREE_D:	OdbError: 2D-Planar Analytic Rigid Surface can be defined only if the instance is of type TWO_D_PLANAR or AXISYMMETRIC.

AnalyticRigidSurfExtrude(...)

This method is used to define a three-dimensional cylindrical *AnalyticSurface* on the instance.

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius
A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

localCoordData
A sequence of sequences of Floats specifying the global coordinates of points used to define the local coordinate system.

Return value

`None.`

Exceptions

If *OdbPart* associated with the part instance is not of type THREE_D:
OdbError: Analytic Rigid Surface of type CYLINDER can be defined only if the instance is of type THREE_D.

`AnalyticRigidSurfRevolve(...)`

This method is used to define a three-dimensional *AnalyticSurface* of revolution on the instance.

Required arguments

name
The name of the analytic surface.

profile
A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius
A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

localCoordData
A sequence of sequences of Floats specifying the global coordinates of points used to define the local coordinate system.

Return value

`None.`

Exceptions

If *OdbPart* associated with the part instance is not of type THREE_D:
OdbError: Analytic Rigid Surface of type REVOLUTION can be defined only if the instance is of type THREE_D.

`RigidBody(...)`

This method defines an *OdbRigidBody* on the instance.

Required arguments

referenceNode An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are INPUT, and CENTER_OF_MASS. The default value is INPUT.

isothermal A Boolean specifying an isothermal rigid body. The default value is OFF. This parameter is used only for a fully-coupled thermal stress analysis.

elset An *OdbSet* specifying an element set assigned to the rigid body.

pinNodes An *OdbSet* specifying pin-type nodes assigned to the rigid body.

tieNodes An *OdbSet* specifying tie-type nodes assigned to the rigid body.

analyticSurface An *AnalyticSurface* specifying the Analytic Rigid Surface assigned to the rigid body.

Return value

`None.`

Exceptions

If *referenceNode* is not a node set: OdbError: Rigid body definition requires a node set.

`getNodeFromLabel(...)`

This method is used to retrieved a node with a specific label from an instance object.

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

`getNodeFromLabel(...)`

This method is used to retrieved a node with a specific label from an instance object.

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

Members

The OdbInstance object can have the following members:

name A String specifying the instance name.

type A SymbolicConstant specifying the type of the *Part* object. Only a value of DEFORMABLE_BODY is currently supported.

embeddedSpace A SymbolicConstant specifying the dimensionality of the *Part* object. Possible values are THREE_D, TWO_D_PLANAR, AXISYMMETRIC, and UNKNOWN_DIMENSION.

resultState A SymbolicConstant specifying the state of the Instance as modified by the analysis. This member is only present if the Instance is part of the RootAssemblyState tree. Possible values are:

- PROPAGATED, specifying that the value is the same as the previous frame or the original rootAssembly.
- MODIFIED, specifying that the geometry of the instance has been changed at this frame.

The default value is PROPAGATED.

nodes An *OdbMeshNodeArray* object.

elements An *OdbMeshElementArray* object.

nodeSets A repository of *OdbSet* objects specifying node sets.

elementSets A repository of *OdbSet* objects specifying element sets.

surfaces A repository of *OdbSet* objects specifying surfaces.

sectionAssignments A *SectionAssignmentArray* object.

rigidBodies An *OdbRigidBodyArray* object.

beamOrientations A *BeamOrientationArray* object.

materialOrientations A *MaterialOrientationArray* object.

rebarOrientations A *RebarOrientationArray* object.

analyticSurface An *AnalyticSurface* object specifying analytic Surface defined on the instance.

OdbLoadCase object

The OdbLoadCase object describes a load case.

Access

```
import odbAccess
session.odbs[name].steps[name].frames[i].loadCase
session.odbs[name].steps[name].historyRegions[name].loadCase
session.odbs[name].steps[name].loadCases[name]
```

LoadCase(...)

This method creates an OdbLoadCase object.

Path

session.odbs[*name*].steps[*name*].LoadCase

Required arguments

name A String specifying the name of the OdbLoadCase object.

Optional arguments

None.

Return value

An OdbLoadCase object.

Exceptions

None.

Members

The OdbLoadCase object has members with the same names and descriptions as the arguments to the *LoadCase* method.

OdbMeshElementArray object

A sequence of *OdbMeshElement object*.

Access

```
import odbAccess
session.odbs[name].parts[name].elements
session.odbs[name].parts[name].elementSets[name].elements
session.odbs[name].parts[name].nodeSets[name].elements
session.odbs[name].parts[name].surfaces[name].elements
session.odbs[name].rootAssembly.elements
session.odbs[name].rootAssembly.elementSets[name].elements
session.odbs[name].rootAssembly.instances[name].elements
session.odbs[name].rootAssembly.instances[name].elementSets[name]\
.elements
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.elements
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.elements
session.odbs[name].rootAssembly.nodeSets[name].elements
session.odbs[name].rootAssembly.surfaces[name].elements
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elements
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elementSets[name].elements
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].elements
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].elements
```

`getMemberFromAll(...)`

This method returns a tuple containing a member or members for each OdbMeshElement in the OdbMeshElementArray.

Required arguments

member	A String specifying a OdbMeshElement member name or a tuple of member names.

Optional arguments

None.

Return value

A tuple of member values (one for each OdbMeshElement) or a tuple of tuples of member values (if you specified a tuple of member names in the *member* argument).

Exceptions

An exception occurs if no such member exists.	AttributeError: 'OdbMeshElement' object has no attribute ''

Members

The OdbMeshElementArray object has no members.

OdbMeshElement object

OdbMeshElement objects are created with the `part.addElements` or `rootAssembly.addElements` methods.

Access

```
import odbAccess
session.odbs[name].parts[name].elements[i]
session.odbs[name].parts[name].elementSets[name].elements[i]
session.odbs[name].parts[name].nodeSets[name].elements[i]
session.odbs[name].parts[name].surfaces[name].elements[i]
session.odbs[name].rootAssembly.elements[i]
session.odbs[name].rootAssembly.elementSets[name].elements[i]
session.odbs[name].rootAssembly.instances[name].elements[i]
session.odbs[name].rootAssembly.instances[name].elementSets[name]\
.elements[i]
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.elements[i]
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.elements[i]
session.odbs[name].rootAssembly.nodeSets[name].elements[i]
session.odbs[name].rootAssembly.surfaces[name].elements[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elements[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elementSets[name].elements[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].elements[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].elements[i]
```

`getNormal(...)`

This method returns the normal direction for the element face.

Required arguments

faceIndex The value of *faceIndex* is 0 for a shell element and can range from 0 to 5 for a solid element.

Optional arguments

stepName Name of the step.

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the total fime or frequency.

match A SymbolicConstant specifying which frame to return if there is no frame at the exact frame value. Possible values are CLOSEST, BEFORE, AFTER, and EXACT. The default value is CLOSEST.

When *match*=CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

A tuple of 3 floats representing the unit normal vector. If the element face is collapsed such that a normal cannot be computed, a zero-length vector is returned.

Exceptions

If the exact frame is not found:	OdbError: Frame not found.
If the step name is not found:	OdbError: Step is not present in the ODB.
If *frameValue* is not provided and *stepName* is empty:	*frameValue*OdbError: *stepName* should be specified with *frameValue*.

Members

The OdbMeshElement object can have the following members:

label
An Int specifying the element label.

type
A String specifying the element type.

sectionCategory
A *SectionCategory* object specifying the element section properties.

connectivity
A tuple of Ints specifying the element connectivity. For connector elements connected to ground, the other node is repeated in the connectivity data. The position of the ground node cannot be ascertained. This is a limitation. It is important to note the difference with MeshElement object of MDB where the connectivity is node indices instead of node labels.

instanceNames
A tuple of Strings specifying the instance names for nodes in the element connectivity.

instanceName
A String specifying the instance name.

OdbMeshNodeArray object

A sequence of *OdbMeshNode object*

Access

```
import odbAccess
session.odbs[name].parts[name].nodes
session.odbs[name].parts[name].nodeSets[name].nodes
session.odbs[name].parts[name].surfaces[name].nodes
session.odbs[name].rootAssembly.instances[name].nodes
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.nodes
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.nodes
session.odbs[name].rootAssembly.nodes
session.odbs[name].rootAssembly.nodeSets[name].nodes
session.odbs[name].rootAssembly.surfaces[name].nodes
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodes
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].nodes
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].nodes
```

`getMemberFromAll(...)`

This method returns a tuple containing a member or members for each OdbMeshNode in the OdbMeshNodeArray.

Required arguments

member	A String specifying a OdbMeshNode member name or a tuple of member names.

Optional arguments

None.

Return value

A tuple of member values (one for each OdbMeshNode) or a tuple of tuples of member values (if you specified a tuple of member names in the *member* argument).

Exceptions

An exception occurs if no such member exists.	AttributeError: 'OdbMeshNode' object has no attribute ''

Members

The OdbMeshNodeArray object has no members.

OdbMeshNode object

OdbMeshNode objects are created with the `part.addNodes` method.

Access

```
import odbAccess
session.odbs[name].parts[name].nodes[i]
session.odbs[name].parts[name].nodeSets[name].nodes[i]
session.odbs[name].parts[name].surfaces[name].nodes[i]
session.odbs[name].rootAssembly.instances[name].nodes[i]
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.nodes[i]
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.nodes[i]
session.odbs[name].rootAssembly.nodes[i]
session.odbs[name].rootAssembly.nodeSets[name].nodes[i]
session.odbs[name].rootAssembly.surfaces[name].nodes[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodes[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].nodes[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].nodes[i]
```

Members

The OdbMeshNode object has the following members:

label

An Int specifying the node label.

coordinates

A tuple of Floats specifying the nodal coordinates in the global Cartesian coordinate system.

OdbPart object

The OdbPart object is similar to the kernel Part object and contains nodes and elements, but not geometry.

Access

```
import odbAccess
session.odbs[name].parts[name]
```

Part(...)

This method creates an OdbPart object. Nodes and elements are added to this object at a later stage.

Path

```
session.odbs[name].Part
```

Required arguments

name A String specifying the part name.

embeddedSpace A SymbolicConstant specifying the dimensionality of the *Part* object. Possible values are THREE_D, TWO_D_PLANAR, and AXISYMMETRIC.

type A SymbolicConstant specifying the type of the *Part* object. Possible values are DEFORMABLE_BODY and ANALYTIC_RIGID_SURFACE.

Optional arguments

None.

Return value

An OdbPart object.

Exceptions

None.

`addElements(...)`

This method adds elements to an OdbPart object using element labels and nodal connectivity.

Warning:

Adding elements not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

labels A sequence of Ints specifying the element labels.

connectivity A sequence of sequences of Ints specifying the nodal connectivity.

type A String specifying the element type.

Optional arguments

elementSetName A String specifying a name for this element set. The default value is the empty string.

sectionCategory A *SectionCategory* object for this element set.

Return value

`None.`

Exceptions

None.

`addElements(...)`

This method adds elements to an OdbPart object using a sequence of element labels and nodal connectivity.



Warning:

Adding elements not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

elementData A sequence of sequences of Ints specifying the element labels and nodal connectivity, in the form ((*label*, *c1*, *c2*, *c3*, *c4*), (*label*, *c1*, *c2*, *c3*, *c4*), ...).

type A String specifying the element type. The value can be user defined.

Optional arguments

elementSetName A String specifying a name for this element set. The default value is None.

sectionCategory A *SectionCategory* object for this element set.

Return value

`None.`

Exceptions

None.

addNodes(...)

This method adds nodes to an OdbPart object using node labels and coordinates.



Warning:

Adding nodes not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

labels A sequence of Ints specifying the node labels.

coordinates A sequence of sequences of Floats specifying the nodal coordinates.

Optional arguments

nodeSetName A String specifying a name for this node set. The default value is None.

Return value

`None`.

Exceptions

None.

addNodes(...)

This method adds nodes to an OdbPart object using a sequence of node labels and coordinates.



Warning:

Adding nodes not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Required arguments

nodeData A sequence of tuples specifying the node labels and coordinates, in the form ((*label*, *x*, *y*, *z*), (*label*, *x*, *y*, *z*), ...).

Optional arguments

nodeSetName A String specifying a name for this node set. The default value is None.

Return value

`None`.

Exceptions

None.

assignBeamOrientation(...)

This method assigns a beam section orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

method A SymbolicConstant specifying the assignment method. Only a value of N1_COSINES is currently supported.

vector A sequence of three Floats specifying the approximate local $\mathbf{n_1}$ -direction of the beam cross-section.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignMaterialOrientation(...)

This method assigns a material orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

localCSys An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

stackDirection A SymbolicConstant specifying the stack or thickness direction of the material. Possible values are STACK_1, STACK_2, STACK_3, and STACK_ORIENTATION. The default value is STACK_3.

Return value
`None.`

Exceptions
None.

`assignRebarOrientation(...)`

This method assigns a rebar reference orientation to a region of a part instance.

Required arguments

region An *OdbSet* specifying a region on an instance.

localCsys An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value
`None.`

Exceptions
None.

`getElementFromLabel(...)`

This method is used to retrieved an element with a specific label from a part object.

Required arguments

label An Int specifying the element label.

Optional arguments
None.

Return value
An *OdbMeshElement* object.

Exceptions

If no element with the specified label exists:	OdbError: Invalid element label

getNodeFromLabel(...)

This method is used to retrieved a node with a specific label from a part object.

Required arguments

label	An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists:	OdbError: Invalid node label

AnalyticRigidSurf2DPlanar(...)

This method is used to define a two-dimensional *AnalyticSurface* object on the part object.

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius	A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If OdbPart is of type THREE_D:	OdbError: 2D-Planar Analytic Rigid Surface can be defined only if the part is of type TWO_D_PLANAR or AXISYMMETRIC.

AnalyticRigidSurfExtrude(...)

This method is used to define a three-dimensional cylindrical *AnalyticSurface* on the part object.

Required arguments

name The name of the analytic surface.

profile A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If OdbPart is not of type THREE_D: OdbError: Analytic Rigid Surface of type CYLINDER can be defined only if the part is of type THREE_D.

AnalyticRigidSurfRevolve(...)

This method is used to define a three-dimensional *AnalyticSurface* of revolution on the part object.

Required arguments

name The name of the analytic surface.

profile A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If OdbPart is not of type THREE_D:	OdbError: Analytic Rigid Surface of type REVOLUTION can be defined only if the part is of type THREE_D.

`RigidBody(...)`

This method defines an *OdbRigidBody* on the part object.

Required arguments

referenceNode	An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position	A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are INPUT and CENTER_OF_MASS. The default value is INPUT.
isothermal	A Boolean specifying an isothermal rigid body. The default value is OFF. This parameter is used only for a fully-coupled thermal stress analysis.
elset	An *OdbSet* specifying an element set assigned to the rigid body.
pinNodes	An *OdbSet* specifying pin-type nodes assigned to the rigid body.
tieNodes	An *OdbSet* specifying tie-type nodes assigned to the rigid body.
analyticSurface	An *AnalyticSurface* specifying the Analytic Rigid Surface assigned to the rigid body.

Return value

`None.`

Exceptions

If *referenceNode* is not a node set:	OdbError: Rigid body definition requires a node set.

Members

The OdbPart object has members with the same names and descriptions as the arguments to the *Part* method.

In addition, the OdbPart object can have the following members:

nodes	An *OdbMeshNodeArray* object.
elements	An *OdbMeshElementArray* object.
nodeSets	A repository of *OdbSet* objects specifying node sets.

elementSets

A repository of *OdbSet* objects specifying element sets.

surfaces

A repository of *OdbSet* objects specifying surfaces.

sectionAssignments

A *SectionAssignmentArray* object.

beamOrientations

A *BeamOrientationArray* object.

materialOrientations

A *MaterialOrientationArray* object.

rebarOrientations

A *RebarOrientationArray* object.

rigidBodies

An *OdbRigidBodyArray* object.

analyticSurface

An *AnalyticSurface* object specifying analytic Surface defined on the instance.

OdbPretensionSection object

The pretension section object is used to define an assembly load. It associates a pretension node with a pretension section.

Access

```
import odbAccess
session.odbs[name].rootAssembly.pretensionSections[i]
```

Members

The OdbPretensionSection object can have the following members:

node An *OdbSet* object specifying the node set containing the pretension node.

element An *OdbSet* object specifying the element set that defines the pretension section.

surface An *OdbSet* object specifying the surface set that defines the pretension section.

normal A tuple of Floats specifying the components of the normal to the pretension section.

OdbRigidBody object

The Rigid body object is used to bind a set of elements and/or a set of nodes and/or an analytical surface with a reference node.

Access

```
import odbAccess
session.odbs[name].parts[name].rigidBodies[i]
session.odbs[name].rootAssembly.instances[name].rigidBodies[i]
session.odbs[name].rootAssembly.rigidBodies[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.rigidBodies[i]
```

RigidBody(...)

This method creates a OdbRigidBody object.

Path

session.odbs[*name*].rootAssembly.instances[*name*].RigidBody

session.odbs[*name*].rootAssembly.RigidBody

Required arguments

referenceNode An *OdbSet* object specifying the reference node set associated with the rigid body.

Optional arguments

position A SymbolicConstant specifying the specific location of the OdbRigidBody reference node relative to the rest of the rigid body. Possible values are INPUT and CENTER_OF_MASS. The default value is INPUT.

isothermal A Boolean specifying specify whether the OdbRigidBody can have temperature gradients or be isothermal. This is used only for fully coupled thermal-stress analysis The default value is ON.

elements An *OdbSet* object specifying the element set whose motion is governed by the motion of rigid body reference node.

tieNodes An *OdbSet* object specifying the node set which have both translational and rotational degrees of freedom associated with the rigid body.

pinNodes An *OdbSet* object specifying the node set which have only translational degrees of freedom associated with the rigid body.

analyticSurface An *AnalyticSurface* object specifying the analytic surface whose motion is governed by the motion of rigid body reference node.

Return value

An OdbRigidBody object.

Exceptions

None.

Members

The OdbRigidBody object has members with the same names and descriptions as the arguments to the *RigidBody* method.

OdbSequenceAnalyticSurfaceSegment object

A sequence of *AnalyticSurfaceSegment* describing an analytic surface profile.

Access

```
import odbAccess
session.odbs[name].parts[name].analyticSurface.segments
session.odbs[name].rootAssembly.instances[name].analyticSurface\
.segments
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.analyticSurface.segments
```

AnalyticSurfaceProfile()

This method creates a OdbSequenceAnalyticSurfaceSegment object.

Path

odbAccess.AnalyticSurfaceProfile

Arguments

None.

Return value

An OdbSequenceAnalyticSurfaceSegment object.

Exceptions

None.

Start(...)

This method adds a *AnalyticSurfaceSegment* describing the first segment of the surface profile.

Required arguments

origin A sequence of Floats specifying the coordinates of start point.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`Line(...)`

This method adds a *AnalyticSurfaceSegment* describing the line segment of the surface profile.

Required arguments

endPoint A sequence of Floats specifying the coordinates of end point.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`Circle(...)`

This method adds a *AnalyticSurfaceSegment* describing a circular segment of the surface profile.

Required arguments

center A sequence of Floats specifying the coordinates of center of the circular segment.

endPoint A sequence of Floats specifying the coordinates of end point of the circular segment.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`Parabola(...)`

This method adds a *AnalyticSurfaceSegment* describing a parabolic segment of the surface profile.

Required arguments

middlePoint A sequence of Floats specifying the coordinates of middle point of the parabolic segment.

endPoint A sequence of Floats specifying the coordinates of end point of the parabolic segment.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The OdbSequenceAnalyticSurfaceSegment object has no members.

OdbSet object

The set objects are used to identify regions of a model.

Access

```
import odbAccess
session.odbs[name].parts[name].elementSets[name]
session.odbs[name].parts[name].nodeSets[name]
session.odbs[name].parts[name].surfaces[name]
session.odbs[name].rootAssembly.elementSets[name]
session.odbs[name].rootAssembly.instances[name].elementSets[name]
session.odbs[name].rootAssembly.instances[name].nodeSets[name]
session.odbs[name].rootAssembly.instances[name].surfaces[name]
session.odbs[name].rootAssembly.nodeSets[name]
session.odbs[name].rootAssembly.surfaces[name]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elementSets[name]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name]
```

NodeSet(...)

This method creates a node set from an array of *OdbMeshNode* objects (for part instance-level sets) or from a sequence of arrays of *OdbMeshNode* objects (for assembly-level sets).

Path

session.odbs[*name*].parts[*name*].NodeSet

session.odbs[*name*].rootAssembly.instances[*name*].NodeSet

session.odbs[*name*].rootAssembly.NodeSet

Required arguments

name

A String specifying the name of the set and the repository key.

nodes

A sequence of *OdbMeshNode* objects. For example, for a part:

```
nodes=part1.nodes[1:5]
```

For an assembly:

```
nodes=(instance1.nodes[6:7], instance2.nodes[1:5])
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

NodeSetFromNodeLabels(...)

This method creates a node set from a sequence of node labels.

Path

session.odbs[*name*].parts[*name*].NodeSetFromNodeLabels

session.odbs[*name*].rootAssembly.instances[*name*].NodeSetFromNodeLabels

session.odbs[*name*].rootAssembly.NodeSetFromNodeLabels

Required arguments

name
A String specifying the name of the set and the repository key.

nodeLabels
A sequence of node labels. A node label is a sequence of Int node identifiers. For example, for a part:

```
nodeLabels=(2,3,5,7)
```

For an assembly:

```
nodeLabels=(('Instance-1', (2,3,5,7)), ('Instance-2', (1,2,3)))
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

ElementSet(...)

This method creates an element set from an array of *OdbMeshElement* objects (for part instance-level sets) or from a sequence of arrays of *OdbMeshElement* objects (for assembly-level sets).

Path

session.odbs[*name*].parts[*name*].ElementSet

session.odbs[*name*].rootAssembly.instances[*name*].ElementSet

Required arguments

name
A String specifying the name of the set and the repository key.

elements

A sequence of *OdbMeshElement* objects. For example, for a part:

```
elements=instance1.elements[1:5]
```

For an assembly:

```
elements=(instance1.elements[1:5], instance2.elements[1:5])
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

ElementSetFromElementLabels(...)

This method creates an element set from a sequence of element labels.

Path

session.odbs[*name*].parts[*name*].ElementSetFromElementLabels

session.odbs[*name*].rootAssembly.instances[*name*].ElementSetFromElementLabels

session.odbs[*name*].rootAssembly.ElementSetFromElementLabels

Required arguments

name

A String specifying the name of the set and the repository key.

elementLabels

A sequence of element labels. An element label is a sequence of Int element identifiers. For example, for a part:

```
elementLabels=(2,3,5,7)
```

For an assembly:

```
elementLabels=(('Instance-1', (2,3,5,7)), ('Instance-2',
(1,2,3)))
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

MeshSurface(...)

This method creates a surface from the element and side identifiers for the assembly.

Path

session.odbs[*name*].parts[*name*].MeshSurface

session.odbs[*name*].rootAssembly.instances[*name*].MeshSurface

session.odbs[*name*].rootAssembly.MeshSurface

Required arguments

name

A String specifying the name of the set and the repository key.

meshSurfaces

A sequence of sequences. Each sequence consists of an element sequence and a side identifier. The possible side identifiers depend on the type of element, as described in the following table:

Sequence of elements	Side identifiers
Solid elements	FACE1, FACE2, FACE3, FACE4, FACE5, FACE6
Three-dimensional shell elements	SIDE1, SIDE2
Two-dimensional elements	FACE1, FACE2, FACE3, FACE4
Wire elements	END, END2

For example:

```
side1Elements=instance1.elements[217:218]
side2Elements=instance2.elements[100:105]
assembly.MeshSurface(name='Surf-1',
meshSurfaces=((side1Elems,SIDE1),
(side2Elems,SIDE2)))
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

MeshSurfaceFromElsets(...)

This method creates a mesh surface from a sequence of element sets.

Path

session.odbs[*name*].parts[*name*].MeshSurfaceFromElsets

session.odbs[*name*].rootAssembly.instances[*name*].MeshSurfaceFromElsets

session.odbs[*name*].rootAssembly.MeshSurfaceFromElsets

Required arguments

name
A String specifying the name of the set and the repository key.

elementSetSeq
A sequence of element sets. For example,

```
elementSetSeq=((elset1,SIDE1),(elset2,SIDE2))
```

where
`elset1=session.odbs[name].rootAssembly.elementSets['Clutch']`
and `SIDE1` and `SIDE2` indicate the side of the element set.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

MeshSurfaceFromLabels(...)

This method creates a mesh surface from a sequence of surface labels.

Path

session.odbs[*name*].parts[*name*].MeshSurfaceFromLabels

session.odbs[*name*].rootAssembly.instances[*name*].MeshSurfaceFromLabels

session.odbs[*name*].rootAssembly.MeshSurfaceFromLabels

Required arguments

name
A String specifying the name of the set and the repository key.

surfaceLabels
A sequence of surface labels. For example,

```
surfaceLabels=(('Instance-1', ((10, FACE1), (11, FACE2))),
('Instance-2', ((10, FACE3), (12, FACE4))))
```

where `10` is an element number and `FACE1` indicates the side of the element.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

Members

The OdbSet object can have the following members:

name
A String specifying the name of the set and the repository key.

instanceNames
A tuple of Strings specifying the namespaces for the nodes, elements, and faces; `None` if the set is on a Part or an *OdbInstance* object.

nodes
An *OdbMeshNodeArray* object specifying the nodes of an OdbSet. If a set spans more than one part instance, this member is a sequence of sequences for each part instance.

elements
An *OdbMeshElementArray* object specifying the elements of an OdbSet. If a set spans more than one part instance, this member is a sequence of sequences for each part instance.

faces
A tuple of SymbolicConstants specifying the element face. If a set spans more than one part instance, this member is a sequence of sequences for each part instance.

instances
A repository of an *OdbInstance* object.

isInternal
A Boolean specifying whether the set is internal.

OdbStep object

An output database contains the same steps of the model database that originated it.

Access

```
import odbAccess
session.odbs[name].steps[name]
```

Step(...)

This method creates an OdbStep object.

Path

```
session.odbs[name].Step
```

Required arguments

name
A String specifying the repository key.

description
A String specifying the step description.

domain
A SymbolicConstant specifying the domain of the step. Possible values are TIME, FREQUENCY, ARC_LENGTH, and MODAL.

The type of *OdbFrame* object that can be created for this step is based on the value of the *domain* argument.

Optional arguments

timePeriod
A Float specifying the time period of the step. *timePeriod* is required if *domain*=TIME; otherwise, this argument is not applicable. The default value is 0.0.

previousStepName
A String specifying the preceding step. If *previousStepName* is the empty string, the last step in the repository is used. If *previousStepName* is not the last step, this will result in a change to the *previousStepName* member of the step that was in that position. A special value 'Initial' refers to the internal initial model step and might be used exclusively for inserting a new step at the first position before any other existing steps. The default value is an empty string.

procedure
A String specifying the step procedure. The default value is an empty string. The following is the list of valid procedures:

```
*ANNEAL
*BUCKLE
*COMPLEX FREQUENCY
*COUPLED TEMPERATURE-DISPLACEMENT
*COUPLED TEMPERATURE-DISPLACEMENT, CETOL
*COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE
*COUPLED THERMAL-ELECTRICAL, STEADY STATE
*COUPLED THERMAL-ELECTRICAL
*COUPLED THERMAL-ELECTRICAL, DELTMX
*DYNAMIC
```

```
*DYNAMIC, DIRECT
*DYNAMIC, EXPLICIT
*DYNAMIC, SUBSPACE
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICT
*ELECTROMAGNETIC, HIGH FREQUENCY, TIME HARMONIC
*ELECTROMAGNETIC, LOW FREQUENCY, TIME DOMAIN
*ELECTROMAGNETIC, LOW FREQUENCY, TIME DOMAIN, DIRECT
*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
*FREQUENCY
*GEOSTATIC
*HEAT TRANSFER
*HEAT TRANSFER, DELTAMX=__
*HEAT TRANSFER, STEADY STATE
*MAGNETOSTATIC
*MAGNETOSTATIC, DIRECT
*MASS DIFFUSION
*MASS DIFFUSION, DCMAX=
*MASS DIFFUSION, STEADY STATE
*MODAL DYNAMIC
*RANDOM RESPONSE
*RESPONSE SPECTRUM
*SOILS
*SOILS, CETOL/UTOL
*SOILS, CONSOLIDATION
*SOILS, CONSOLIDATION, CETOL/UTOL
*STATIC
*STATIC, DIRECT
*STATIC, RIKS
*STEADY STATE DYNAMICS
*STEADY STATE TRANSPORT
*STEADY STATE TRANSPORT, DIRECT
*STEP PERTURBATION, *STATIC
*SUBSTRUCTURE GENERATE
*USA ADDDED MASS GENERATION
*VISCO
```

totalTime A Float specifying the analysis time spend in all the steps previous to this step. The default value is −1.0.

nlgeom A Boolean specifying the *nlgeom* value for all steps. The default value is True.

Return value

An OdbStep object.

Exceptions

If *previousStepName* is invalid: ValueError: previousStepName is invalid.

`getFrame(...)`

This method retrieves an *OdbFrame* object associated with a given frame value.

Required arguments

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the step time or frequency.

Optional arguments

match A SymbolicConstant specifying which frame to return if there is no frame at the exact frame value. Possible values are CLOSEST, BEFORE, AFTER, and EXACT. The default value is CLOSEST.

When *match*=CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

`getFrame(...)`

This method retrieves an *OdbFrame* object associated with a given load case.

Required arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

Optional arguments

None.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

`getFrame(...)`

This method retrieves an *OdbFrame* object associated with a given load case and frame value.

Required arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the step time or frequency.

Optional arguments

match A SymbolicConstant specifying which frame to return if there is no frame at the exact frame value. Possible values are CLOSEST, BEFORE, AFTER, and EXACT. The default value is CLOSEST.

When *match*=CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

`getHistoryRegion(...)`

This method retrieves a *HistoryRegion* object associated with a HistoryPoint in the model.

Required arguments

point A *HistoryPoint* object specifying the point in the model.

Optional arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

Return value

A *HistoryRegion* object.

Exceptions

If a *HistoryRegion* object is not found: OdbError: HistoryRegion not found.

`setDefaultDeformedField(...)`

This method sets the default deformed field variable in a step.

Required arguments

field A *FieldOutput* object specifying the default deformed field variable for visualization.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`setDefaultField(...)`

This method sets the default field variable in a step.

Required arguments

field A *FieldOutput* object specifying the default field variable for visualization.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The OdbStep object has members with the same names and descriptions as the arguments to the *Step* method.

In addition, the OdbStep object can have the following members:

number An Int specifying the step number.

nlgeom A Boolean specifying whether geometric nonlinearity can occur in this step.

mass A Float specifying the current value of the mass of the model. This does not include the mass of the acoustic media if any present.

acousticMass A Float specifying the current value of the mass of the acoustic media of the model.

frames An *OdbFrameArray* object.

historyRegions

A repository of *HistoryRegion* objects.

loadCases

A repository of *OdbLoadCase* objects.

massCenter

A tuple of Floats specifying the coordinates of the center of mass.

inertiaAboutCenter

A tuple of Floats specifying the moments and products of inertia about the center of mass. For 3D models inertia quantities are written in the following order: I(XX), I(YY), I(ZZ), I(XY), I(XZ), and I(YZ). For 2D models only I(ZZ) and I(XY) are outputted.

inertiaAboutOrigin

A tuple of Floats specifying the moments and products of inertia about the origin of the global coordinate system. For 3-D models inertia quantities are written in the following order: I(XX), I(YY), I(ZZ), I(XY), I(XZ), and I(YZ). For 2D models only I(ZZ) and I(XY) are outputted.

acousticMassCenter

A tuple of Floats specifying the coordinates of the center of mass of the acoustic media.

RebarOrientation object

The RebarOrientation object represents the orientation of the rebar reference.

Access

```
import odbAccess
session.odbs[name].parts[name].rebarOrientations[i]
session.odbs[name].rootAssembly.instances[name].rebarOrientations[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.rebarOrientations[i]
```

Members

The RebarOrientation object can have the following members:

axis
A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. Possible values are AXIS_1, AXIS_2, and AXIS_3.

angle
A Float specifying the angle of the additional rotation.

region
An *OdbSet* object specifying a region for which the rebar orientation is defined.

csys
An *OdbDatumCsys* object specifying a datum coordinates system.

ScratchOdb object

A scratch output database is associated with an open output database and is used to store session-related, non-persistent objects, such as Step, Frame and *FieldOutput* objects. Abaqus creates a scratch output database when needed for these non-persistent objects during an Abaqus/CAE session. Abaqus deletes the scratch output database when the associated output database is closed.

Access

```
import odbAccess
session.scratchOdbs[name]
```

ScratchOdb(...)

This method creates a new ScratchOdb object.

Path

```
session.ScratchOdb
```

Required arguments

odb An *Odb* object specifying the output database with which to associate.

Optional arguments

None.

Return value

A ScratchOdb object.

Exceptions

None.

Members

The ScratchOdb object has no members.

SectionCategory object

The SectionCategory object is used to group regions of the model with like sections.
Section definitions that contain the same number of section points or integration points are grouped together.

To access data for a particular section definition, use the individual *Section* objects in the output database. For more information, see *Beam Section profile commands* and *Section commands*.

Access

```
import
odbAccess
session.odbs[name].parts[name].elements[i].sectionCategory
session.odbs[name].parts[name].elementSets[name].elements[i]\
.sectionCategory
session.odbs[name].parts[name].nodeSets[name].elements[i]\
.sectionCategory
session.odbs[name].parts[name].surfaces[name].elements[i]\
.sectionCategory
session.odbs[name].rootAssembly.elements[i].sectionCategory
session.odbs[name].rootAssembly.elementSets[name].elements[i]\
.sectionCategory
session.odbs[name].rootAssembly.instances[name].elements[i]\
.sectionCategory
session.odbs[name].rootAssembly.instances[name].elementSets[name]\
.elements[i].sectionCategory
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.elements[i].sectionCategory
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.elements[i].sectionCategory
session.odbs[name].rootAssembly.nodeSets[name].elements[i]\
.sectionCategory
session.odbs[name].rootAssembly.surfaces[name].elements[i]\
.sectionCategory
session.odbs[name].sectionCategories[name]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elements[i].sectionCategory
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elementSets[name].elements[i].sectionCategory
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].elements[i].sectionCategory
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].elements[i].sectionCategory
```

SectionCategory(...)

This method creates a SectionCategory object.

Path

session.odbs[*name*].SectionCategory

Required arguments

name
A String specifying the name of the category.

description A String specifying the description of the category.

Optional arguments

None.

Return value

A SectionCategory object.

Exceptions

None.

Members

The SectionCategory object has members with the same names and descriptions as the arguments to the *SectionCategory* method.

In addition, the SectionCategory object can have the following member:

sectionPoints A *SectionPointArray* object.

SectionPoint object

The SectionPoint object describes the location of a section point within a section category.

Access

```
import
odbAccess
session.odbs[name].parts[name].elements[i].sectionCategory\
.sectionPoints[i]
session.odbs[name].parts[name].elementSets[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].parts[name].nodeSets[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].parts[name].surfaces[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.elements[i].sectionCategory\
.sectionPoints[i]
session.odbs[name].rootAssembly.elementSets[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.instances[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.instances[name].elementSets[name]\
.elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.nodeSets[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].rootAssembly.surfaces[name].elements[i]\
.sectionCategory.sectionPoints[i]
session.odbs[name].sectionCategories[name].sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name]\
.locations[i].sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.elementSets[name].elements[i].sectionCategory\
.sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.nodeSets[name].elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.surfaces[name].elements[i].sectionCategory.sectionPoints[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.sectionPoint
```

SectionPoint(...)

This method creates a SectionPoint object.

Path

session.odbs[*name*].sectionCategories[*name*].SectionPoint

Required arguments

number	An Int specifying the number of the section point. See *Beam Elements* and *Shell Elements* for the numbering convention.
description	A String specifying the description of the section point.

Optional arguments

None.

Return value

A SectionPoint object.

Exceptions

None.

Members

The SectionPoint object has members with the same names and descriptions as the arguments to the *SectionPoint* method.

SectorDefinition object

The SectorDefinition object describes the number of symmetry sectors and axis of symmetry for a cyclic symmetry model.

Access

```
import odbAccess
session.odbs[name].sectorDefinition
```

Members

The SectorDefinition object has the following members:

numSectors

An Int specifying the number of sectors in the cyclic symmetry model.

symmetryAxis

A tuple of tuples of Floats specifying the coordinates of two points on the axis of symmetry.

UserData object

The UserData object contains user-defined XY data. The UserData object has no constructor; it is created automatically when an *Odb* object is created.

Access

```
import odbAccess
session.odbs[name].userData
```

XYData(...)

This method creates an *XYData* object from a sequence of *X*–*Y* data pairs.

Path

session.odbs[name].userData.XYData

Required arguments

name A String specifying the repository key.

data A sequence of pairs of Floats specifying the *X*–*Y* data pairs.

Optional arguments

sourceDescription A String specifying the source of the *X*–*Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription A String specifying the content of the *X*–*Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription A String specifying additional information about the *X*–*Y* data (e.g., “for whole model”). The default value is an empty string.

legendLabel A String specifying the label to be used in the legend. The default value is the name of the XYData object.

xValuesLabel A String specifying the label for the X-values. This value may be overridden if the *X*–*Y* data are combined with other *X*–*Y* data. The default value is an empty string.

yValuesLabel A String specifying the label for the Y-values. This value may be overridden if the *X*–*Y* data are combined with other *X*–*Y* data. The default value is an empty string.

axis1QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the X -axis1-values.

axis2QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the Y -axis2-values.

Return value

An *XYData* object.

Exceptions

InvalidNameError.

Members

The UserData object can have the following members:

name
A String specifying the repository key.

sourceDescription
A String specifying the source of the *X–Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription
A String specifying the content of the *X–Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription
A String specifying additional information about the *X–Y* data (e.g., “for whole model”). The default value is an empty string.

xValuesLabel
A String specifying the label for the X-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

yValuesLabel
A String specifying the label for the Y-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

axis1QuantityType
A *QuantityType* object specifying the *QuantityType* object associated to the X -axis1- values.

axis2QuantityType
A *QuantityType* object specifying the *QuantityType* object associated to the Y -axis2- values.

legendLabel
A String specifying the label to be used in the legend. The default value is the name of the XYData object.

xyDataObjects
A repository of *XYData* objects.

annotations
A repository of *Annotation* objects.

data
A tuple of pairs of Floats specifying the *X–Y* data pairs.

Odb commands

The Odb commands do the following:

- Determine if an output database (`.odb`) file needs to be upgraded to the current release.
- Determine the extreme value for an output variable over a number of fields; for example, over a number of load cases.
- Open an existing output database file and create a new Odb object.
- Upgrade an output database file to the current release and write the upgraded output database to a new file.

isUpgradeRequiredForOdb

This method determines if an output database file needs to be upgraded to the current release.

You can access this method using either of the following techniques:

- From a script running outside Abaqus/CAE. For example,

```
import odbAccess
needsUpgrade = odbAccess.isUpgradeRequiredForOdb(
    upgradeRequiredOdbPath='myOdb.odb')
```

- From the Visualization module in Abaqus/CAE. For example,

```
import visualization
needsUpgrade = session.isUpgradeRequiredForOdb(
    upgradeRequiredOdbPath='myOdb.odb')
```

Required arguments

upgradeRequiredOdbPath An String specifying the path to an output database file to test. The test determines if the output database needs to be upgraded to the current release.

Optional arguments

None.

Return value

A Boolean indicating the result of the test. A value of True indicates that the output database needs to be upgraded to the current release.

Exceptions

None.

maxEnvelope

Retrieve the maximum value of an output variable over a number of fields.

Required arguments

The `maxEnvelope` method does not use keyword arguments.

Argument
A list of similar fieldOutput objects from which the maximum value will be computed.
A SymbolicConstant specifying the invariant or component label to be used when comparing vectors or tensors. Possible values are: • MAGNITUDE • MISES • TRESCA • PRESS • INV3 • MAX_PRINCIPAL • MID_PRINCIPAL • MIN_PRINCIPAL You must provide either this argument or the following argument if the field is a vector or tensor.
A String specifying the component of the tensor or vector to be used for selecting the maximum value.

Optional arguments

None.

Return value

A sequence of two fieldOutput objects. The first fieldOutput object contains the maximum value. The second fieldOutput object contains the index of the field containing the maximum value. The index follows the order in which fields are positioned in the list of fieldOutput objects provided as the argument to the function.

Exceptions

OdbError

TypeError	This function takes no keyword arguments.

minEnvelope

Retrieve the minimum value of an output variable over a number of fields.

Required arguments

The `minEnvelope` method does not use keyword arguments.

Argument
A list of similar fieldOutput objects from which the maximum value will be computed.
A SymbolicConstant specifying the invariant or component label to be used when comparing vectors or tensors. Possible values are: • MAGNITUDE • MISES • TRESCA

Argument
• PRESS • INV3 • MAX_PRINCIPAL • MID_PRINCIPAL • MIN_PRINCIPAL You must provide either this argument or the following argument if the field is a vector or tensor.
A String specifying the component of the tensor or vector to be used for selecting the minimum value.

Optional arguments

None.

Return value

A sequence of two fieldOutput objects. The first fieldOutput object contains the minimum value. The second fieldOutput object contains the index of the field containing the minimum value. The index follows the order in which fields are positioned in the list of fieldOutput objects provided as the argument to the function.

Exceptions

OdbError

TypeError This function takes no keyword arguments.

openOdb

This method opens an existing output database (`.odb`) file and creates a new Odb object. You typically execute this method outside of Abaqus/CAE when, in most cases, only one output database is open at any time. For example,

```
import odbAccess
shockLoadOdb = odbAccess.openOdb(path='myOdb.odb')
```

Required arguments

path A String specifying the path to an existing output database (`.odb`) file.

Optional arguments

readOnly A Boolean specifying whether the file will permit only read access or both read and write access. The initial value is False, indicating that both read and write access will be permitted.

readInternalSets A Boolean specifying whether the file will permit access to sets specified as Internal on the database. The initial value is False, indicating that internal sets will not be read.

Return value

An Odb object.

Exceptions

If the output database was generated by a previous release of Abaqus and needs upgrading:	OdbError: The database is from a previous release of Abaqus. Run `abaqus upgrade -job <newFilename> -odb <oldFileName>` to upgrade it.
If the output database was generated by a newer release of Abaqus, and the installation of Abaqus needs upgrading:	OdbError: Abaqus installation must be upgraded before this output database can be opened.

openOdb

This method opens an existing output database (`.odb`) file and creates a new Odb object. This method is accessed only via the session object inside Abaqus/CAE and adds the new Odb object to the session.odbs repository. This method allows you to open multiple output databases at the same time and to use the repository key to specify a particular output database. For example,

```
import visualization
session.openOdb(name='myOdb', path='stress.odb', readOnly=True)
```

Required arguments

name A String specifying the repository key. If the*name* is not the same as the*path* to the output database (`.odb`) file, the *path* must be specified as well. Additionally, to support backwards compatibility of the interface, if the *name* parameter is omitted, the *path* and *name* will be presumed to be the same.

Optional arguments

path A String specifying the path to an existing output database (`.odb`) file.

readOnly A Boolean specifying whether the file will permit only read access or both read and write access. The initial value is TRUE when the output database file is opened from Abaqus/CAE, indicating that only read access will be permitted.

Return value

An Odb object.

Exceptions

If the output database was generated by a previous release of Abaqus and needs upgrading:	OdbError: The database is from a previous release of Abaqus. Run `abaqus upgrade -job <newFilename> -odb <oldFileName>` to upgrade it.

If the output database was generated by a newer release of Abaqus, and the installation of Abaqus needs upgrading:	OdbError: Abaqus installation must be upgraded before this output database can be opened.
If the file is not a valid database:	AbaqusError: Cannot open file <*filename*>.

upgradeOdb

This method upgrades an existing Odb object to the current release and writes the upgraded version of the Odb object to a file. In addition, Abaqus/CAE writes information about the status of the upgrade to a log (`*.log`) file.

You can access this method using either of the following techniques:

- From a script running outside Abaqus/CAE. For example,

```
import odbAccess
odbAccess.upgradeOdb(existingOdbPath='oldOdb',
    upgradedOdbPath='upgradedOdb')
```

- From the session object in Abaqus/CAE. For example,

```
import visualization
session.upgradeOdb(existingOdbPath='oldOdb',
    upgradedOdbPath='upgradedOdb')
```

Required arguments

existingOdbPath	An String specifying the path to the file containing the output database to be upgraded.
upgradedOdbPath	An String specifying the path to the file that will contain the upgraded output database.

Optional arguments

None.

Return value

`None.`

Exceptions

If the output database upgrade fails:	OdbError: cannot convert database

Odb Display commands

Odb display commands are used to store the context of an output database for a viewport.

In this section:

- *OdbDisplay object*
- *CommonOptions object*
- *ContourOptions object*
- *DefaultOdbDisplay object*
- *DisplayBodyOptions object*
- *OrientationOptions object*
- *SuperimposeOptions object*
- *SymbolOptions object*
- *ViewCut object*
- *ViewerOptions object*

OdbDisplay object

The OdbDisplay object stores the context of an output database for a viewport. The OdbDisplay object has no constructor. Abaqus creates the *defaultOdbDisplay* member when you import the Visualization module. Abaqus creates the *odbDisplay* member when a viewport is created, using the attributes from the previous active viewport. The previous active viewport contains the attributes from the *defaultOdbDisplay* object for the session. The attributes from the *defaultOdbDisplay* object are copied from the previous active viewport to create the new viewport.

OdbDisplay objects are accessed in one of two ways:

- The default output database display options. These settings are used as defaults when other *odbDisplay* members are created and can be set to customize user preferences.
- The output database display options associated with a particular viewport.

Access

```
session.viewports[name].layers[name].odbDisplay
import visualization
session.viewports[name].odbDisplay
```

`moveCameraToCsys()`

This method specifies a new position for the camera. It is available only when *movieMode*=ON (in the *View* object). The new camera position is the origin of the coordinate system specified by the *cameraCsysName* member of the *BasicOptions* object.

Arguments

None.

Return value

`None.`

Exceptions

None.

`setDeformedVariable(...)`

This method specifies the field output variable or FieldOutput object to be used when displaying the deformed shape of the model.

Required arguments

One of the following two mutually exclusive arguments must be provided:

variableLabel A String specifying the field output variable.

field A String specifying the *FieldOutput* object.

Optional arguments

None.

Return value
None.

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.

setFrame(...)

This method specifies the step and frame for the OdbDisplay object.

Required arguments

step
An Int specifying the step index. Possible values are $0 \leq step \leq (numSteps - 1)$.

frame
An Int specifying the frame in the specified step. Valid values are $0 \leq frame \leq (numFramesInStep - 1)$. If $frame \geq (numFramesInStep - 1)$ the last frame will be displayed.

Optional arguments

None.

Return value
None.

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.
If the Odb object does not contain valid step data:	There are no valid step data on the odb. Requested operation canceled.
If an invalid step index is passed in as an argument:	Invalid step index:step. Available step indices: 0 - n

setFrame(...)

This method specifies the frame for the OdbDisplay object.

Required arguments

frame
An OdbFrame object.

Optional arguments

None.

Return value
None.

Exceptions
None.

setPrimaryVariable(...)

This method specifies the field output variable for which to obtain results.

Required arguments

variableLabel A String specifying the field output variable. The *variableLabel* and *field* arguments are mutually exclusive.

field A String specifying the FieldOutput object. The *variableLabel* and *field* arguments are mutually exclusive.

outputPosition A SymbolicConstant specifying the position from which to obtain data. Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, ELEMENT_CENTROID, WHOLE_ELEMENT, WHOLE_REGION, WHOLE_PART_INSTANCE, WHOLE_MODEL, and GENERAL_PARTICLE. Data can be obtained only from the position at which they were written to the output database during the analysis.

Optional arguments

refinement A sequence of a SymbolicConstant and a String. Possible values for the SymbolicConstant are INVARIANT and COMPONENT. The String specifies an available component or invariant for the specified *variableLabel*. This argument is required only if a refinement is available for the specified *variableLabel*.

sectionPoint A Dictionary with String keys and String values. Each key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
sectionPoint={'shell < MAT > < 7 section points >':'SPOS,
    (fraction = 1.0)', 'shell < MAT > < 5 section points >':
    'SPOS, (fraction = 1.0)', }
```

Return value
None.

Exceptions

If the viewport is not associated with any Odb object: The current viewport is not associated with an ODB file. Requested operation canceled.

setPrimarySectionPoint(...)

This method specifies the section point for the current primary, symbol and status variables.

Required arguments

sectionPoint

A Dictionary with String keys and String values. Each key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
sectionPoint={'shell < MAT > < 7 section points >':'SPOS,
    (fraction = 1.0)', 'shell < MAT > < 5 section points >':
    'SPOS, (fraction = 1.0)', }
```

activePly

A String specifying the active ply name.

Optional arguments

None.

Return value

`None.`

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.

setStatusVariable(...)

This method specifies the field output variable for filtering element display based on a status criteria.

Required arguments

variableLabel

A String specifying the field output variable. The *variableLabel* and *field* arguments are mutually exclusive.

field

A String specifying the FieldOutput object. The *variableLabel* and *field* arguments are mutually exclusive.

outputPosition

A SymbolicConstant specifying the position from which to obtain data. Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, ELEMENT_CENTROID, WHOLE_ELEMENT, WHOLE_REGION, WHOLE_PART_INSTANCE, WHOLE_MODEL, and GENERAL_PARTICLE. Data can be obtained only from the position at which they were written to the output database during the analysis.

Optional arguments

refinement

A sequence of a SymbolicConstant and a String. Possible values for the SymbolicConstant are INVARIANT and COMPONENT. The String specifies an available component or invariant for the specified *variableLabel*. This argument is required only if a refinement is available for the specified *variableLabel*.

sectionPoint

A Dictionary with String keys and String values. Each key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
sectionPoint={'shell < MAT > < 7 section points >':'SPOS,
    (fraction = 1.0)', 'shell < MAT > < 5 section points >':

    'SPOS, (fraction = 1.0)', }
```

statusMinimum

A Float specifying the minimum result value to be considered for element removal.

statusMaximum

A Float specifying the maximum result value to be considered for element removal.

statusInsideRange

A Boolean utilized when both *statusMinimum* and *statusMaximum* are given. Elements will be removed when they contain values between the minimum and maximum, inclusive, when true. Elements will be removed when they contain values outside of the minimum and maximum, exclusive, when false.

useStatus

A Boolean to specify if the status variable is to be active.

applyStatusToUndeformed

A Boolean to specify if the active status variable is to remove elements from undeformed plots. The default value is False.

Return value

`None.`

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.

`setSymbolVariable(...)`

This method specifies the field output variable for which to obtain results used for symbol plots. This variable must be in the form of vector or tensor data. The output quantity can also be specified with this command to control the display of resultants, or components.

Required arguments

variableLabel

A String specifying the field output variable. The *variableLabel* and *field* arguments are mutually exclusive.

field

A String specifying the FieldOutput object. The *variableLabel* and *field* arguments are mutually exclusive.

outputPosition

A SymbolicConstant specifying the position from which to obtain data. Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, ELEMENT_CENTROID, WHOLE_ELEMENT, WHOLE_REGION, WHOLE_PART_INSTANCE, WHOLE_MODEL, and GENERAL_PARTICLE. Data can be obtained only from the position at which they were written to the output database during the analysis.

Optional arguments

refinement

A sequence of a SymbolicConstant and a String. Possible values for the SymbolicConstant are INVARIANT and COMPONENT. The String specifies an available component or invariant for the specified *variableLabel*. This argument is required only if a refinement is available for the specified *variableLabel*.

sectionPoint

A Dictionary with String keys and String values. Each key specifies a region in the model; the corresponding value specifies a section point within that region. For example:

```
sectionPoint={'shell < MAT > < 7 section points >':'SPOS,
    (fraction = 1.0)', 'shell < MAT > < 5 section points >':
    'SPOS, (fraction = 1.0)', }
```

tensorQuantity

A SymbolicConstant specifying the tensor quantity to be displayed with the symbol. This value is set in the *SymbolOptions* object.

vectorQuantity

A SymbolicConstant specifying the vector quantity to be displayed with the symbol. This value is set in the *SymbolOptions* object.

Return value

`None.`

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.

`setStreamVariable(...)`

This method specifies the field output variable for which to obtain results used for stream plots. This variable must be in the form of nodal vector data.

Required arguments

variableLabel

A String specifying the field output variable.

Optional arguments

None.

Return value

```
None.
```

Exceptions

If the viewport is not associated with any Odb object:	The current viewport is not associated with an ODB file. Requested operation canceled.

setValues(...)

This method specifies member values for the OdbDisplay object.

Required arguments

None.

Optional arguments

visibleDisplayGroups	A List of *DisplayGroup* objects.
viewCut	A Boolean specifying whether to display the cut. The default value is OFF.
viewCutNames	A List of *ViewCut* objects.

Return value

```
None.
```

Exceptions

None.

Members

The OdbDisplay object can have the following members:

name	A String specifying the name of the output database associated with the OdbDisplay object.
fieldSteps	A tuple of Strings specifying field steps. Each item in the sequence consists of a tuple that contains the following step information: • *element0*: A String specifying the step name. • *element1*: A String specifying the step description. • *element2*: A Float specifying the time value at the start of the step. • *element3*: A Float specifying the time period of the step.

- *element4*: A Float specifying the user modified time period of the step.
- *element5*: An Int specifying the domain type of the step. Possible values are:
  - 0: Time domain
  - 1: Frequency domain
  - 2: Modal domain
  - 3: Arc Length (Riks) domain
- *element6*: A String specifying the default frame label.
- *element7*: A sequence of strings specifying the frame labels for all available frames in the step.
- *element8*: A sequence of floats specifying the frame values for all available frames in the step.
- *element9*: A Int specifying whether the step is user defined. Possible values are 0 indicating the step is defined in the analysis and 1 indicating the step is user defined.
- *element10*: A sequence of machine readable strings encoding the currently active frame numbers.

fieldVariables An *OdbFieldVarList* object.

modelVariableList An *OdbModelFieldVarList* object.

nodeSet A repository of *OdbSet* objects specifying the set label. The repository is read-only.

elementSet A repository of *OdbSet* objects specifying the set label. The repository is read-only.

surfaceSet A repository of *OdbSet* objects specifying the set label. The repository is read-only.

display A *DisplayOptions* object.

contourOptions A *ContourOptions* object.

commonOptions A *CommonOptions* object.

symbolOptions A *SymbolOptions* object.

superimposeOptions A *SuperimposeOptions* object.

displayBodyOptions A *DisplayBodyOptions* object.

freeBodyOptions A *FreeBodyOptions* object.

streamOptions A *StreamOptions* object.

viewCutOptions A *ViewCutOptions* object.

viewCuts

A repository of *ViewCut* objects.

displayGroup

A *DisplayGroup* object specifying the current display group and referring to an object in the *displayGroups* member of *Session*.

displayGroupInstances

A *DisplayGroupInstanceRepository* object.

basicOptions

A *BasicOptions* object.

materialOrientationOptions

An *OrientationOptions* object.

fieldFrame

A tuple of Strings specifying the step label and the frame label when the current step is user defined. Alternatively, *fieldFrame* maybe specified as a pair of Ints with the step index and the frame index, when the current step is defined in the analysis.

primaryVariable

A tuple specifying variables.

Each item in the sequence consists of a tuple containing the following elements:

- Element 0: A String specifying the variable label.
- Element 1: An Int specifying the output position. Possible integer values are:
  - 0: *UNDEFINED_POSITION*
  - 1: *NODAL*
  - 2: *INTEGRATION_POINT*
  - 3: *ELEMENT_FACE*
  - 4: *ELEMENT_NODAL*
  - 5: *WHOLE ELEMENT*
  - 6: *ELEMENT_CENTROID*
  - 7: *WHOLE_REGION*
  - 8: *WHOLE_PART_INSTANCE*
  - 9: *WHOLE_MODEL*
  - 10:*GENERAL_PARTICLE*
- Element 2: An Int specifying the data type. Possible values are:
  - 0: *ENUMERATION*
  - 1: *BOOLEAN*
  - 2: *INTEGER*
  - 3: *SCALAR*
  - 4: *VECTOR*
  - 5: *QUATERNION_2D*
  - 6: *QUATERNION_3D*
  - 7: TENSOR
  - 8: TENSOR_3D_FULL

  - 9: TENSOR_3D_PLANAR
  - 10: TENSOR_3D_SURFACE
  - 11: TENSOR_2D_PLANAR
  - 12: TENSOR_2D_SURFACE

- Element 3: An Int specifying the storage type. Possible values are:
  - 0: *FLOAT*
  - 1: *DOUBLE*
  - 2: *INTEGER*
  - 3: *BOOLEAN*

- Element 4: An Int specifying the refinement type. Possible values are:
  - 0: *NO_REFINEMENT*
  - 1: *INVARIANT*
  - 2: *COMPONENT*

- Element 5: A String specifying the refinement label.
- Element 6: An Int specifying the refinement index.
- Element 7: An Int specifying whether section point information is available. Possible values are 1 when section point information is available; 0, when this information is unavailable.
- Element 8: A sequence of a String specifying the name of the ply and category selection tuples (see below) specifying the section point information.

  A category selection tuple consists of the following elements:
  - Element 0: A String specifying the category label.
  - Element 1: An Int specifying whether to use both top and bottom section points to obtain results. Possible values are 1 to use both section points and 0 to use only the top section point.
  - Element 2: An Int specifying the top section point index.
  - Element 3: A String specifying the top section label.
  - Element 4: An Int specifying the bottom section point index.
  - Element 5: A String specifying the bottom section label.
  - Element 6: An Int specifying the category id.

- Element 9: An Int specifying whether the data are complex. Possible values are 1 when the data are complex; 0, when the data is not complex.
- Element 10: A Float specifying the minimum possible value for the data.
- Element 11: A Float specifying the maximum possible value for the data.
- Element 12: An Int specifying whether the data is derived. Possible values are 1 when the data is derived; 0, when the data is not derived.
- Element 13: An Int specifying whether the variable is a surface variable. Possible values are 1 when the variable is a surface variable; 0, when the variable is not a surface variable.

deformedVariable A tuple specifying variables.

For information on the sequence, see the member *primaryVariable*.

statusVariable A tuple of SymbolicConstants specifying variables.

For information on the sequence, see the member *primaryVariable*.

symbolVariable A tuple of SymbolicConstants specifying variables.

For information on the sequence, see the member *primaryVariable*.

applyStatusToUndeformed A tuple of SymbolicConstants specifying a Boolean to specify if elements are to be removed in undeformed states based on an active status variable

statusInsideRange A tuple of SymbolicConstants specifying a Boolean to specify if the status range should be inside a specified minimum and maximum. The range will be outside when false.

statusMinimum A tuple of Floats specifying a Float value for the minimum status range value.

statusMaximum A tuple of Floats specifying a Float value for the maximum status range value.

useStatus A tuple of SymbolicConstants specifying a Boolean to specify if elements are to be removed based on the status variable

firstFrame A pair of Ints specifying the step index and the frame index of the first available frame. This sequence is read-only.

prevFrame A pair of Ints specifying the step index and the frame index of the frame previous to the current frame. This sequence is read-only.

nextFrame A pair of Ints specifying the step index and the frame index of the frame following the current frame. This sequence is read-only.

lastFrame A pair of Ints specifying the step index and the frame index of the last available frame. This sequence is read-only.

CommonOptions object

The CommonOptions object stores values and attributes that are common to all plot states. The CommonOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.commonOptions* member when you import the Visualization module. Abaqus creates a *commonOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.commonOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

CommonOptions objects are accessed in one of two ways:

- The default common options. These settings are used as defaults when other *commonOptions* members are created. These settings can be set to customize user preferences.
- The common options associated with a particular viewport.

The CommonOptions object is derived from the *DGCommonOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.commonOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].layers[name].odbDisplay.commonOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].odbDisplay.commonOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
```

`setValues(...)`

This method modifies the CommonOptions object.

Required arguments

None.

Optional arguments

options

A CommonOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is SHADED.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

deformationScaling

A SymbolicConstant specifying the deformation scale factor mode. Possible values are AUTO, UNIFORM, and NONUNIFORM. The default value is AUTO.

uniformScaleFactor

A Float specifying the uniform deformation scaling constant when *deformationScaling*=UNIFORM. The default value is *autoDeformationScaleValue*.

nonuniformScaleFactor

A sequence of three Floats specifying the deformation scaling in each of the three coordinate directions when *deformationScaling*=NONUNIFORM. The default value is (*autoDeformationScaleValue*, *autoDeformationScaleValue*, *autoDeformationScaleValue*).

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

labelFont

A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

elemLabelColor

A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

faceLabelColor

A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeLabelColor

A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolColor

A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

coordinateScaleFactors

A sequence of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

normals

A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay

A SymbolicConstant specifying whether to draw element normals or surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color
A String specifying the color to be used to plot an arrow along the beam n_1-direction. The default value is "Blue".

beamN2Color
A String specifying the color to be used to plot an arrow along the beam n_2-direction. The default value is "Red".

beamTangentColor
A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

normalArrowLength
A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness
A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle
A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency
A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor
A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

Return value
`None.`

Exceptions
RangeError.

Members
The CommonOptions object can have the following members:

deformationScaling
A SymbolicConstant specifying the deformation scale factor mode. Possible values are AUTO, UNIFORM, and NONUNIFORM. The default value is AUTO.

uniformScaleFactor
A Float specifying the uniform deformation scaling constant when *deformationScaling*=UNIFORM. The default value is *autoDeformationScaleValue*.

autoDeformationScaleValue
A Float specifying the deformation scale factor value when *deformationScaling*=AUTO. This value is read-only.

nonuniformScaleFactor
A tuple of three Floats specifying the deformation scaling in each of the three coordinate directions when *deformationScaling*=NONUNIFORM. The default value is (*autoDeformationScaleValue*, *autoDeformationScaleValue*, *autoDeformationScaleValue*).

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is SHADED.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor
An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are 0≤*elementShrinkPercentage*≤ 90. The default value is 5.

coordinateScale
A Boolean specifying whether to scale coordinates. The default value is OFF.

normals
A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay
A SymbolicConstant specifying whether to draw element normals or surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

normalArrowLength
A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness
A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle
A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency
A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor
A Float specifying the translucency factor when *translucency*=ON. Possible values are 0.0≤*translucencyFactor*≤ 1.0. The default value is 0.3.

edgeColorWireHide
A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade
A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor
A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

labelFont
A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabelColor
A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabelColor
A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabelColor
A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbolColor
A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color

A String specifying the color to be used to plot an arrow along the beam n_1-direction. The default value is "Blue".

beamN2Color

A String specifying the color to be used to plot an arrow along the beam n_2-direction. The default value is "Red".

beamTangentColor

A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

ContourOptions object

The ContourOptions object stores values and attributes associated with a contour plot. The ContourOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.contourOptions* member when you import the Visualization module. Abaqus creates a *contourOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.contourOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

ContourOptions objects are accessed in one of two ways:

- The default contour options. These settings are used as defaults when other *contourOptions* members are created. These settings can be set to customize user preferences.
- The contour options associated with a particular viewport.

The ContourOptions object is derived from the *DGContourOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.contourOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].layers[name].odbDisplay.contourOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].odbDisplay.contourOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
```

`setValues(...)`

This method modifies the ContourOptions object.

Required arguments

None.

Optional arguments

options

A ContourOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

contourType

A SymbolicConstant specifying the contour type. Possible values are LINE, BANDED, QUILT, and ISOSURFACE. The default value is BANDED.

contourMethod

A SymbolicConstant specifying the contour rendering method. Possible values are TEXTURE_MAPPED and TESSELLATED. The default value is TEXTURE_MAPPED.

tickmarkPlots

A Boolean specifying whether tick mark plots should be displayed on line-type elements. If *tickmarkPlots*=ON, Abaqus displays a tick mark plot. If *tickmarkPlots*=OFF, Abaqus displays contours on the element faces. The default value is OFF.

contourStyle

A SymbolicConstant specifying the interval style of the contour plot. Possible values are CONTINUOUS and UNIFORM. The default value is UNIFORM.

numIntervals

An Int specifying the number of intervals when *contourStyle*=UNIFORM. Possible values are $2 \leq$*numIntervals*≤ 24. The default value is 12.

intervalType

A SymbolicConstant specifying the interval type of the contour plot. Possible values are UNIFORM, LOG, and USER_DEFINED. The default value is UNIFORM.

intervalValues

A sequence of Floats specifying the interval values when *intervalType*=USER_DEFINED.

maxAutoCompute

A Boolean specifying whether the contour range maximum value should be computed from the output data to be contoured. The default value is ON.

maxValue

A Float specifying the contour range maximum value to be used in the plot when *maxAutoCompute*=ON. The default value is *autoMaxValue*.

minAutoCompute

A Boolean specifying whether the contour range minimum value should be computed from the output data to be contoured. The default value is ON.

minValue

A Float specifying the contour range minimum value to be used in the plot when *minAutoCompute*=ON. The default value is *autoMinValue*.

animationAutoLimits

A SymbolicConstant specifying the method to be used when contour limits are automatically computed for animation. *animationAutoLimits* will only effect the minimum limit and/or maximum limit when *minAutoCompute* and/or *maxAutoCompute*=True. Possible values are FIRST_AND_LAST, CURRENT_FRAME, RECOMPUTE_EACH_FRAME, and ALL_FRAMES. The default value is ALL_FRAMES.

edgeColorLine

A String specifying the edge color to be used when *contourType*=LINE. The default value is "White".

edgeColorBandedQuilt

A String specifying the edge color to be used when *contourType*=BANDED or QUILT. The default value is "Black".

spectrum

A String specifying the name of the color spectrum to be used in the contour plot. The default value is "Rainbow".

reversedContourLegendRange

A Boolean specifying whether the contour legend should show the lowest value at the top and the highest value at the bottom (*reversedContourLegendRange*=ON) or vice versa. The default value is OFF.

outsideLimitsMode

A SymbolicConstant specifying the color of contour values that exceed the limits of the plot. Possible values are SPECTRUM and SPECIFY.

When *outsideLimitsMode*=SPECTRUM, the maximum and minimum contour spectrum colors are used for values that exceed the limits of the plot. When *outsideLimitsMode*=SPECIFY, the values of *outsideLimitsAboveColor* and *outsideLimitsBelowColor* are used for values that exceed the limits of the plot.

outsideLimitsAboveColor

A String specifying the color to be used for values that exceed the limits of the plot when *outsideLimitsMode*=SPECIFY. The default value is "Grey80".

outsideLimitsBelowColor

A String specifying the color to be used for values that exceed the limits of the plot when *outsideLimitsMode*=SPECIFY. The default value is "Grey20".

intervalLineAttributes

A sequence of sequences of SymbolicConstants specifying the line style and line thickness for each interval in the plot when *contourType*=LINE. The size of the outer sequence must be equal to *numIntervals*-1. The inner sequence consists of two SymbolicConstants specifying the line style and line thickness. For possible values, refer to the *edgeLineStyle* and *edgeLineThickness* members of the *DGCommonOptions* object. The default is ((SOLID, VERY_THIN),).

contourEdges

A Boolean specifying whether to plot the edges of each contour interval when *contourType*=BANDED or ISOSURFACE. The default value is OFF.

contourEdgeColor

A String specifying the color to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. The default value is "Grey60".

contourEdgeStyle

A SymbolicConstant specifying the edge line style to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

contourEdgeThickness

A SymbolicConstant specifying the edge line thickness to be used to plot the edge of the contour intervals when *contourType*=BANDED or ISOSURFACE. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

tickmarkAxisLength

A SymbolicConstant specifying the length of the tick mark plot axes. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

tickmarkBaseValue

A Float specifying the base contour value defining the tick mark axis contour value that intersects the elements. Possible values are *autoMinValue*≤*tickmarkBaseValue*≤*autoMaxValue*. The default value is 0.0.

tickmarkOrientation

A SymbolicConstant specifying the orientation of the tick mark plots. Possible values are N1 and N2. The default value is N2.

tickmarkCurveColor A String specifying the color to be used to plot the tick mark curve. The default value is "Cyan".

averagedOrientationDisplay A Boolean specifying the display of the nodal averaged coordinate systems used when averaging element vector or tensor data. The default value is OFF.

extrapolatedAveraging A Boolean specifying whether to auto-compute contour limits using extrapolated values alone or extrapolated values that are averaged. The default value is OFF.

showMaxLocation A Boolean specifying whether to display location of maximum value. The default value is OFF.

showMinLocation A Boolean specifying whether to display location of minimum value. The default value is OFF.

legendHideOutsideLimits A Boolean specifying whether to hide the values outside the specified minimum/maximum for the contour interval when *legendHideOutsideLimits*=ON. The default value is OFF.

Return value

`None.`

Exceptions

RangeError.

Members

The ContourOptions object can have the following members:

contourType A SymbolicConstant specifying the contour type. Possible values are LINE, BANDED, QUILT, and ISOSURFACE. The default value is BANDED.

numIntervals An Int specifying the number of intervals when *contourStyle*=UNIFORM. Possible values are $2 \leq$*numIntervals*≤ 24. The default value is 12.

intervalType A SymbolicConstant specifying the interval type of the contour plot. Possible values are UNIFORM, LOG, and USER_DEFINED. The default value is UNIFORM.

maxAutoCompute A Boolean specifying whether the contour range maximum value should be computed from the output data to be contoured. The default value is ON.

maxValue A Float specifying the contour range maximum value to be used in the plot when *maxAutoCompute*=ON. The default value is *autoMaxValue*.

minAutoCompute A Boolean specifying whether the contour range minimum value should be computed from the output data to be contoured. The default value is ON.

minValue

A Float specifying the contour range minimum value to be used in the plot when *minAutoCompute*=ON. The default value is *autoMinValue*.

animationAutoLimits

A SymbolicConstant specifying the method to be used when contour limits are automatically computed for animation. *animationAutoLimits* will only effect the minimum limit and/or maximum limit when *minAutoCompute* and/or *maxAutoCompute*=True. Possible values are FIRST_AND_LAST, CURRENT_FRAME, RECOMPUTE_EACH_FRAME, and ALL_FRAMES. The default value is ALL_FRAMES.

outsideLimitsMode

A SymbolicConstant specifying the color of contour values that exceed the limits of the plot. Possible values are SPECTRUM and SPECIFY.

When *outsideLimitsMode*=SPECTRUM, the maximum and minimum contour spectrum colors are used for values that exceed the limits of the plot. When *outsideLimitsMode*=SPECIFY, the values of *outsideLimitsAboveColor* and *outsideLimitsBelowColor* are used for values that exceed the limits of the plot.

extrapolatedAveraging

A Boolean specifying whether to auto-compute contour limits using extrapolated values alone or extrapolated values that are averaged. The default value is OFF.

showMaxLocation

A Boolean specifying whether to display location of maximum value. The default value is OFF.

showMinLocation

A Boolean specifying whether to display location of minimum value. The default value is OFF.

autoMaxValue

A Float specifying the maximum value to be used in the plot. The value is computed from the output data to be contoured. This value is read-only.

autoMinValue

A Float specifying the minimum value to be used in the plot. The value is computed from the output data to be contoured. This value is read-only.

outsideLimitsAboveColor

A String specifying the color to be used for values that exceed the limits of the plot when *outsideLimitsMode*=SPECIFY. The default value is "Grey80".

outsideLimitsBelowColor

A String specifying the color to be used for values that exceed the limits of the plot when *outsideLimitsMode*=SPECIFY. The default value is "Grey20".

spectrum

A String specifying the name of the color spectrum to be used in the contour plot. The default value is "Rainbow".

reversedContourLegendRange

A Boolean specifying whether the contour legend should show the lowest value at the top and the highest value at the bottom (*reversedContourLegendRange*=ON) or vice versa. The default value is OFF.

intervalValues

A tuple of Floats specifying the interval values when *intervalType*=USER_DEFINED.

contourMethod
A SymbolicConstant specifying the contour rendering method. Possible values are TEXTURE_MAPPED and TESSELLATED. The default value is TEXTURE_MAPPED.

tickmarkPlots
A Boolean specifying whether tick mark plots should be displayed on line-type elements. If *tickmarkPlots*=ON, Abaqus displays a tick mark plot. If *tickmarkPlots*=OFF, Abaqus displays contours on the element faces. The default value is OFF.

contourStyle
A SymbolicConstant specifying the interval style of the contour plot. Possible values are CONTINUOUS and UNIFORM. The default value is UNIFORM.

contourEdges
A Boolean specifying whether to plot the edges of each contour interval when *contourType*=BANDED or ISOSURFACE. The default value is OFF.

contourEdgeStyle
A SymbolicConstant specifying the edge line style to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

contourEdgeThickness
A SymbolicConstant specifying the edge line thickness to be used to plot the edge of the contour intervals when *contourType*=BANDED or ISOSURFACE. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

averagedOrientationDisplay
A Boolean specifying the display of the nodal averaged coordinate systems used when averaging element vector or tensor data. The default value is OFF.

matchingPlyLabels
A Boolean specifying whether the label for the matching ply shows up in the viewport. The default value is OFF.

colorByMatchingPlies
A Boolean specifying whether the contour color is driven by the matching ply. The default value is OFF.

tickmarkAxisLength
A SymbolicConstant specifying the length of the tick mark plot axes. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

tickmarkBaseValue
A Float specifying the base contour value defining the tick mark axis contour value that intersects the elements. Possible values are
autoMinValue$\leq$*tickmarkBaseValue*$\leq$*autoMaxValue*. The default value is 0.0.

tickmarkOrientation
A SymbolicConstant specifying the orientation of the tick mark plots. Possible values are N1 and N2. The default value is N2.

edgeColorLine
A String specifying the edge color to be used when *contourType*=LINE. The default value is "White".

edgeColorBandedQuilt
A String specifying the edge color to be used when *contourType*=BANDED or QUILT. The default value is "Black".

contourEdgeColor

A String specifying the color to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. The default value is "Grey60".

tickmarkCurveColor

A String specifying the color to be used to plot the tick mark curve. The default value is "Cyan".

intervalLineAttributes

A tuple of tuples of SymbolicConstants specifying the line style and line thickness for each interval in the plot when *contourType*=LINE. The size of the outer sequence must be equal to *numIntervals*-1. The inner sequence consists of two SymbolicConstants specifying the line style and line thickness. For possible values, refer to the *edgeLineStyle* and *edgeLineThickness* members of the *DGCommonOptions* object. The default is ((SOLID, VERY_THIN),).

legendHideOutsideLimits

A Boolean specifying whether to hide the values outside the specified min/max in the contour legend. This setting hides the *autoMinValue* and *autoMaxValue* from the spectrum when *legendHideOutsideLimits*=ON.The default value is OFF.

DefaultOdbDisplay object

The DefaultOdbDisplay object is a limited-functionality version of the *OdbDisplay* object.

Access

```
import visualization
session.defaultOdbDisplay
```

Members

The DefaultOdbDisplay object has the following members:

basicOptions	A *BasicOptions* object.
commonOptions	A *CommonOptions* object.
contourOptions	A *ContourOptions* object.
displayBodyOptions	A *DisplayBodyOptions* object.
freeBodyOptions	A *FreeBodyOptions* object.
streamOptions	A *StreamOptions* object.
materialOrientationOptions	An *OrientationOptions* object.
superimposeOptions	A *SuperimposeOptions* object.
symbolOptions	A *SymbolOptions* object.
viewCutOptions	A *ViewCutOptions* object.

DisplayBodyOptions object

The DisplayBodyOptions object stores values and attributes that are common to all plot states. The DisplayBodyOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.displayBodyOptions* member when you import the Visualization module. Abaqus creates a *displayBodyOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.displayBodyOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

DisplayBodyOptions objects are accessed in one of two ways:

- The default display body options. These settings are used as defaults when other *displayBodyOptions* members are created. These settings can be set to customize user preferences.
- The display body options associated with a particular viewport.

The DisplayBodyOptions object is derived from the *DGDisplayBodyOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.displayBodyOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].odbDisplay.displayBodyOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].odbDisplay.displayBodyOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
```

`setValues(...)`

This method modifies the DisplayBodyOptions object.

Required arguments

None.

Optional arguments

options

A DisplayBodyOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

edgeColorWireHide
A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade
A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

edgeLineStyle
A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness
A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

fillColor
A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

colorCodeOverride
A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elementShrink
A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor
An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale
A Boolean specifying whether to scale coordinates. The default value is OFF.

coordinateScaleFactors
A sequence of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

translucency
A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor
A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

Return value
`None.`

Exceptions
RangeError.

Members
The DisplayBodyOptions object can have the following members:

visibleEdges
A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \le$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

translucency

A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor

A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \le$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

OrientationOptions object

The OrientationOptions object stores values and attributes associated with a material orientation plot. The OrientationOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.materialOrientationOptions* member when you import the Visualization module. Abaqus creates a *materialOrientationOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.materialOrientationOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

OrientationOptions objects are accessed in one of two ways:

- The default material orientation options. These settings are used as defaults when other *materialOrientationOptions* members are created. These settings can be set to customize user preferences.
- The material orientation options associated with a particular viewport.

The OrientationOptions object is derived from the *DGOrientationOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.materialOrientationOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].odbDisplay\
.materialOrientationOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
session.viewports[name].odbDisplay.materialOrientationOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
```

`setValues(...)`

This method modifies the OrientationOptions object.

Required arguments

None.

Optional arguments

options

An OrientationOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

axis1Color

A String specifying the color of axis 1 of the material orientation triad. The default value is "Cyan".

showAxis1
A Boolean specifying whether axis 1 of the material orientation triad should be displayed. The default value is ON.

axis2Color
A String specifying the color of axis 2 of the material orientation triad. The default value is "Yellow".

showAxis2
A Boolean specifying whether axis 2 of the material orientation triad should be displayed. The default value is ON.

axis3Color
A String specifying the color of axis 3 of the material orientation triad. The default value is "Red".

showAxis3
A Boolean specifying whether axis 3 of the material orientation triad should be displayed. The default value is ON.

symbolSize
A Float specifying the size of the material orientation triad. The default value is 12.0.

lineThickness
A SymbolicConstant specifying the thickness of the material orientation triad. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

orientation
A SymbolicConstant specifying the orientation used for composites. Possible values are PLY and LAYUP. The default value is PLY.

arrowheadStyle
A SymbolicConstant specifying the arrowhead style for the material orientation triad. Possible values are NONE, FILLED, and WIRE. The default value is NONE.

scaleMode
A SymbolicConstant specifying the scaling basis for the material orientation triad. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

Return value
`None.`

Exceptions
RangeError.

Members
The OrientationOptions object can have the following members:

showAxis1
A Boolean specifying whether axis 1 of the material orientation triad should be displayed. The default value is ON.

showAxis2
A Boolean specifying whether axis 2 of the material orientation triad should be displayed. The default value is ON.

showAxis3
A Boolean specifying whether axis 3 of the material orientation triad should be displayed. The default value is ON.

symbolSize

A Float specifying the size of the material orientation triad. The default value is 12.0.

lineThickness

A SymbolicConstant specifying the thickness of the material orientation triad. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

orientation

A SymbolicConstant specifying the orientation used for composites. Possible values are PLY and LAYUP. The default value is PLY.

arrowheadStyle

A SymbolicConstant specifying the arrowhead style for the material orientation triad. Possible values are NONE, FILLED, and WIRE. The default value is NONE.

scaleMode

A SymbolicConstant specifying the scaling basis for the material orientation triad. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

axis1Color

A String specifying the color of axis 1 of the material orientation triad. The default value is "Cyan".

axis2Color

A String specifying the color of axis 2 of the material orientation triad. The default value is "Yellow".

axis3Color

A String specifying the color of axis 3 of the material orientation triad. The default value is "Red".

SuperimposeOptions object

The SuperimposeOptions object stores values and attributes associated with the undeformed shape when the display contains both the deformed shape and the undeformed shape. The SuperimposeOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.superimposeOptions* member when you import the Visualization module. Abaqus creates a *superimposeOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.superimposeOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

SuperimposeOptions objects are accessed in one of two ways:

- The default superimpose options. These settings are used as defaults when other *superimposeOptions* members are created. These settings can be set to customize user preferences.
- The superimpose options associated with a particular viewport.

The SuperimposeOptions object is derived from the *DGSuperimposeOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.superimposeOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].odbDisplay.superimposeOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
session.viewports[name].odbDisplay.superimposeOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
```

`setValues(...)`

This method modifies the SuperimposeOptions object.

Required arguments

None.

Optional arguments

options

A SuperimposeOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is WIREFRAME.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is FEATURE.

NONE can be used only when *renderStyle*=SHADED.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "Green".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "Green".

labelFont

A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

elemLabelColor

A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

faceLabelColor

A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeLabelColor

A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI

- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolColor

A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

coordinateScaleFactors

A sequence of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

normals

A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay

A SymbolicConstant specifying whether to draw element normals and surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_1}$-direction. The default value is "Blue".

beamN2Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_2}$-direction. The default value is "Red".

beamTangentColor

A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

normalArrowLength

A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq translucencyFactor \leq 1.0$. The default value is 0.3.

deformedOffsetMode A SymbolicConstant specifying the offset of the deformed shape from the undeformed shape. Possible values are NONE, UNIFORM, and NONUNIFORM. The default value is NONE.

NONE specifies that there should be no offset of the deformed shape from the undeformed shape.

uniformOffset A Float specifying the uniform offset value when *deformedOffsetMode*=UNIFORM. The default value is 0.0.

nonuniformOffset A sequence of three Floats specifying the offset value in each of the three coordinate directions when *deformedOffsetMode*=NONUNIFORM. The default value is (0, 0, 0).

Return value

`None.`

Exceptions

RangeError.

Members

The SuperimposeOptions object can have the following members:

deformedOffsetMode A SymbolicConstant specifying the offset of the deformed shape from the undeformed shape. Possible values are NONE, UNIFORM, and NONUNIFORM. The default value is NONE.

NONE specifies that there should be no offset of the deformed shape from the undeformed shape.

uniformOffset A Float specifying the uniform offset value when *deformedOffsetMode*=UNIFORM. The default value is 0.0.

nonuniformOffset A tuple of three Floats specifying the offset value in each of the three coordinate directions when *deformedOffsetMode*=NONUNIFORM. The default value is (0, 0, 0).

renderStyle A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is WIREFRAME.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is FEATURE.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

normals

A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay

A SymbolicConstant specifying whether to draw element normals and surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

normalArrowLength

A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness

A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle

A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency

A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor

A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "Green".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "Green".

labelFont

A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabelColor

A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabelColor

A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabelColor

A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbolColor

A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color

A String specifying the color to be used to plot an arrow along the beam n_1-direction. The default value is "Blue".

beamN2Color

A String specifying the color to be used to plot an arrow along the beam n_2-direction. The default value is "Red".

beamTangentColor

A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

SymbolOptions object

The SymbolOptions object stores values and attributes associated with a symbol plot. The SymbolOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.symbolOptions* member when you import the Visualization module. Abaqus creates a *symbolOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.symbolOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

SymbolOptions objects are accessed in one of two ways:

- The default symbol options. These settings are used as defaults when other *symbolOptions* members are created. These settings can be set to customize user preferences.
- The symbol options associated with a particular viewport.

The SymbolOptions object is derived from the *DGSymbolOptions* object.

Access

```
import visualization
session.defaultOdbDisplay.symbolOptions
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].odbDisplay.symbolOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
session.viewports[name].odbDisplay.symbolOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
```

`setValues(...)`

This method modifies the SymbolOptions object.

Required arguments

None.

Optional arguments

options

A SymbolOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

vectorQuantity

A SymbolicConstant specifying the vector quantity to display. Possible values are RESULTANT and VECTOR_COMPONENT. The default value is RESULTANT.

vectorLineThickness
A SymbolicConstant specifying the vector line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

vectorArrowheadStyle
A SymbolicConstant specifying the vector arrowhead style. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

vectorColor
A String specifying the vector color. The default value is "Red".

vectorColorMethod
A SymbolicConstant specifying the vector color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

vectorColorSpectrum
A String specifying the vector color spectrum name. The default value is "Rainbow".

vectorIntervalNumber
An Int specifying the number of color intervals for vector symbols. The default value is 12.

symbolDensity
A Float specifying the factor for randomized sampling. The default value is 1.0.

constantLengthArrows
A Boolean specifying whether to use constant-length arrows for vector symbols. The default value is OFF.

tensorColorMethod
A SymbolicConstant specifying the tensor color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

tensorColorSpectrum
A String specifying the tensor color spectrum name. The default value is "Rainbow".

tensorIntervalNumber
An Int specifying the number of color intervals for tensor symbols. The default value is 12.

vectorMaxValueAutoCompute
A Boolean specifying whether the maximum vector value is to be computed automatically. The default value is ON.

vectorMaxValue
A Float specifying the user-specified maximum vector value. The default value is *autoVectorMaxValue*.

vectorMinValueAutoCompute
A Boolean specifying whether the minimum vector value is to be computed automatically. The default value is ON.

vectorMinValue
A Float specifying the user-specified minimum vector value. The default value is *autoVectorMinValue*.

tensorQuantity
A SymbolicConstant specifying the tensor quantity to display. Possible values are ALL_PRINCIPAL_COMPONENTS, PRINCIPAL_COMPONENT, ALL_DIRECT_COMPONENTS, and DIRECT_COMPONENT. The default value is ALL_PRINCIPAL_COMPONENTS.

arrowSymbolSize
An Int specifying the length of vector and tensor symbols. The default value is 6.

tensorMaxPrinColor A String specifying the color of the maximum principal tensor symbols. The default value is "Red".

tensorMinPrinColor A String specifying the color of the minimum principal tensor symbols. The default value is "Cyan".

tensorMidPrinColor A String specifying the color of the intermediate principal tensor symbols. The default value is "Yellow".

tensorSelectedPrinColor A String specifying the color of the selected principal tensor symbols. The default value is "Red".

tensorLineThickness A SymbolicConstant specifying the line thickness of the tensor symbols. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

tensorArrowheadStyle A SymbolicConstant specifying the arrowhead style of the tensor symbols. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

tensorMaxValueAutoCompute A Boolean specifying whether the maximum tensor value is to be computed automatically. The default value is ON.

tensorMaxValue A Float specifying the user-specified maximum tensor value. The default value is *autoTensorMaxValue*.

tensorMinValueAutoCompute A Boolean specifying whether the minimum tensor value is to be computed automatically. The default value is ON.

tensorMinValue A Float specifying the user-specified minimum tensor value. The default value is *autoTensorMinValue*.

Return value

`None.`

Exceptions

RangeError.

Members

The SymbolOptions object can have the following members:

vectorQuantity A SymbolicConstant specifying the vector quantity to display. Possible values are RESULTANT and VECTOR_COMPONENT. The default value is RESULTANT.

vectorMaxValueAutoCompute A Boolean specifying whether the maximum vector value is to be computed automatically. The default value is ON.

vectorMaxValue

A Float specifying the user-specified maximum vector value. The default value is *autoVectorMaxValue*.

vectorMinValueAutoCompute

A Boolean specifying whether the minimum vector value is to be computed automatically. The default value is ON.

vectorMinValue

A Float specifying the user-specified minimum vector value. The default value is *autoVectorMinValue*.

tensorQuantity

A SymbolicConstant specifying the tensor quantity to display. Possible values are ALL_PRINCIPAL_COMPONENTS, PRINCIPAL_COMPONENT, ALL_DIRECT_COMPONENTS, and DIRECT_COMPONENT. The default value is ALL_PRINCIPAL_COMPONENTS.

tensorMaxValueAutoCompute

A Boolean specifying whether the maximum tensor value is to be computed automatically. The default value is ON.

tensorMaxValue

A Float specifying the user-specified maximum tensor value. The default value is *autoTensorMaxValue*.

tensorMinValueAutoCompute

A Boolean specifying whether the minimum tensor value is to be computed automatically. The default value is ON.

tensorMinValue

A Float specifying the user-specified minimum tensor value. The default value is *autoTensorMinValue*.

autoVectorMaxValue

The SymbolicConstant NOT_SET or a Float specifying the vector maximum value when *vectorMaxValueAutoCompute*=ON. This value is read-only. The default value is NOT_SET.

autoVectorMinValue

The SymbolicConstant NOT_SET or a Float specifying the vector minimum value when *vectorMinValueAutoCompute*=ON. This value is read-only. The default value is NOT_SET.

autoTensorMaxValue

The SymbolicConstant NOT_SET or a Float specifying the tensor maximum value when *tensorMaxValueAutoCompute*=ON. This value is read-only. The default value is NOT_SET.

autoTensorMinValue

The SymbolicConstant NOT_SET or a Float specifying the tensor minimum value when *tensorMinValueAutoCompute*=ON. This value is read-only. The default value is NOT_SET.

vectorLineThickness

A SymbolicConstant specifying the vector line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

vectorColorMethod

A SymbolicConstant specifying the vector color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

tensorColorMethod
A SymbolicConstant specifying the tensor color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

vectorArrowheadStyle
A SymbolicConstant specifying the vector arrowhead style. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

arrowSymbolSize
An Int specifying the length of vector and tensor symbols. The default value is 6.

vectorIntervalNumber
An Int specifying the number of color intervals for vector symbols. The default value is 12.

symbolDensity
A Float specifying the factor for randomized sampling. The default value is 1.0.

constantLengthArrows
A Boolean specifying whether to use constant-length arrows for vector symbols. The default value is OFF.

tensorIntervalNumber
An Int specifying the number of color intervals for tensor symbols. The default value is 12.

tensorLineThickness
A SymbolicConstant specifying the line thickness of the tensor symbols. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

tensorArrowheadStyle
A SymbolicConstant specifying the arrowhead style of the tensor symbols. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

numberFormatT
A SymbolicConstant specifying the number format for tensor. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

numberFormatV
A SymbolicConstant specifying the number format for vector. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

arrowScaleMode
A SymbolicConstant specifying the arrow scaling mode. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

drawLabelT
A Boolean specifying whether to draw tensor labels. The default value is OFF.

drawLabelV
A Boolean specifying whether to draw vector labels. The default value is OFF.

numDigitsT
An Int specifying the number of digits in the tensor label. The default value is 2.

numDigitsV
An Int specifying the number of digits in the vector label. The default value is 2.

vectorColor
A String specifying the vector color. The default value is "Red".

vectorColorSpectrum
A String specifying the vector color spectrum name. The default value is "Rainbow".

tensorColorSpectrum
A String specifying the tensor color spectrum name. The default value is "Rainbow".

textColorT
A String specifying the text color for tensor. The default value is "Yellow".

textColorV
A String specifying the text color for vector. The default value is "Yellow".

textFontT
A String specifying the text font for tensor. The default value is "verdana".

textFontV
A String specifying the text font for vector. The default value is "verdana".

tensorMaxPrinColor
A String specifying the color of the maximum principal tensor symbols. The default value is "Red".

tensorMinPrinColor
A String specifying the color of the minimum principal tensor symbols. The default value is "Cyan".

tensorMidPrinColor
A String specifying the color of the intermediate principal tensor symbols. The default value is "Yellow".

tensorSelectedPrinColor
A String specifying the color of the selected principal tensor symbols. The default value is "Red".

ViewCut object

The ViewCut object is used to store values and attributes associate with ViewCut type objects. ViewCut objects can be created using the methods described below. The methods accessed via the *OdbDisplay* object cause the ViewCut object to be added to the session.viewports[name].odbDisplay.viewCuts repository.

Access

```
import visualization
session.viewports[name].layers[name].odbDisplay.viewCuts[name]
session.viewports[name].odbDisplay.viewCuts[name]
```

ViewCut(...)

This method creates a ViewCut object.

Path

```
session.viewports[name].layers[name].odbDisplay.ViewCut
session.viewports[name].odbDisplay.ViewCut
```

Required arguments

name
A String specifying the repository key.

shape
A SymbolicConstant specifying the shape of the ViewCut object. Possible values are PLANE, CYLINDER, SPHERE, and ISOSURFACE.

origin
A sequence of three Floats specifying the X-, Y-, and Z-coordinates of the origin of the plane, cylinder or sphere cut. This origin is not required if the cut shape is ISOSURFACE or if the cut is defined by the *csysName* argument.

normal
A sequence of Floats specifying the X-, Y-, and Z-coordinates of the normal axis to the plane defining the cut, when the plane is defined using the *origin* argument or a SymbolicConstant defining this normal axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is not required if the cut *shape* is CYLINDER, SPHERE or ISOSURFACE.

axis2
A sequence of three Floats specifying the X-, Y-, and Z-coordinates of the second axis of the plane defining the cut, when the plane is defined using the *origin* argument or a SymbolicConstant defining this second axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is used to rotate the plane cut. It is not required if the cut *shape* is CYLINDER, SPHERE or ISOSURFACE.

csysName
A String specifying the name of the *DatumCsys* object to be used to define the cut. This name is not required if the cut *shape* is ISOSURFACE or if the cut is defined by the *origin* argument.

cylinderAxis
A sequence of Floats specifying the X-, Y-, and Z-coordinates of the cylinder axis defining the cut, when the cut is defined using the *origin* argument or a SymbolicConstant defining this cylinder axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is not required if the cut *shape* is PLANE, SPHERE, or ISOSURFACE.

Optional arguments

followDeformation A Boolean specifying whether the cut will follow the deformation or be static. The default value is OFF.

overrideAveraging A Boolean specifying averaging for element based fields associated with isosurface cuts will be set to compute-average with a threshold of 100% when true. The current field options will be used when false. The default value is ON.

referenceFrame A SymbolicConstant specifying which reference frame will be used when the cut follows the deformation. Possible values are FIRST_FRAME, LAST_FRAME, and CURRENT_FRAME. The default value is FIRST_FRAME.

Return value

A ViewCut object.

Exceptions

None.

`setValues(...)`

This method modifies the ViewCut object.

Required arguments

None.

Optional arguments

At least one of the following must be provided:

angle A Float specifying the rotation angle of the cut defined with a *shape* set to PLANE.

motion A SymbolicConstant specifying the type of motion for the cut defined with a *shape* set to PLANE. Possible values are TRANSLATE and ROTATE. The default value is TRANSLATE.

position A Float specifying the position of the cut defined with a *shape* set to PLANE.

radius A Float specifying the radius of the cut defined with a *shape* set to CYLINDER or SPHERE.

rotationAxis A SymbolicConstant specifying the rotation axis for the cut defined with a *shape* set to PLANE. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_2.

value A Float specifying the value of the cut defined with a *shape* set to ISOSURFACE.

showModelAboveCut A Boolean specifying whether to display the model above the cut. The default value is OFF.

showModelOnCut A Boolean specifying whether to display the model on the cut. The default value is ON.

showModelBelowCut A Boolean specifying whether to display the model below the cut. The default value is ON.

showFreeBodyCut A Boolean specifying whether to display the free body cut. The default value is OFF.

csysName A String specifying the name of the *DatumCsys* object to be used to define the cut. This name is not required if the cut *shape* is ISOSURFACE or if the cut is defined by the *origin* argument.

origin A sequence of three Floats specifying the X-, Y-, and Z-coordinates of the origin of the plane, cylinder or sphere cut. This origin is not required if the cut shape is ISOSURFACE or if the cut is defined by the *csysName* argument.

normal A sequence of Floats specifying the X-, Y-, and Z-coordinates of the normal axis to the plane defining the cut, when the plane is defined using the *origin* argument or a SymbolicConstant defining this normal axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is not required if the cut *shape* is CYLINDER, SPHERE or ISOSURFACE.

axis2 A sequence of three Floats specifying the X-, Y-, and Z-coordinates of the second axis of the plane defining the cut, when the plane is defined using the *origin* argument or a SymbolicConstant defining this second axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is used to rotate the plane cut. It is not required if the cut *shape* is CYLINDER, SPHERE or ISOSURFACE.

cylinderAxis A sequence of Floats specifying the X-, Y-, and Z-coordinates of the cylinder axis defining the cut, when the cut is defined using the *origin* argument or a SymbolicConstant defining this cylinder axis, when the cut is defined by the *csysName* argument. Possible values are AXIS_1, AXIS_2, AXIS_3. This axis is not required if the cut *shape* is PLANE, SPHERE, or ISOSURFACE.

followDeformation A Boolean specifying whether the cut will follow the deformation or be static. The default value is OFF.

overrideAveraging A Boolean specifying averaging for element based fields associated with isosurface cuts will be set to compute-average with a threshold of 100% when true. The current field options will be used when false. The default value is ON.

referenceFrame A SymbolicConstant specifying which reference frame will be used when the cut follows the deformation. Possible values are FIRST_FRAME, LAST_FRAME, and CURRENT_FRAME. The default value is FIRST_FRAME.

Return value

`None.`

Exceptions

None.

updateVariable()

This method updates the field associated with an isosurface cut to be consistent with the current primary variable.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The ViewCut object has members with the same names and descriptions as the arguments to the *ViewCut* method.

In addition, the ViewCut object has the following members:

angle A Float specifying the rotation angle of the cut defined with a *shape* set to PLANE.

motion A SymbolicConstant specifying the type of motion for the cut defined with a *shape* set to PLANE. Possible values are TRANSLATE and ROTATE. The default value is TRANSLATE.

position A Float specifying the position of the cut defined with a *shape* set to PLANE.

radius A Float specifying the radius of the cut defined with a *shape* set to CYLINDER or SPHERE.

rotationAxis A SymbolicConstant specifying the rotation axis for the cut defined with a *shape* set to PLANE. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_2.

value A Float specifying the value of the cut defined with a *shape* set to ISOSURFACE.

showModelAboveCut A Boolean specifying whether to display the model above the cut. The default value is OFF.

showModelOnCut A Boolean specifying whether to display the model on the cut. The default value is ON.

showModelBelowCut A Boolean specifying whether to display the model below the cut. The default value is ON.

showFreeBodyCut A Boolean specifying whether to display the free body cut. The default value is OFF.

active A Boolean specifying whether the cut is displayed.

cutRange

A pair of Floats specifying the acceptable range for positioning the cut.

crossSectionalArea

A Float returning the cross-sectional area of the cut when *showFreeBodyCut* is set to ON.

ViewerOptions object

The ViewerOptions object specifies options to set the result caching parameters. The ViewerOptions object has no constructor. Abaqus creates the *viewerOptions* member when a session is started.

Access

```
import visualization
session.viewerOptions
```

`setValues(...)`

This method modifies the ViewerOptions object.

Required arguments

None.

Optional arguments

primaryVariableCaching
A Boolean specifying whether results are currently cached. Caching improves the performance of subsequent access. The default value is ON.

deformedVariableCaching
A Boolean specifying whether deformation vectors are currently cached. Caching improves the performance of subsequent access. The default value is ON.

cutVariableCaching
A Boolean specifying whether the values used for displaying cut models are currently cached. Caching improves the performance of subsequent access. The default value is ON.

odbUpdateChecking
A Boolean specifying whether the current `.odb` file should be checked for updates. Setting *odbUpdateChecking* to OFF can improve Viewer performance when accessing data from a remote file. The default value is ON.

odbUpdateCheckInterval
An Int specifying the minimum time between status checks (in seconds). The default value is 0.

Return value

`None.`

Exceptions

None.

Members

The ViewerOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

Optimization commands

Optimization commands are used to perform topology, shape, or sizing optimization of your model given a set of objectives and a set of restrictions.

In this section:

- *OptimizationTask object*
- *BeadTask object*
- *CombinedTermDesignResponse object*
- *DesignDirection object*
- *DesignResponse object*
- *DrillControl object*
- *FixedRegion object*
- *FrozenArea object*
- *GeometricRestriction object*
- *Growth object*
- *LocalStopCondition object*
- *ObjectiveFunction object*
- *OptimizationConstraint object*
- *OptimizationObjective object*
- *PenetrationCheck object*
- *ShapeDemoldControl object*
- *ShapeMemberSize object*
- *ShapePlanarSymmetry object*
- *ShapePointSymmetry object*
- *ShapeRotationalSymmetry object*
- *ShapeTask object*
- *SingleTermDesignResponse object*
- *SizingClusterAreas object*
- *SizingCyclicSymmetry object*
- *SizingFrozenArea object*
- *SizingMemberSize object*
- *SizingPlanarSymmetry object*
- *SizingPointSymmetry object*
- *SizingRotationalSymmetry object*
- *SizingTask object*
- *SlideRegionControl object*
- *StampControl object*
- *StepOption object*
- *StopCondition object*
- *TopologyCyclicSymmetry object*
- *TopologyDemoldControl object*
- *TopologyMemberSize object*

- *TopologyOverhangControl object*
- *TopologyPlanarSymmetry object*
- *TopologyPointSymmetry object*
- *TopologyRibDesign object*
- *TopologyRotationalSymmetry object*
- *TopologyMillingControl object*
- *TopologyTask object*
- *TurnControl object*
- *BeadFilter object*
- *BeadFixedRegion object*
- *BeadGrowth object*
- *BeadPenetrationCheck object*
- *BeadPlanarSymmetry object*
- *BeadPointSymmetry object*
- *BeadRotationalSymmetry object*

OptimizationTask object

The OptimizationTask object is the abstract base type for other OptimizationTask objects. The OptimizationTask object has no explicit constructor. The methods and members of the OptimizationTask object are common to all objects derived from OptimizationTask.

Access

```
import optimization
mdb.models[name].optimizationTasks[name]
```

Members

The OptimizationTask object can have the following members:

name A String specifying the optimization task repository key.

region The SymbolicConstant MODEL or a *Region* object specifying the region to which the optimization task is applied. The default value is MODEL.

designResponses A repository of *DesignResponse* objects.

objectiveFunctions A repository of *ObjectiveFunction* objects.

optimizationConstraints A repository of *OptimizationConstraint* objects.

geometricRestrictions A repository of *GeometricRestriction* objects.

stopConditions A repository of *StopCondition* objects.

BeadTask object

The BeadTask object defines a bead task.

The BeadTask object is derived from the *OptimizationTask* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name]
```

BeadTask(...)

This method creates a BeadTask object.

Path

```
mdb.models[name].BeadTask
```

Required arguments

name A String specifying the optimization task repository key.

Optional arguments

abaqusSensitivities A Boolean specifying whether to use Abaqus to compute the design responses and their sensitivities. The default value is True.

algorithm A SymbolicConstant specifying the optimization task algorithm. Possible values are GENERAL_OPTIMIZATION and CONDITION_BASED_OPTIMIZATION. The default value is GENERAL_OPTIMIZATION.

areBCRegionsFrozen A Boolean specifying whether to exclude elements with boundary conditions from the optimization. The default value is OFF.

beadIter An int specifying the step size of the optimization. The default value is 1.

beadMaxMembraneStress A float specifying maximum membrane/bending stress. The default value is 0.1.

beadMinStress A float specifying minimum stress. The default value is 0.001.

beadPerturbation A Sets perturbation size for finite differences. The default value is 0.0001.

beadWidth A SymbolicConstant specifying the Optimization product default or a float specifying the bead width. The default value is DEFAULT.

curveSmooth
A float specifying relative value to the middle element edge length such that normals in this area do not cross each other. The default value is 5.

filterRadius
A float specifying the filter radius. The default value is 4.

filterRadiusBy
A SymbolicConstant specifying the method used to define filter radius. Possible values are VALUE and REFERENCE. The default is VALUE.

flipNormalDir
A Boolean specifying whether the growth direction is along the normal direction of elements or opposite to the normal direction. The default value is OFF

frozenBoundaryConditionRegion
When nodes with boundary conditions are excluded from the optimization (*frozenBoundaryConditionRegions* = ON). you can specify that this exclusion apply to nodes throughout the model or only to those nodes from a specific region. Set this parameter to the SymbolicConstant MODEL to apply the freeze to the entire model, or set this parameter to a Region object to specify an individual region over which nodes with boundary conditions should be frozen. The default value is MODEL.

isSensCalcOnlyOnDesignNodes
A Boolean specifying whether to calculate the sensitivities only on design nodes or the whole model. The default value is ON

modeTrackingRegion
The SymbolicConstant MODEL or a *Region* object specifying the region to use for mode tracking. The default value is MODEL.

nodalMoveLimit
A Float specifying the maximum change in nodal displacement per design cycle. The default value is 0.1.

nodeSmooth
A SymbolicConstant specifying the Optimization product default or a float specifying the node smooth. The default value is DEFAULT.

nodeUpdateStrategy
A SymbolicConstant specifying the strategy for how the nodal displacements are updated in the method of moving asymptotes. Possible values are NORMAL, CONSERVATIVE, and AGGRESSIVE. The default value is CONSERVATIVE.

numTrackedModes
An Int specifying the number of modes included in mode tracking. The default value is 5.

updateShapeBasisVectors
A SymbolicConstant specifying whether to update shape basis vectors in the first design cycle or every design cycle. Possible values are EVERY_CYCLE and FIRST_CYCLE. The default value is EVERY_CYCLE.

groupOperator
A Boolean specifying whether the group in the design response will be evaluated using the existing algorithm or a new algorithm based on Abaqus sensitivities. The default value of False means that the existing algorithm will be used.

Return value

A BeadTask object.

Exceptions

None.

setValues(...)

This method modifies the BeadTask object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadTask* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadTask object has members with the same names and descriptions as the arguments to the *BeadTask* method.

In addition, the BeadTask object can have the following members:

designResponses	A repository of *DesignResponse* objects.
objectiveFunctions	A repository of *ObjectiveFunction* objects.
optimizationConstraints	A repository of *OptimizationConstraint* objects.
geometricRestrictions	A repository of *GeometricRestriction* objects.

CombinedTermDesignResponse object

The CombinedTermDesignResponse object defines a combined-term design response.

The CombinedTermDesignResponse object is derived from the *DesignResponse* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].designResponses[name]
```

CombinedTermDesignResponse(...)

This method creates a CombinedTermDesignResponse object.

Path

```
mdb.models[name].optimizationTasks[name].CombinedTermDesignResponse
```

Required arguments

name A String specifying the design response repository key.

terms A sequence of Strings specifying the names of the design responses to combine.

Optional arguments

filterMaxRadius `None` or a sequence of Floats specifying the maximum radius of influence used when *method* is FILTER. The default value is `None`.

filterExponent A Float specifying the exponent used when *method* is FILTER. The default value is 1.0.

filterRadiusReduction A Float specifying the reduction of the radius depending on surface bending, used when *method* is FILTER. The default value is 0.2.

highCutOff `None` or a sequence of Floats specifying the upper bound of the vector value used when *method* is CUT_OFF. All values greater than the *highCutOff* are set to the *highCutOff* value. The default value is `None`.

lowCutOff A Float specifying the lower bound of the vector value used when *method* is CUT_OFF. All values less than the *lowCutOff* are treated as 0. The default value is 0.0.

method A SymbolicConstant specifying the method used to combine selected design responses. Possible values are:

- ABSOLUTE_DIFFERENCE
- ABSOLUTE_VALUE

- ADD
- COSINE
- CUT_OFF
- DELTA_OVER_1_ITERATION
- DELTA_OVER_2_ITERATIONS
- DELTA_OVER_3_ITERATIONS
- DELTA_OVER_4_ITERATIONS
- DELTA_OVER_5_ITERATIONS
- DELTA_OVER_6_ITERATIONS
- DIVIDE
- EXPONENTIAL
- FILTER
- INTEGER
- LOG
- MAXIMUM
- MINIMUM
- MULTIPLY
- NATURAL_LOG
- NEAREST_INTEGER
- NORM
- NORM_FIRST
- NTH_POWER
- NTH_ROOT
- SIGN
- SINE
- SQUARE_ROOT
- SUBTRACT
- TANGENT
- WEIGHTED_ADD

The default value is ADD.

weights

A sequence of Floats specifying the weights to apply to the list of design responses used when *method* is WEIGHTED_ADD. The default value is an empty sequence.

Return value

A CombinedTermDesignResponse object.

Exceptions

None.

`setValues(...)`

This method modifies the CombinedTermDesignResponse object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CombinedTermDesignResponse* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The CombinedTermDesignResponse object has members with the same names and descriptions as the arguments to the *CombinedTermDesignResponse* method.

DesignDirection object

The DesignDirection object defines a design direction geometric restriction.

The DesignDirection object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

DesignDirection(...)

This method creates a DesignDirection object.

Path

```
mdb.models[name].optimizationTasks[name].DesignDirection
```

Required arguments

name
A String specifying the geometric restriction repository key.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

mainPoint
`None` or a *Region* object specifying the main point used when *mainPointDetermination* is SPECIFY. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for assigning point priority. Possible values are MAXIMUM, MINIMUM, and SPECIFY. The default value is MAXIMUM.

movementRestriction
A SymbolicConstant specifying whether movement in the region should follow only the direction of the *mainPoint*, only the magnitude, or both the magnitude of the *mainPoint* and the directions specified by *u1*, *u2* and *u3*. Possible values are DIRECTION, MAGNITUDE, and VECTOR. The default value is VECTOR.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

u1

A Boolean specifying whether movement in the region should follow the *masterPoint* in the 1-direction. This is used when *movementRestriction* is VECTOR. The default value is ON.

u2

A Boolean specifying whether movement in the region should follow the *masterPoint* in the 2-direction. This is used when *movementRestriction* is VECTOR. The default value is ON.

u3

A Boolean specifying whether movement in the region should follow the *masterPoint* in the 3-direction. This is used when *movementRestriction* is VECTOR. The default value is ON.

Return value

A DesignDirection object.

Exceptions

None.

`setValues(...)`

This method modifies the DesignDirection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DesignDirection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DesignDirection object has members with the same names and descriptions as the arguments to the *DesignDirection* method.

DesignResponse object

The DesignResponse object is the abstract base type for other DesignResponse objects. The DesponseResponse object has no explicit constructor. The methods and members of the DesignResponse object are common to all objects derived from DesignResponse.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].designResponses[name]
```

Members

The DesignResponse object has the following member:

name
A String specifying the design response repository key.

DrillControl object

The DrillControl object defines a drill control geometric restriction.

The DrillControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

DrillControl(...)

This method creates a DrillControl object.

Path

```
mdb.models[name].optimizationTasks[name].DrillControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the direction of the drill axis positioned at the *csys* origin. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

drawAngle
A Float specifying the draw angle. The default value is 0.0.

mainPoint
`None` or a *Region* object specifying the main point used when *mainPointDetermination* is SPECIFY. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for assigning point priority. Possible values are MAXIMUM, MINIMUM, and SPECIFY. The default value is MAXIMUM.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

tolerance1	A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.
tolerance2	A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.
tolerance3	A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.
undercutTolerance	A Float specifying the undercut tolerance. The default value is 0.0.

Return value

A DrillControl object.

Exceptions

None.

`setValues(...)`

This method modifies the DrillControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DrillControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The DrillControl object has members with the same names and descriptions as the arguments to the *DrillControl* method.

FixedRegion object

The FixedRegion object defines a fixed region geometric restriction.

The FixedRegion object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

FixedRegion(...)

This method creates a FixedRegion object.

Path

```
mdb.models[name].optimizationTasks[name].FixedRegion
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

presumeFeasibleRegionAtStart A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

u1 A Boolean specifying whether to fix the region in the 1-direction. The default value is OFF.

u2 A Boolean specifying whether to fix the region in the 2-direction. The default value is OFF.

u3 A Boolean specifying whether to fix the region in the 3-direction. The default value is OFF.

Return value

A FixedRegion object.

Exceptions

None.

setValues(...)

This method modifies the FixedRegion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FixedRegion* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FixedRegion object has members with the same names and descriptions as the arguments to the *FixedRegion* method.

FrozenArea object

The FrozenArea object defines a frozen area geometric restriction.

The FrozenArea object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

FrozenArea(...)

This method creates a FrozenArea object.

Path

```
mdb.models[name].optimizationTasks[name].FrozenArea
```

Required arguments

name A String specifying the geometric restriction repository key.

Optional arguments

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Return value

A FrozenArea object.

Exceptions

None.

`setValues(...)`

This method modifies the FrozenArea object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FrozenArea* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FrozenArea object has members with the same names and descriptions as the arguments to the *FrozenArea* method.

GeometricRestriction object

The GeometricRestriction object is the abstract base type for other GeometricRestriction objects. The GeometricRestriction object has no explicit constructor. The methods and members of the GeometricRestriction object are common to all objects derived from GeometricRestriction.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

Members

The GeometricRestriction object can have the following members:

name

A String specifying the geometric restriction repository key.

region

A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Growth object

The Growth object defines a growth geometric restriction.

The Growth object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

Growth(...)

This method creates a Growth object.

Path

```
mdb.models[name].optimizationTasks[name].Growth
```

Required arguments

name
A String specifying the geometric restriction repository key.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

growth
A Float specifying the maximum optimization displacement in the growth direction. Either *growth* or *shrink* or both must be specified. The default value is 0.0.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

shrink
A Float specifying the maximum optimization displacement in the shrink direction. Either *growth* or *shrink* or both must be specified The default value is 0.0.

Return value

A Growth object.

Exceptions

None.

setValues(...)

This method modifies the Growth object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Growth* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Growth object has members with the same names and descriptions as the arguments to the *Growth* method.

LocalStopCondition object

The LocalStopCondition object defines a local stop condition.

The LocalStopCondition object is derived from the *StopCondition* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].stopConditions[name]
```

LocalStopCondition(...)

This method creates a LocalStopCondition object.

Path

```
mdb.models[name].optimizationTasks[name].LocalStopCondition
```

Required arguments

name A String specifying the stop condition repository key.

referenceFactor A Float specifying the factor used to modify the reference value.

Optional arguments

comparisonOperation A SymbolicConstant specifying the operation used to compare the selected value to the reference value. Possible values are LESS_THAN, LESS_THAN_EQUAL, EQUAL, GREATER_THAN_EQUAL, and GREATER_THAN. The default value is LESS_THAN.

identifier A SymbolicConstant specifying the variable identifier of the compared value. Possible values are:

- ABSOLUTE_GROWTH_MOVEMENT
- ABSOLUTE_SHRINK_MOVEMENT
- GROWTH_MOVEMENT
- SHRINK_MOVEMENT
- MOVEMENT
- TOTAL_ABSOLUTE_MOVEMENT
- EQUIV_STRESS
- FREE_TASK_REGION_EQUIV_STRESS
- RESTRICTED_TASK_REGION_EQUIV_STRESS
- SURFACE_POINT_EQUIV_STRESS
- TASK_REGION_EQUIV_STRESS

The default value is MOVEMENT.

identifierOperation
A SymbolicConstant specifying the operation used to evaluate values in the region. Possible values are MAXIMUM, MINIMUM, and SUM. The default value is MAXIMUM.

referenceDesignCycle
A SymbolicConstant specifying the iteration from which a value is compared to the reference value. Possible values are FIRST and PREVIOUS. The default value is PREVIOUS.

referenceOperation
A SymbolicConstant specifying the operation used to modify the reference value by the reference factor. Possible values are ADD, DIVIDE, MULTIPLY, and SUBTRACT. The default value is ADD.

region
The SymbolicConstant MODEL or a *Region* object specifying the region to which the stop condition is applied. The default value is MODEL.

Return value

A LocalStopCondition object.

Exceptions

None.

`setValues(...)`

This method modifies the LocalStopCondition object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *LocalStopCondition* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The LocalStopCondition object has members with the same names and descriptions as the arguments to the *LocalStopCondition* method.

ObjectiveFunction object

The ObjectiveFunction object defines the objective of the optimization.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].objectiveFunctions[name]
```

ObjectiveFunction(...)

This method creates an ObjectiveFunction object.

Path

```
mdb.models[name].optimizationTasks[name].ObjectiveFunction
```

Required arguments

name A String specifying the objective function repository key.

objectives An *OptimizationObjectiveArray* object.

Optional arguments

target A SymbolicConstant specifying the target of the objective function. Possible values are MINIMIZE, MAXIMIZE, and MINIMIZE_MAXIMUM. The default value is MINIMIZE.

Return value

An ObjectiveFunction object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the ObjectiveFunction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ObjectiveFunction* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ObjectiveFunction object has members with the same names and descriptions as the arguments to the *ObjectiveFunction* method.

OptimizationConstraint object

The OptimizationConstraint object constrains an optimization from making changes to the topology of the model.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].optimizationConstraints[name]
```

OptimizationConstraint(...)

This method creates an OptimizationConstraint object.

Path

```
mdb.models[name].optimizationTasks[name].OptimizationConstraint
```

Required arguments

name A String specifying the optimization constraint repository key.

designResponse A String specifying the name of the design response to constrain.

restrictionValue A Float specifying the value to which the design response should be constrained.

Optional arguments

restrictionMethod A SymbolicConstant specifying the method used to constrain the design response. Possible values are ABSOLUTE_EQUAL, ABSOLUTE_GREATER_THAN_EQUAL, ABSOLUTE_LESS_THAN_EQUAL, RELATIVE_EQUAL, RELATIVE_GREATER_THAN_EQUAL, and RELATIVE_LESS_THAN_EQUAL. The default value is ABSOLUTE_EQUAL.

Return value

An OptimizationConstraint object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the OptimizationConstraint object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *OptimizationConstraint* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The OptimizationConstraint object has members with the same names and descriptions as the arguments to the *OptimizationConstraint* method.

OptimizationObjective object

An OptimizationObjective is an object used to define objectives in an objective function.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].objectiveFunctions[name]\
.objectives[i]
```

Members

The OptimizationObjective object has the following members:

suppress
A Boolean specifying whether the objective is suppressed or not. The default value is OFF.

weight
A Float specifying the weight applied to the design response value. The default value is 1.0.

referenceValue
The SymbolicConstant DEFAULT or a Float specifying the reference value used in evaluating a design response. For topology optimization, DEFAULT> indicates the reference value is 0. For shape optimization, DEFAULT indicates the reference value is the nodal average. The default value is DEFAULT.

designResponse
A String specifying the name of the design response.

PenetrationCheck object

The PenetrationCheck object defines a penetration check geometric restriction.

The PenetrationCheck object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

PenetrationCheck(...)

This method creates a PenetrationCheck object.

Path

```
mdb.models[name].optimizationTasks[name].PenetrationCheck
```

Required arguments

name	A String specifying the geometric restriction repository key.
penetrationCheckRegion	A *Region* object specifying the penetration check region.
region	A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

presumeFeasibleRegionAtStart	A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

Return value

A PenetrationCheck object.

Exceptions

None.

setValues(...)

This method modifies the PenetrationCheck object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PenetrationCheck* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The PenetrationCheck object has members with the same names and descriptions as the arguments to the *PenetrationCheck* method.

ShapeDemoldControl object

The ShapeDemoldControl object defines a shape demold control geometric restriction.

The ShapeDemoldControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

ShapeDemoldControl(...)

This method creates a ShapeDemoldControl object.

Path

```
mdb.models[name].optimizationTasks[name].ShapeDemoldControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

pullDirection
A *VertexArray* object of length 2 specifying the demold pull direction. Instead of through a *Vertex*, each point might be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

collisionCheckRegion
The SymbolicConstant DEMOLD_REGION or a *Region* object specifying the collision check region. If the value is DEMOLD_REGION, then the value of *region* is used as both the demold region and the collision check region. The default value is DEMOLD_REGION.

csys
`None` or a *DatumCsys* object specifying the local coordinate system of the *pullDirection*. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int indicating the identifier of the *DatumCsys*. The default value is `None`.

drawAngle
A Float specifying the draw angle. The default value is 0.0.

mainPointDetermination
A SymbolicConstant specifying the rule for assigning point priority. Possible values are MAXIMUM and MINIMUM. The default value is MAXIMUM.

presumeFeasibleRegionAtStart	A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.
tolerance1	A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.
tolerance2	A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.
tolerance3	A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.
undercutTolerance	A Float specifying the undercut tolerance. The default value is 0.0.

Return value

A ShapeDemoldControl object.

Exceptions

None.

`setValues(...)`

This method modifies the ShapeDemoldControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapeDemoldControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapeDemoldControl object has members with the same names and descriptions as the arguments to the *ShapeDemoldControl* method.

ShapeMemberSize object

The ShapeMemberSize object defines a shape member size geometric restriction.

The ShapeMemberSize object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

ShapeMemberSize(...)

This method creates a ShapeMemberSize object.

Path

```
mdb.models[name].optimizationTasks[name].ShapeMemberSize
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

maxThickness A Float specifying the maximum thickness. The default value is 0.0.

minThickness A Float specifying the minimum thickness. The default value is 0.0.

sizeRestriction A SymbolicConstant specifying whether to restrict the minimum or maximum thickness. Possible values are MAXIMUM and MINIMUM. The default value is MINIMUM.

assignNodeGroupRegion A bool specifying whether to use the node group region. The default value is OFF.

nodeGroupRegion A *Node Region* object specifying the check node group.

Return value

A ShapeMemberSize object.

Exceptions

None.

setValues(...)

This method modifies the ShapeMemberSize object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapeMemberSize* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapeMemberSize object has members with the same names and descriptions as the arguments to the *ShapeMemberSize* method.

ShapePlanarSymmetry object

The ShapePlanarSymmetry object defines a shape planar symmetry geometric restriction.

The ShapePlanarSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

ShapePlanarSymmetry(...)

This method creates a ShapePlanarSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].ShapePlanarSymmetry
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the vector positioned at the *csys* origin that is normal to the symmetry plane. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

allowNonSymmetricMesh
A Boolean specifying whether to allow a nonsymmetric mesh for this geometric restriction. The default value is TRUE.

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for determining the main node. Possible values are MAXIMUM and MINIMUM. The default value is MAXIMUM.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

tolerance1
A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.

tolerance2
A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.

tolerance3
A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.

Return value

A ShapePlanarSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the ShapePlanarSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapePlanarSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapePlanarSymmetry object has members with the same names and descriptions as the arguments to the *ShapePlanarSymmetry* method.

ShapePointSymmetry object

The ShapePointSymmetry object defines a shape point symmetry geometric restriction.

The ShapePointSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

ShapePointSymmetry(...)

This method creates a ShapePointSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].ShapePointSymmetry
```

Required arguments

name
A String specifying the geometric restriction repository key.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the symmetry point represented as the origin of a local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for determining the main node. Possible values are MAXIMUM and MINIMUM. The default value is MAXIMUM.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

tolerance1
A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.

tolerance2
A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.

tolerance3
A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.

Return value

A ShapePointSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the ShapePointSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapePointSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapePointSymmetry object has members with the same names and descriptions as the arguments to the *ShapePointSymmetry* method.

ShapeRotationalSymmetry object

The ShapeRotationalSymmetry object defines a shape rotational symmetry geometric restriction.

The ShapeRotationalSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

ShapeRotationalSymmetry(...)

This method creates a ShapeRotationalSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].ShapeRotationalSymmetry
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the vector positioned at the *csys* origin, used as the axis of symmetry. Instead of through a *Vertex*, each point might be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

allowNonSymmetricMesh
A Boolean specifying whether to allow a nonsymmetric mesh for this geometric restriction. The default value is TRUE.

angle
A Float specifying the segment size of the repeating pattern in degrees. If the *angle* value is 0, no repeating pattern is created. The default value is 0.0.

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

mainPoint
`None` or a *Region* object specifying the main point used when *mainPointDetermination* is SPECIFY. The default value is `None`.

mainPointDetermination	A SymbolicConstant specifying the rule for determining the main node. Possible values are MAXIMUM, MINIMUM, and SPECIFY. The default value is MAXIMUM.
presumeFeasibleRegionAtStart	A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.
startPoint	A tuple of Floats representing the coordinates of a start point of the rotational symmetry.
tolerance1	A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.
tolerance2	A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.
tolerance3	A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.

Return value

A ShapeRotationalSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the ShapeRotationalSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapeRotationalSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapeRotationalSymmetry object has members with the same names and descriptions as the arguments to the *ShapeRotationalSymmetry* method.

ShapeTask object

The ShapeTask object defines a shape task.

The ShapeTask object is derived from the *OptimizationTask* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name]
```

ShapeTask(...)

This method creates a ShapeTask object.

Path

```
mdb.models[name].ShapeTask
```

Required arguments

name A String specifying the optimization task repository key.

Optional arguments

abaqusSensitivities A Boolean specifying whether to use Abaqus to compute the design responses and their sensitivities. The default value is True.

absoluteStepSizeControl A SymbolicConstant specifying whether to control the permitted absolute step size by the average optimization displacement or minimum optimization displacement. Possible values are MINIMUM and AVERAGE. The default value is MINIMUM.

activateDurability A boolean specifying whether or not the durability approach of optimization is turned on. The default value is ON.

additionalDurabilityFiles A String specifying the path of additional files pertaining to durability optimization. Only valid if the *activateDurability* argument is ON.

algorithm A SymbolicConstant specifying the optimization task algorithm. Possible values are GENERAL_OPTIMIZATION and CONDITION_BASED_OPTIMIZATION. The default value is CONDITION_BASED_OPTIMIZATION.

constrainedLaplacianConvergenceLevel A SymbolicConstant specifying the constrained Laplacian convergence level. Possible values are NORMAL, CONSERVATIVE, and AGGRESSIVE. The default value is NORMAL.

curvatureSmoothingEdgeLength

A Float specifying the edge length for the movement vector. The default value is 5.0.

durabilityInputfile

A string specifying the path of the input file. Only valid if the *activateDurability* argument is ON and is a required argument in that case.

durabilitySolver

A String specifying the type of solver for durability optimization. Possible values are: FE_SAFE, FEMFAT, and CUSTOM. The default value is FE_SAFE. Only valid if the *activateDurability* argument is ON.

equalityConstraintTolerance

A Float specifying the equality constraint tolerance. The default value is 10^{-3}.

featureRecognitionAngle

A Float specifying the mesh smoothing feature recognition angle for edges and corners. The default value is 30.0.

filterExponent

A Float specifying the weight depending on the radius, used when *filterMaxRadius* is specified. The default value is 1.0.

filterMaxRadius

`None` or a Float specifying the maximum influence radius for equivalent stress. The default value is `None`.

filterRadiusReduction

`None` or a Float specifying the reduction of the radius depending on surface bending, used when *filterMaxRadius* is specified. The default value is `None`.

firstCycleDeletedVolumeTechnique

A SymbolicConstant specifying the method of specifying volume that can be removed immediately in the first design cycle. Possible values are OFF, PERCENTAGE, and ABSOLUTE. The default value is OFF.

freezeBoundaryConditionRegions

A Boolean specifying whether to exclude nodes with boundary conditions from the optimization. The default value is OFF.

frozenBoundaryConditionRegion

The SymbolicConstant MODEL or a *Region* object specifying the region in which to freeze boundary condition regions, or the SymbolicConstant MODEL, used with *freezeBoundaryConditionRegions*. The default value is MODEL.

geometricRestrictionEvaluationFrequency

A SymbolicConstant specifying the frequency of evaluating geometric restrictions during mesh smoothing. Possible values are LOW, MEDIUM, and HIGH. The default value is LOW.

growthScaleFactor

A Float specifying the scale factor to apply to optimization displacements for nodes with growth. The default value is 1.0.

haltUponViolation

A Boolean specifying whether to halt the optimization if quality criteria are not satisified. The default value is OFF.

layerReferenceRegion

`None` or a *Region* object specifying the region specifying the first node layer for mesh smoothing, used when *meshSmoothingRegionMethod* is TASK_REGION_LAYERS. The default value is `None`.

meshSmoothingRegionMethod

A SymbolicConstant specifying the method used to determine the mesh smoothing region. The REGION value uses the *smoothingRegion*. The NUMBER_OF_LAYERS value uses the *layerReferenceRegion*. The TASK_REGION_LAYERS value will smooth six layers using the task region. Possible values are TASK_REGION_LAYERS, REGION, and NUMBER_OF_LAYERS. The default value is TASK_REGION_LAYERS.

meshSmoothingStrategy

A SymbolicConstant specifying the method smoothing strategy. Possible values are CONSTRAINED_LAPLACIAN and LOCAL_GRADIENT. The default value is CONSTRAINED_LAPLACIAN.

midsideInterpolation

A SymbolicConstant specifying the approach used when treating midside node positions during optimization. POSITIONS indicates midside node positions are interpolated linearly by position. OPTIMIZATION_DISPLACEMENT indicates they are interpolated by optimization displacement of corner nodes. Possible values are POSITIONS and OPTIMIZATION_DISPLACEMENT. The default value is POSITIONS.

numFreeNodeLayers

The SymbolicConstant FIX_NONE or an Int specifying the number of node layers adjoining the task region to remain free during mesh smoothing. A value of 0 indicates that no layers are free and all layers are fixed. The default value is 0.

numSmoothedElementLayers

`None` or an Int specifying the number of layers for mesh smoothing when *meshSmoothingRegionMethod* is NUMBER_OF_LAYERS. The default value is `None`.

presumeFeasibleBCRegionAtStart

A Boolean specifying whether to ignore automatically frozen boundary condition regions in the first design cycle. This is used with *freezeBoundaryConditionRegions*. The default value is ON.

quadMaxAngle

A Float specifying the maximum angle for quad elements during mesh smoothing. The default value is 160.0.

quadMinAngle

A Float specifying the minimum angle for quad elements during mesh smoothing. The default value is 20.0.

quadSkew

A Float specifying the skew angle for quad elements during mesh smoothing, used with *reportQualityViolation*. The default value is 30.0.

quadTaper

A Float specifying the taper for quad elements during mesh smoothing, used with *reportQualityViolation*. The default value is 0.5.

region

The SymbolicConstant MODEL or a *Region* object specifying the region to which the optimization task is applied. The default value is MODEL.

reportPoorQualityElements

A Boolean specifying whether to report poor quality elements during mesh smoothing. The default value is OFF.

reportQualityViolation

A Boolean specifying whether to report a quality criteria violation during mesh smoothing. The default value is OFF.

shrinkScaleFactor

A Float specifying the scale factor to apply to optimization displacements for nodes with shrinkage. The default value is 1.0.

smoothingRegion

`None` or a *Region* object specifying the mesh smoothing region, used when *meshSmoothingRegionMethod* is REGION. The default value is `None`.

targetMeshQuality

A SymbolicConstant specifying the target mesh quality for mesh smoothing. Possible values are NONE, LOW, MEDIUM, and HIGH. The default value is LOW.

tetAspectRatio

A Float specifying the tet element aspect ratio during mesh smoothing. The default value is 100.0.

tetMaxAspect

A Float specifying the maximum tet element aspect ratio during mesh smoothing. The default value is 8.0.

tetMinAspect

A Float specifying the minimum tet element aspect ratio during mesh smoothing. The default value is 0.222.

tetSkew

A Float specifying the tet element skew value during mesh smoothing. The default value is 100.0.

triMaxAngle

A Float specifying the tri element maximum angle during mesh smoothing. The default value is 140.0.

triMinAngle

A Float specifying the tri element maximum angle during mesh smoothing. The default value is 20.0.

updateShapeBasisVectors

A SymbolicConstant specifying whether to update shape basis vectors in the first design cycle or every design cycle. Possible values are EVERY_CYCLE and FIRST_CYCLE. The default value is EVERY_CYCLE.

groupOperator

A Boolean specifying whether the group in the design response will be evaluated using the existing algorithm or a new algorithm based on Abaqus sensitivities. The default value of False means that the existing algorithm will be used.

Return value

A ShapeTask object.

Exceptions

None.

setValues(...)

This method modifies the ShapeTask object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ShapeTask* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The ShapeTask object has members with the same names and descriptions as the arguments to the *ShapeTask* method.

In addition, the ShapeTask object can have the following members:

designResponses	A repository of *DesignResponse* objects.
objectiveFunctions	A repository of *ObjectiveFunction* objects.
optimizationConstraints	A repository of *OptimizationConstraint* objects.
geometricRestrictions	A repository of *GeometricRestriction* objects.
stopConditions	A repository of *StopCondition* objects.

SingleTermDesignResponse object

The SingleTermDesignResponse object defines a single-term design response.

The SingleTermDesignResponse object is derived from the *DesignResponse* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].designResponses[name]
```

SingleTermDesignResponse(...)

This method creates a SingleTermDesignResponse object.

Path

```
mdb.models[name].optimizationTasks[name].SingleTermDesignResponse
```

Required arguments

name A String specifying the design response repository key.

identifier A String specifying the name of the variable identifier.

Optional arguments

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

drivingRegion `None` or a sequence of Floats specifying the driving region used when *identifier* is an internal nodal variable. The default value is `None`.

operation A SymbolicConstant specifying the operation used on values in the region. Possible values are MAXIMUM, MINIMUM, and SUM. The default value is SUM.

region The SymbolicConstant MODEL or a *Region* object specifying the region of the design response variable. The default value is MODEL.

shellLayer A SymbolicConstant specifying the location used for shell layer values. Possible values are BOTTOM, MAXIMUM, MIDDLE, MINIMUM, and TOP. The default value is MAXIMUM.

stepOptions A *StepOptionArray* object.

Return value

A SingleTermDesignResponse object.

Exceptions

None.

`setValues(...)`

This method modifies the SingleTermDesignResponse object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SingleTermDesignResponse* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SingleTermDesignResponse object has members with the same names and descriptions as the arguments to the *SingleTermDesignResponse* method.

SizingClusterAreas object

The SizingClusterAreas object defines a sizing cluster areas geometric restriction.

The SizingClusterAreas object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingClusterAreas(...)

This method creates a SizingClusterAreas object.

Path

```
mdb.models[name].optimizationTasks[name].SizingClusterAreas
```

Required arguments

name A String specifying the geometric restriction repository key.

regions Tuple of *Region* objects specifying the regions to which the geometric restriction is applied.

Return value

A SizingClusterAreas object.

Exceptions

None.

setValues(...)

This method modifies the SizingClusterAreas object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingClusterAreas* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SizingClusterAreas object has members with the same names and descriptions as the arguments to the *SizingClusterAreas* method.

SizingCyclicSymmetry object

The SizingCyclicSymmetry object defines a sizing cyclic symmetry geometric restriction.

The SizingCyclicSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingCyclicSymmetry(...)

This method creates a SizingCyclicSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].SizingCyclicSymmetry
```

Required arguments

name	A String specifying the geometric restriction repository key.
region	A *Region* object specifying the region to which the geometric restriction is applied.
translation	A Float specifying the translation distance.

Optional arguments

axis	A SymbolicConstant specifying the translation direction defined along an axis positioned at the *csys* origin. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.
csys	`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.
ignoreFrozenArea	A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A SizingCyclicSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the SizingCyclicSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingCyclicSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SizingCyclicSymmetry object has members with the same names and descriptions as the arguments to the *SizingCyclicSymmetry* method.

SizingFrozenArea object

The SizingFrozenArea object defines a sizing frozen area geometric restriction.

The SizingFrozenArea object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingFrozenArea (...)

This method creates a SizingFrozenArea object.

Path

```
mdb.models[name].optimizationTasks[name].SizingFrozenArea
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

None.

Return value

A SizingFrozenArea object.

Exceptions

None.

setValues(...)

This method modifies the SizingFrozenArea object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingFrozenArea* method, except for the *name* argument.

Return value
`None.`

Exceptions
None.

Members
The SizingFrozenArea object has members with the same names and descriptions as the arguments to the *SizingFrozenArea* method.

SizingMemberSize object

The SizingMemberSize object defines a sizing member size geometric restriction.

The SizingMemberSize object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingMemberSize(...)

This method creates a SizingMemberSize object.

Path

```
mdb.models[name].optimizationTasks[name].SizingMemberSize
```

Required arguments

name	A String specifying the geometric restriction repository key.
region	A *Region* object specifying the region to which the geometric restriction is applied.
minWidth	A Float specifying the min width.

setValues(...)

This method modifies the sizingMemberSize object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *sizingMemberSize* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The sizingMemberSize object has members with the same names and descriptions as the arguments to the *sizingMemberSize* method.

SizingPlanarSymmetry object

The SizingPlanarSymmetry object defines a sizing planar symmetry geometric restriction.

The SizingPlanarSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingPlanarSymmetry(...)

This method creates a SizingPlanarSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].SizingPlanarSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A SizingPlanarSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the sizingPlanarSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *sizingPlanarSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The sizingPlanarSymmetry object has members with the same names and descriptions as the arguments to the *sizingPlanarSymmetry* method.

SizingPointSymmetry object

The SizingPointSymmetry object defines a sizing point symmetry geometric restriction.

The SizingPointSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingPointSymmetry(...)

This method creates a SizingPointSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].SizingPointSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys `None` or a *DatumCsys* object specifying the position of the symmetry point defined as the origin of a local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A SizingPointSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the SizingPointSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingPointSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SizingPointSymmetry object has members with the same names and descriptions as the arguments to the *SizingPointSymmetry* method.

SizingRotationalSymmetry object

The SizingRotationalSymmetry object defines a sizing rotational symmetry geometric restriction.

The SizingRotationalSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SizingRotationalSymmetry(...)

This method creates a SizingRotationalSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].SizingRotationalSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

angle A Float specifying the repeating segment size, an angle in degrees.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A SizingRotationalSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the SizingRotationalSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingRotationalSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SizingRotationalSymmetry object has members with the same names and descriptions as the arguments to the *SizingRotationalSymmetry* method.

SizingTask object

The SizingTask object defines a Sizing task.

The SizingTask object is derived from the *OptimizationTask* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name]
```

SizingTask(...)

This method creates a SizingTask object.

Path

```
mdb.models[name].SizingTask
```

Required arguments

name A String specifying the optimization task repository key.

Optional arguments

abaqusSensitivities A Boolean specifying whether to use Abaqus to compute the design responses and their sensitivities. The default value is True.

elementThicknessDeltaStopCriteria A Float specifying the stop criteria based on the change in element thickness. The default value is 0.5×10^{-2}.

freezeBoundaryConditionRegions A Boolean specifying whether to exclude elements with boundary conditions from the optimization. The default value is OFF.

freezeLoadRegions A Boolean specifying whether to exclude elements with loads and elements with loaded nodes from the optimization. The default value is ON.

modeTrackingRegion The SymbolicConstatnt MODEL or a *Region* object specifying the region to use for mode tracking. The default value is MODEL.

numFulfilledStopCriteria An Int specifying the number of stop criteria. The default value is 2.

numTrackedModes An Int specifying the number of modes included in mode tracking. The default value is 5.

objectiveFunctionDeltaStopCriteria A Float specifying the stop criteria based on the change in objective function. The default value is 0.001.

stopCriteriaDesignCycle An Int specifying the first design cycle used to evaluate convergence criteria. The default value is 4.

thicknessMoveLimit A Float specifying the maximum change in thickness per design cycle. The default value is 0.25.

thicknessUpdateStrategy A SymbolicConstant specifying the strategy for how the thickness is updated in the method of moving asymptotes. Possible values are NORMAL, CONSERVATIVE, and AGGRESSIVE. The default value is NORMAL.

groupOperator A Boolean specifying whether the group in the design response will be evaluated using the existing algorithm or a new algorithm based on Abaqus sensitivities. The default value of False means that the existing algorithm will be used.

Return value

A SizingTask object.

Exceptions

None.

`setValues(...)`

This method modifies the SizingTask object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SizingTask* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SizingTask object has members with the same names and descriptions as the arguments to the *SizingTask* method.

In addition, the SizingTask object can have the following members:

designResponses A repository of *DesignResponse* objects.

objectiveFunctions A repository of *ObjectiveFunction* objects.

optimizationConstraints

A repository of *OptimizationConstraint* objects.

geometricRestrictions

A repository of *GeometricRestriction* objects.

stopConditions

A repository of *StopCondition* objects.

SlideRegionControl object

The SlideRegionControl object defines a slide region control geometric restriction.

The SlideRegionControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

SlideRegionControl(...)

This method creates a SlideRegionControl object.

Path

```
mdb.models[name].optimizationTasks[name].SlideRegionControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the axis of revolution. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates. This is used when *approach* is TURN.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

approach
A SymbolicConstant specifying the restriction approach. The SymbolicConstant FREE_FORM indicates a free-form slide region, and the SymbolicConstant TURN indicates that the restriction should conserve a turnable surface. Possible values are FREE_FORM and TURN. The default value is FREE_FORM.

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. This is used when *approach* is TURN. The default value is `None`.

freeFormRegion
`None` or a *Region* object specifying the free-form region. This is used when *approach* is FREE_FORM. The default value is `None`.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

revolvedRegion `None` or a *Region* object specifying the region to revolve into a slide region. This is used when *approach* is TURN. The default value is `None`.

tolerance1 A Float specifying the geometric tolerance in the 1-direction. This is used when *approach* is TURN. The default value is 0.01.

tolerance2 A Float specifying the geometric tolerance in the 2-direction. This is used when *approach* is TURN. The default value is 0.01.

tolerance3 A Float specifying the geometric tolerance in the 3-direction. This is used when *approach* is TURN. The default value is 0.01.

Return value

A SlideRegionControl object.

Exceptions

None.

`setValues(...)`

This method modifies the SlideRegionControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SlideRegionControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SlideRegionControl object has members with the same names and descriptions as the arguments to the *SlideRegionControl* method.

StampControl object

The StampControl object defines a stamp control geometric restriction.

The StampControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

StampControl(...)

This method creates a StampControl object.

Path

```
mdb.models[name].optimizationTasks[name].StampControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the stamping direction. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

drawAngle
A Float specifying the draw angle. The default value is 0.0.

mainPoint
`None` or a *Region* object specifying the main point used when *mainPointDetermination* is SPECIFY. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for assigning point priority. Possible values are MAXIMUM, MINIMUM, and SPECIFY. The default value is MAXIMUM.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

tolerance1 A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.

tolerance2 A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.

tolerance3 A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.

undercutTolerance A Float specifying the undercut tolerance. The default value is 0.0.

Return value

A StampControl object.

Exceptions

None.

`setValues(...)`

This method modifies the StampControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StampControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The StampControl object has members with the same names and descriptions as the arguments to the *StampControl* method.

StepOption object

A StepOption is an object used to define step options in a design response.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].designResponses[name]\
.stepOptions[i]
```

Members

The StepOption object can have the following members:

lowerMode

The SymbolicConstant ALL or an Int specifying the lower mode in the range of modes to consider in the step. *lowerMode* is ignored for steps without modes. The default value is ALL.

upperMode

The SymbolicConstant ALL or an Int specifying the upper mode in the range of modes to consider in the step. *upperMode* is ignored for steps without modes. The default value is ALL.

loadCase

The SymbolicConstant ALL or a String specifying the name of the load case. *loadCase* is ignored when the specified *step* does not contain a load case. The default value is ALL.

step

The SymbolicConstant ALL or a String specifying the name of the step. The default value is ALL.

model

A string specifying the name of the model from which the steps are supposed to be used in the design response. Specify only if the steps are not from the current model.

StopCondition object

The StopCondition object is the abstract base type for other StopCondition objects. The StopCondition object has no explicit constructor. The methods and members of the StopCondition object are common to all objects derived from StopCondition.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].stopConditions[name]
```

Members

The StopCondition object can have the following members:

name

A String specifying the stop condition repository key.

region

The SymbolicConstant MODEL or a *Region* object specifying the region to which the stop condition is applied. The default value is MODEL.

TopologyCyclicSymmetry object

The TopologyCyclicSymmetry object defines a topology cyclic symmetry geometric restriction.

The TopologyCyclicSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyCyclicSymmetry(...)

This method creates a TopologyCyclicSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyCyclicSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

translation A Float specifying the translation distance.

Optional arguments

axis A SymbolicConstant specifying the translation direction defined along an axis positioned at the *csys* origin. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A TopologyCyclicSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyCyclicSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyCyclicSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyCyclicSymmetry object has members with the same names and descriptions as the arguments to the *TopologyCyclicSymmetry* method.

TopologyDemoldControl object

The TopologyDemoldControl object defines a topology demold control geometric restriction.

The TopologyDemoldControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyDemoldControl(...)

This method creates a TopologyDemoldControl object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyDemoldControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system of the *pullDirection*. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int indicating the identifier of the *DatumCsys*. The default value is `None`.

draftAngle
A Float specifying the draft angle. The default value is 0.0.

collisionCheckRegion
The SymbolicConstant DEMOLD_REGION or a *Region* object specifying the collision check region. If the value is DEMOLD_REGION, then the value of *region* is used as both the demold region and the collision check region. The default value is DEMOLD_REGION.

pointRegion
A *Region* object specifying the point on a plane perpendicular to the pull direction, used to specify the central plane when *technique* is POINT.

pullDirection
A *VertexArray* object of length 2 specifying the demold pull direction. Instead of through a *Vertex*, each point may be specified through a tuple of coordinates.

technique
A SymbolicConstant specifying the demold technique. Possible values are AUTO, AUTO_TIGHT, POINT, SURFACE, and STAMP. The default value is AUTO.

Return value

A TopologyDemoldControl object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyDemoldControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyDemoldControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyDemoldControl object has members with the same names and descriptions as the arguments to the *TopologyDemoldControl* method.

TopologyMemberSize object

The TopologyMemberSize object defines a topology member size geometric restriction.

The TopologyMemberSize object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyMemberSize(...)

This method creates a TopologyMemberSize object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyMemberSize
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

maxThickness A Float specifying the maximum thickness. The default value is 0.0.

minThickness A Float specifying the minimum thickness. The default value is 0.0.

separation A Float specifying the minimum gap. The default value is 0.0.

sizeRestriction A SymbolicConstant specifying whether to restrict the minimum or maximum thickness or an envelope of both. Possible values are ENVELOPE, MAXIMUM, and MINIMUM. The default value is MINIMUM.

method A SymbolicConstant specifying the algorithm for the maximum member size constraint. Possible values are FILTER and LOCAL_VOLUME. The default value is FILTER.

Return value

A TopologyMemberSize object.

Exceptions

None.

setValues(...)

This method modifies the TopologyMemberSize object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyMemberSize* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyMemberSize object has members with the same names and descriptions as the arguments to the *TopologyMemberSize* method.

TopologyOverhangControl object

The TopologyOverhangControl object defines a topology overhang control geometric restriction.

The TopologyOverhangControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyOverhangControl(...)

This method creates a TopologyOverhangControl object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyOverhangControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

pullDirection
A *VertexArray* object of length 2 specifying the overhang control print direction. Instead of through a *Vertex*, each point can be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system of the *pullDirection*. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int indicating the identifier of the *DatumCsys*. The default value is `None`.

draftAngle
A Float specifying the overhang angle. The default value is 45.0.

overhangCheckRegion
The SymbolicConstant OVERHANG_REGION or a *Region* object specifying the overhang check region. If the value is OVERHANG_REGION, the value of *region* is used as both the overhang control region and the overhang check region. The default value is OVERHANG_REGION.

pointRegion
A *Region* object specifying the point on a plane perpendicular to the *pullDirection* that is used to specify the base plane when *technique* is POINT.

radius
A Float specifying the radius to define the size of the cones that are used in the internal check for the overhang criteria.

technique A SymbolicConstant specifying the overhang control technique used to define the base plane. Possible values are AUTO, POINT, and NONE. The default value is AUTO.

Return value

A TopologyOverhangControl object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyOverhangControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyOverhangControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyOverhangControl object has members with the same names and descriptions as the arguments to the *TopologyOverhangControl* method.

TopologyPlanarSymmetry object

The TopologyPlanarSymmetry object defines a topology planar symmetry geometric restriction.

The TopologyPlanarSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyPlanarSymmetry(...)

This method creates a TopologyPlanarSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyPlanarSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A TopologyPlanarSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyPlanarSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyPlanarSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyPlanarSymmetry object has members with the same names and descriptions as the arguments to the *TopologyPlanarSymmetry* method.

TopologyPointSymmetry object

The TopologyPointSymmetry object defines a topology point symmetry geometric restriction.

The TopologyPointSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyPointSymmetry(...)

This method creates a TopologyPointSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyPointSymmetry
```

Required arguments

name
A String specifying the geometric restriction repository key.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the position of the symmetry point defined as the origin of a local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea
A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A TopologyPointSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the TopologyPointSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyPointSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyPointSymmetry object has members with the same names and descriptions as the arguments to the *TopologyPointSymmetry* method.

TopologyRibDesign object

The TopologyRibDesign object defines a topology rib design geometric restriction.

The TopologyRibDesign object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyRibDesign(...)

This method creates a TopologyRibDesign object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyRibDesign
```

Required arguments

name
A String specifying the geometric restriction repository key.

ribDirection
A *VertexArray* object of length 2 specifying the out-of-plane growth direction of the ribs. Instead of through a *Vertex*, each point can be specified through a tuple of coordinates.

ribThickness
A Float specifying the average thickness of the ribs.

ribDistance
A Float specifying the average distance between the rib centers. The distance must be larger than twice the average element edge length.

region
A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system of the *pullDirection*. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int indicating the identifier of the *DatumCsys*. The default value is `None`.

ribDesignCheckRegion
The SymbolicConstant RIBDESIGN_REGION or a *Region* object specifying the overhang check region. If the value is OVERHANG_REGION, the value of *region* is used as both the overhang control region and the overhang check region. The default value is RIBDESIGN_REGION.

Return value

A TopologyRibDesign object.

Exceptions

None.

setValues(...)

This method modifies the TopologyRibDesign object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyRibDesign* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyRibDesign object has members with the same names and descriptions as the arguments to the *TopologyRibDesign* method.

TopologyRotationalSymmetry object

The TopologyRotationalSymmetry object defines a topology rotational symmetry geometric restriction.

The TopologyRotationalSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyRotationalSymmetry(...)

This method creates a TopologyRotationalSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyRotationalSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

angle A Float specifying the repeating segment size, an angle in degrees.

region A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

ignoreFrozenArea A Boolean specifying whether to ignore frozen areas. The default value is OFF.

Return value

A TopologyRotationalSymmetry object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyRotationalSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyRotationalSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyRotationalSymmetry object has members with the same names and descriptions as the arguments to the *TopologyRotationalSymmetry* method.

TopologyMillingControl object

The TopologyMillingControl object defines a topology milling control geometric restriction.

The TopologyMillingControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TopologyMillingControl(...)

This method creates a TopologyMillingControl object.

Path

```
mdb.models[name].optimizationTasks[name].TopologyMillingControl
```

Required arguments

name A String specifying the geometric restriction repository key.

millingDirections A tuple of *VertexArray* objects of length 2 specifying the milling directions. Each point can be specified through a tuple of coordinates instead of through a *Vertex*.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys `None` or a *DatumCsys* object specifying the local coordinate system of the *millingDirections*. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int indicating the identifier of the *DatumCsys*. The default value is `None`.

millingCheckRegion The SymbolicConstant MILLING_REGION or a *Region* object specifying the milling check region. If the value is MILLING_REGION, the value of *region* is used as both the milling control region and the milling check region. The default value is MILLING_REGION.

radius A Float specifying the radius for the collision check during the removal of the elements for the milling criteria.

Return value

A TopologyMillingControl object.

Exceptions

None.

`setValues(...)`

This method modifies the TopologyMillingControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyMillingControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyMillingControl object has members with the same names and descriptions as the arguments to the *TopologyMillingControl* method.

TopologyTask object

The TopologyTask object defines a topology task.

The TopologyTask object is derived from the *OptimizationTask* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name]
```

TopologyTask(...)

This method creates a TopologyTask object.

Path

```
mdb.models[name].TopologyTask
```

Required arguments

name A String specifying the optimization task repository key.

Optional arguments

abaqusSensitivities A Boolean specifying whether to use Abaqus to compute the design responses and their sensitivities. The default value is True.

algorithm A SymbolicConstant specifying the optimization task algorithm. Possible values are GENERAL_OPTIMIZATION and CONDITION_BASED_OPTIMIZATION. The default value is GENERAL_OPTIMIZATION.

densityMoveLimit A Float specifying the maximum density change per design cycle. The default value is 0.25.

densityUpdateStrategy A SymbolicConstant specifying the strategy for how the densities are updated in the method of moving asymptotes. Possible values are NORMAL, CONSERVATIVE, and AGGRESSIVE. The default value is NORMAL.

elementDensityDeltaStopCriteria A Float specifying the stop criteria based upon the change in element densities. The default value is 0.5×10^{-2}.

filterRadius `None` or a Float specifying the mesh filter radius for mesh independence and minimum size. The default value is `None`.

firstCycleDeletedVolume A Float specifying the volume that can be removed immediately in the first design cycle. The default value is 5.0.

firstCycleDeletedVolumeTechnique
A SymbolicConstant specifying the method of quantifying volume that can be removed immediately in the first design cycle. Possible values are OFF, PERCENTAGE, and ABSOLUTE. The default value is OFF.

freezeBoundaryConditionRegions
A Boolean specifying whether to exclude elements with boundary conditions from the optimization. The default value is OFF.

freezeLoadRegions
A Boolean specifying whether to exclude elements with loads and elements with loaded nodes from the optimization. The default value is ON.

frequencySpectrumWeight
A Float specifying the weighting factor for frequency spectrum peaks. The default value is 6.0.

initialDensity
A SymbolicConstant specifying the Optimization product default or a float specifying the initial density. The default value is DEFAULT.

materialInterpolationPenalty
A Float specifying the penalty factor for the material interpolation technique. The default value is 3.0.

materialInterpolationTechnique
A SymbolicConstant specifying the material interpolation technique: optimization product default, solid isotropic material with penalization, or rational approximation of material properties. Possible values are DEFAULT, SIMP, and RAMP. The default value is DEFAULT.

maxDensity
A Float specifying the maximum density in the density update. The default value is 1.0.

minDensity
A Float specifying the minimum density in the density update. The default value is 10^{-3}.

modeTrackingRegion
The SymbolicConstant MODEL or a *Region* object specifying the region to use for mode tracking. The default value is MODEL.

numDesignCycles
An Int specifying the number of design cycles permitted when *stepSize* is DYNAMIC. The default value is 15.

numFulfilledStopCriteria
An Int specifying the number of stop criteria. The default value is 2.

numTrackedModes
An Int specifying the number of modes included in mode tracking. The default value is 5.

objectiveFunctionDeltaStopCriteria
A Float specifying the stop criteria based on the change in objective function. The default value is 10^{-3}.

region
The SymbolicConstant MODEL or a *Region* object specifying the region to which the optimization task is applied. The default value is MODEL.

softDeletionMethod
A SymbolicConstant specifying the method used when *softDeletionRegion* is specified. The STANDARD method avoids creating disconnected regions. The

AGGRESSIVE method only considers the *softDeletionThreshold*. The MAX_SHEAR_STRAIN, MAX_ELASTOPLASTIC_STRAIN and VOLUME_COMPRESSION methods do not need the *softDeletionRadius*. Possible values are STANDARD, AGGRESSIVE, MAX_SHEAR_STRAIN, MIN_PRINCIPAL_STRAIN, MAX_ELASTOPLASTIC_STRAIN and VOLUME_COMPRESSION. The default value is STANDARD.

softDeletionRadius

A Float specifying the radius to use when considering neighboring soft elements to delete. The default value is 0.0.

softDeletionRegion

`None` or a *Region* object specifying the region in which the soft elements should be deleted during optimization. The default value is `None`.

softDeletionThreshold

A Float specifying the relative material density value used to identify soft elements. Those with values below the threshold are considered for removal. For STANDARD and AGGRESSIVE methods positive values are accepted and the default value is 0.05. For MAX_SHEAR_STRAIN and MAX_ELASTOPLASTIC_STRAIN methods positive values are accepted whereas for MIN_PRINCIPAL_STRAIN and VOLUME_COMPRESSION methods negative values are accepted.

stepSize

A SymbolicConstant specifying the size of the increment for volume modification. Possible values are DYNAMIC, VERY_SMALL, SMALL, MODERATE, MEDIUM, and LARGE. The default value is MEDIUM.

stiffnessMassDamping

The SymbolicConstant AVERAGE_EDGE_LENGTH or a Float specifying the stiffness mass damping for the task region. The default value is AVERAGE_EDGE_LENGTH.

stopCriteriaDesignCycle

An Int specifying the first design cycle used to evaluate convergence criteria. The default value is 4.

structuralMassDamping

`None` or a Float specifying the structural mass damping for the task region. The default value is `None`.

viscousMassDamping

`None` or a Float specifying the viscous mass damping for the task region. The default value is `None`.

viscousStiffnessDamping

`None` or a Float specifying the viscous stiffness damping for the task region. The default value is `None`.

groupOperator

A Boolean specifying whether the group in the design response will be evaluated using the existing algorithm or a new algorithm based on Abaqus sensitivities. The default value of False means that the existing algorithm will be used.

Return value

A TopologyTask object.

Exceptions

None.

setValues(...)

This method modifies the TopologyTask object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TopologyTask* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TopologyTask object has members with the same names and descriptions as the arguments to the *TopologyTask* method.

In addition, the TopologyTask object can have the following members:

designResponses	A repository of *DesignResponse* objects.
objectiveFunctions	A repository of *ObjectiveFunction* objects.
optimizationConstraints	A repository of *OptimizationConstraint* objects.
geometricRestrictions	A repository of *GeometricRestriction* objects.
stopConditions	A repository of *StopCondition* objects.

TurnControl object

The TurnControl object defines a turn control geometric restriction.

The TurnControl object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

TurnControl(...)

This method creates a TurnControl object.

Path

```
mdb.models[name].optimizationTasks[name].TurnControl
```

Required arguments

name
A String specifying the geometric restriction repository key.

clientDirection
A *VertexArray* object of length 2 specifying the direction of the rotation axis as a vector positioned at the *csys* origin. Instead of through a *Vertex*, each point might be specified through a tuple of coordinates.

region
A *Region* object specifying the region to which the geometric restriction is applied. When used with a *TopologyTask*, there is no default value. When used with a *ShapeTask*, the default value is MODEL.

Optional arguments

csys
`None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

mainPoint
`None` or a *Region* object specifying the main point used when *mainPointDetermination* is SPECIFY. The default value is `None`.

mainPointDetermination
A SymbolicConstant specifying the rule for assigning point priority. Possible values are MAXIMUM, MINIMUM, and SPECIFY. The default value is MAXIMUM.

presumeFeasibleRegionAtStart
A Boolean specifying whether to ignore the geometric restriction in the first design cycle. The default value is ON.

tolerance1
A Float specifying the geometric tolerance in the 1-direction. The default value is 0.01.

tolerance2
A Float specifying the geometric tolerance in the 2-direction. The default value is 0.01.

tolerance3
A Float specifying the geometric tolerance in the 3-direction. The default value is 0.01.

Return value

A TurnControl object.

Exceptions

None.

`setValues(...)`

This method modifies the TurnControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TurnControl* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The TurnControl object has members with the same names and descriptions as the arguments to the *TurnControl* method.

BeadFilter object

The BeadFilter object defines a growth geometric restriction.

The BeadFilter object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadFilter(...)

This method creates a BeadFilter object.

Path

```
mdb.models[name].optimizationTasks[name].BeadGrowth
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

radius A Float specifying the filter radius. The default value is double the average edge length of the model.

filterRadiusBy The SymbolicConstant defines whether the filter radius is in absolute or relative units. For an absolute radius, the value is ABSOLUTE_VALUE. For a relative radius, the value is RELATIVE. The default value is ABSOLUTE_VALUE.

filterCheckRegion The SymbolicConstant FILTER_REGION or a *Region* object specifying the filter check region. If the value is FILTER_REGION, the value of the region is used as both the filter region and the filter check region. The default value is FILTER_REGION.

Return value

A BeadFilter object.

Exceptions

None.

`setValues(...)`

This method modifies the BeadFilter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadFilter* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadFilter object has members with the same names and descriptions as the arguments to the *BeadFilter* method.

BeadFixedRegion object

The BeadFixedRegion object defines a fixed region geometric restriction.

The BeadFixedRegion object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadFixedRegion(...)

This method creates a BeadFixedRegion object.

Path

```
mdb.models[name].optimizationTasks[name].BeadFixedRegion
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

u1 A Boolean specifying whether to fix the region in the 1-direction. The default value is OFF.

u2 A Boolean specifying whether to fix the region in the 2-direction. The default value is OFF.

u3 A Boolean specifying whether to fix the region in the 3-direction. The default value is OFF.

Return value

A BeadFixedRegion object.

Exceptions

None.

setValues(...)

This method modifies the BeadFixedRegion object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadFixedRegion* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadFixedRegion object has members with the same names and descriptions as the arguments to the *BeadFixedRegion* method.

BeadGrowth object

The BeadGrowth object defines a growth geometric restriction.

The BeadGrowth object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadGrowth(...)

This method creates a BeadGrowth object.

Path

```
mdb.models[name].optimizationTasks[name].BeadGrowth
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

beadGrowth A Float specifying the maximum optimization displacement in the growth direction. Either *beadGrowth* or *shrink* or both must be specified. The default value is 0.0.

shrink A Float specifying the maximum optimization displacement in the shrink direction. Either *beadGrowth* or *shrink* or both must be specified The default value is 0.0.

Return value

A BeadGrowth object.

Exceptions

None.

setValues(...)

This method modifies the BeadGrowth object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadGrowth* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadGrowth object has members with the same names and descriptions as the arguments to the *BeadGrowth* method.

BeadPenetrationCheck object

The BeadPenetrationCheck object defines a penetration check geometric restriction.

The BeadPenetrationCheck object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadPenetrationCheck(...)

This method creates a BeadPenetrationCheck object.

Path

```
mdb.models[name].optimizationTasks[name].BeadPenetrationCheck
```

Required arguments

name	A String specifying the geometric restriction repository key.
beadPenetrationCheckRegion	A *Region* object specifying the penetration check region.
region	A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

None.

Return value

A BeadPenetrationCheck object.

Exceptions

None.

setValues(...)

This method modifies the BeadPenetrationCheck object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadPenetrationCheck* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadPenetrationCheck object has members with the same names and descriptions as the arguments to the *BeadPenetrationCheck* method.

BeadPlanarSymmetry object

The BeadPlanarSymmetry object defines a bead planar symmetry geometric restriction.

The BeadPlanarSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadPlanarSymmetry(...)

This method creates a BeadPlanarSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].BeadPlanarSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

Return value

A BeadPlanarSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the BeadPlanarSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadPlanarSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadPlanarSymmetry object has members with the same names and descriptions as the arguments to the *BeadPlanarSymmetry* method.

BeadPointSymmetry object

The BeadPointSymmetry object defines a point symmetry geometric restriction.

The BeadPointSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadPointSymmetry(...)

This method creates a BeadPointSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].BeadPointSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

csys `None` or a *DatumCsys* object specifying the position of the symmetry point defined as the origin of a local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

Return value

A BeadPointSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the BeadPointSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadPointSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadPointSymmetry object has members with the same names and descriptions as the arguments to the *BeadPointSymmetry* method.

BeadRotationalSymmetry object

The BeadRotationalSymmetry object defines a bead rotational symmetry geometric restriction.

The BeadRotationalSymmetry object is derived from the *GeometricRestriction* object.

Access

```
import optimization
mdb.models[name].optimizationTasks[name].geometricRestrictions[name]
```

BeadRotationalSymmetry(...)

This method creates a BeadRotationalSymmetry object.

Path

```
mdb.models[name].optimizationTasks[name].BeadRotationalSymmetry
```

Required arguments

name A String specifying the geometric restriction repository key.

angle A Float specifying the repeating segment size, an angle in degrees.

region A *Region* object specifying the region to which the geometric restriction is applied.

Optional arguments

axis A SymbolicConstant specifying the axis of symmetry. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

csys `None` or a *DatumCsys* object specifying the local coordinate system. If *csys*=`None`, the global coordinate system is used. When this member is queried, it returns an Int. The default value is `None`.

Return value

A BeadRotationalSymmetry object.

Exceptions

None.

setValues(...)

This method modifies the BeadRotationalSymmetry object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeadRotationalSymmetry* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeadRotationalSymmetry object has members with the same names and descriptions as the arguments to the *BeadRotationalSymmetry* method.

Part commands

Features in Abaqus/CAE include Parts, Datums, Partitions, and Assembly operations. Part commands create Feature objects on only the Part object.
The commands that create Feature objects on only the *rootAssembly* object are described in *Assembly commands*.
The commands that create Feature objects on both the Part and the *rootAssembly* objects are described in *Feature commands*.

In this section:

- *Part object*
- *Feature object*
- *AcisFile object*

Part object

The Part object defines the physical attributes of a structure. Parts are instanced into the assembly and positioned before an analysis.

Access

```
import part
mdb.models[name].parts[name]
```

Part(...)

This method creates a Part object and places it in the parts repository.

Path

```
mdb.models[name].Part
```

Required arguments

name A String specifying the repository key.

dimensionality A SymbolicConstant specifying the dimensionality of the part. Possible values are THREE_D, TWO_D_PLANAR, and AXISYMMETRIC.

type A SymbolicConstant specifying the type of the part. Possible values are DEFORMABLE_BODY, EULERIAN, DISCRETE_RIGID_SURFACE, and ANALYTIC_RIGID_SURFACE.

Optional arguments

twist A Boolean specifying whether to include a twist DEGREE OF FREEDOM in the part (available only when *dimensionality*=AXISYMMETRIC and *type*=DEFORMABLE_BODY). The default value is OFF.

Return value

A Part object.

Exceptions

InvalidNameError.

Part(...)

This method copies a Part object and places the copy in the parts repository.

Path

```
mdb.models[name].Part
```

Required arguments

name A String specifying the repository key.

objectToCopy A Part Object to be copied.

Optional arguments

scale A Float specifying the scaling factor to apply to the imported geometric entities during copy. If a scale is specified, *compressFeatureList* will be set to ON, regardless of whether it is specified in the command. The default value is 1.

mirrorPlane A SymbolicConstant specifying how the part is to be mirrored during copy. Possible values are XYPLANE, XZPLANE, YZPLANE. If a mirror plane is specified, *compressFeatureList* will be set to ON, regardless of whether it is specified in the command. The default value is NONE.

compressFeatureList A Boolean specifying whether to compress the feature list when copying a Part object. The default value is OFF. If *mirrorPlane* or *scale* is specified, this argument is ignored.

When you compress the feature, list the resulting part will have a single feature. Any datums or sets in the original part will be lost.

separate A Boolean specifying whether to separate disconnected regions into parts. The default value is OFF.

Return value

A Part object.

Exceptions

InvalidNameError.

PartFromBooleanCut(...)

This method creates a Part in the parts repository after subtracting or cutting the geometries of a group of part instances from that of a base part instance.

Path

```
mdb.models[name].rootAssembly.PartFromBooleanCut
```

Required arguments

name A String specifying the repository key.

instanceToBeCut A *PartInstance* specifying the base instance from which to cut other instances.

cuttingInstances A sequence of *PartInstance* objects specifying the instances with which to cut the base instance.

Optional arguments

None.

Return value

A Part object.

Exceptions

InvalidNameError.

PartFromBooleanMerge(...)

This method creates a Part in the parts repository after merging two or more part instances. The part instances can be either Abaqus native parts or orphan mesh parts, but they cannot be a combination of both.

Path

```
mdb.models[name].rootAssembly.PartFromBooleanMerge
```

Required arguments

name A String specifying the repository key.

instances A sequence of *PartInstance* objects specifying the part instances to merge.

Optional arguments

keepIntersections A Boolean specifying whether the boundary intersections of Abaqus native part instances should be retained after the merge operation. The default value is False.

mergeNodes A SymbolicConstant specifying whether the nodes of orphan mesh part instances should be retained after the merge operation. Possible values are BOUNDARY_ONLY, ALL, or NONE. The default value is BOUNDARY_ONLY.

nodeMergingTolerance A Float specifying the maximum distance between nodes of orphan mesh part instances that will be merged and replaced with a single new node. The location of the new node is the average position of the deleted nodes. The default value is 10^{-6}.

removeDuplicateElements A Boolean specifying whether elements with the same connectivity after the merge will merged into a single element. The default value is ON.

domain A SymbolicConstant specifying whether the part instances being merged are geometric instances or mesh instances. Possible values are GEOMETRY, MESH or BOTH. The default value is GEOMETRY.

Return value

A Part object.

Exceptions

InvalidNameError.

PartFromExtrude2DMesh(...)

This method creates a Part object by extruding an existing two-dimensional orphan mesh Part object in the positive *Z*-direction and places it in the parts repository.

Path

```
mdb.models[modelName].PartFromExtrude2DMesh
```

Required arguments

name	A String specifying the repository key.
part	A Part object specifying an existing two-dimensional orphan mesh Part object.
depth	A Float specifying the total extrusion distance.
elementSize	A Float specifying an approximate element length in the extruded direction.

Optional arguments

None.

Return value

A Part object.

Exceptions

InvalidNameError.

If the specified part is not an orphan mesh part:	Cannot extrude a geometric part.
If the specified part is not two-dimensional:	Cannot extrude a 3D part.
If the specified part is a rigid body:	Cannot change dimension of a rigid body.

PartFromGeometryFile(...)

This method creates a Part object and places it in the parts repository.

Path

```
mdb.models[modelName].PartFromGeometryFile
```

Required arguments

name
A String specifying the repository key.

geometryFile
An *AcisFile* object specifying a file containing geometry.

dimensionality
A SymbolicConstant specifying the dimensionality of the part. Possible values are THREE_D, TWO_D_PLANAR, and AXISYMMETRIC.

type
A SymbolicConstant specifying the type of the part. Possible values are DEFORMABLE_BODY, EULERIAN, DISCRETE_RIGID_SURFACE, and ANALYTIC_RIGID_SURFACE.

Optional arguments

bodyNum
An Int specifying the desired body to be selected from an ACIS object containing a list of *N* ACIS bodies. Possible values are $1 \leq bodyNum \leq N$. The default value is 1.

combine
A Boolean specifying weather to create a single part by combining all the bodies in the ACIS object. This argument is ignored if *bodyNum* is specified. The default value is False.

booleanSolids
A Boolean specifying whether the solids should be boolean while combining all the bodies. The default value is FALSE.

retainBoundary
A Boolean specifying whether the intersecting boundaries should be retained while boolean the solids. The default value is FALSE.

usePartNameFromFile
A Boolean specifying whether the part names specified in a STEP file should be used as the names in the Abaqus model database. If this option is TRUE, the part names in the STEP file will be used; if FALSE, each imported part will be named using the text of the *name* argument followed by a number. This functionality is available only for import from STEP files; for import from all other types of files this option should be FALSE.

stitchTolerance
A Float indicating the maximum gap to be stitched. The value should be smaller than the minimum feature size and bigger than the maximum gap expected to be stitched in the model. Otherwise this command might remove small (sliver) edges that are smaller than the tolerance. The default value is 1.0

twist
A Boolean specifying whether to include a twist DEGREE OF FREEDOM in the part (available only when *dimensionality*=AXISYMMETRIC and *type*=DEFORMABLE_BODY). The default value is OFF.

scale
A Float specifying the scaling factor to apply to the imported geometric entities. The default value is 1.0.

convertToAnalytical
An Int specifying whether to convert to analytical entities. Possible values are 0 or 1. The default value is 0. If *convertToAnalytical*=1, all the numerical entities, such as splines, are converted to analytical entities, such as arcs and lines, during the repair phase of the command.

convertToPrecise An Int specifying whether to convert to precise geometry. Possible values are 0 or 1. The default value is 0. If *convertToPrecise*=1, the application will attempt to re-evaluate the tolerant entities to be more precise.

Return value

A Part object.

Exceptions

InvalidNameError.

If the ACIS file is corrupt:	PartError: the file is corrupt
If the dimensionality does not correspond to what is found in the ACIS file:	PartError: dimensionality does not match the contents of the file
If the type does not correspond to what is found in the ACIS file:	PartError: type does not match the contents of the file

PartFromInstanceMesh(...)

This method creates a Part object containing the mesh found in the supplied *PartInstance* objects and places the new Part object in the parts repository.

Path

```
mdb.models[name].rootAssembly.PartFromInstanceMesh
```

Required arguments

name A String specifying the repository key.

Optional arguments

partInstances A sequence of *PartInstance* objects to be used in the creation of the new mesh part. If the *partInstances* argument is omitted, the new Part object contains the mesh of all the part instances in the assembly.

copyPartSets A Boolean specifying whether to copy sets, surfaces, and attributes from the base part or parts of the specified part instances to the new part. The default is False.

copyAssemblySets A Boolean specifying whether to copy assembly-level sets that reference entities of the specified part instances to the new part. The default is False.

Return value

A Part object.

Exceptions

InvalidNameError.

If the analysis type (deformable or rigid) is not consistent among the supplied part instances:	The selected part instances do not have a consistent analysis type.
If the assembly does not contain a mesh:	The current assembly does not contain a mesh for a mesh part.
If the specified part instances do not contain a mesh:	The selected part instances do not have a mesh for a mesh part.

PartFromMesh(...)

This method creates a Part object containing the mesh found in the part and places the new Part object in the parts repository.

Path

```
mdb.models[name].parts[name].PartFromMesh
```

Required arguments

name	A String specifying the repository key.

Optional arguments

copySets	A Boolean specifying whether to copy sets, surfaces, and attributes to the new part. The default is False.

Return value

A Part object.

Exceptions

InvalidNameError.

If the part does not contain a mesh:	The current part does not contain a mesh for a mesh part.

PartFromMeshMirror(...)

This method creates a Part object by mirroring an existing orphan mesh Part object about a specified plane and places it in the parts repository. The result is a union of the original and the mirrored copy. Contrast the PartFromMeshMirror method with the *mirrorPlane* argument of the Part copy constructor. The *mirrorPlane* argument creates only the second half of the part but does not unite the two halves.

Path

```
mdb.models[modelName].PartFromMeshMirror
```

Required arguments

name	A String specifying the repository key.
part	A Part object specifying an existing orphan mesh part.
point1	A sequence of three Floats specifying a point on the mirror plane. This point is the local origin in the local system of the plane.
point2	A sequence of three Floats specifying a point in the direction of the normal to the mirror plane. This point must not be coincident with *point1*.

Optional arguments

None.

Return value

A Part object.

Exceptions

InvalidNameError.

If the specified part is not an orphan mesh part:	Cannot mirror a geometric part.
If the specified part is a rigid body:	Cannot mirror a rigid body.
If *point1* and *point2* are coincident:	Mirror plane director has zero length.
If the specified part is two-dimensional and the plane is not parallel to the *Z*-axis:	Mirror plane must be parallel to Z-axis for 2D parts.

PartFromNodesAndElements(...)

This method creates a Part object from nodes and elements and places it in the parts repository.

Path

```
mdb.models[modelName].PartFromNodesAndElements
```

Required arguments

name	A String specifying the repository key.
dimensionality	A SymbolicConstant specifying the dimensionality of the part. Possible values are THREE_D, TWO_D_PLANAR, and AXISYMMETRIC.
type	A SymbolicConstant specifying the type of the part. Possible values are DEFORMABLE_BODY, EULERIAN, DISCRETE_RIGID_SURFACE, and ANALYTIC_RIGID_SURFACE.

nodes
A sequence of (*nodeLabels*, *nodeCoords*) specifying the nodes of the mesh. *nodeLabels* is a sequence of Ints specifying the node labels, and *nodeCoords* is a sequence of sequences of three Floats specifying the nodal coordinates.

elements
A sequence of sequences of(*meshType*, *elementLabels*, *elementConns*) specifying the elements of the mesh. *meshType* is a String specifying the element type. *elementlabels* is a sequence of Ints specifying the element labels. *elementConns* is a sequence of sequences of node labels specifying the element connectivity.

Optional arguments

twist
A boolean specifying whether the part is defined with twist. This option has meaning only when *dimensionality*=AXISYMMETRIC. Possible values are ON and OFF. The default value is OFF.

Return value

A Part object.

Exceptions

None.

PartFromOdb(...)

This method creates an orphan mesh Part object by reading an output database. The new part is placed in the parts repository.

Path

```
mdb.models[modelName].PartFromOdb
```

Required arguments

name
A String specifying the repository key.

odb
An output database object.

Optional arguments

fileName
A String specifying the name of the output database file from which to create the part. The default value is an empty string.

instance
A String specifying the part instance in the output database from which to create the part. If no instance name is specified, Abaqus creates an orphan mesh part from the first part instance in the output database.

elementSet
A String specifying an element set defined on the output database. Only elements from this set will be imported. The default is to import all element sets.

shape
A SymbolicConstant specifying the configuration state. Possible values are UNDEFORMED and DEFORMED. The default value is UNDEFORMED.

step
An Int specifying the step number for reading deformed coordinates. $\mathbf{0 \le step \le N-1}$ where $\boldsymbol{N}$ is the number of available steps. The default value is the last available step. You should specify the *step* argument only when *shape*=DEFORMED.

frame
An Int specifying the frame number for reading deformed coordinates. $\mathbf{0 \le frame \le N-1}$ where $\boldsymbol{N}$ is the number of available frames. The default value is the last available frame. You should specify the *frame* argument only when *shape*=DEFORMED.

twist
A Boolean specifying whether to include a twist DEGREE OF FREEDOM in the part (available only when *dimensionality*=AXISYMMETRIC and *type*=DEFORMABLE_BODY). The default value is OFF.

Return value

A Part object.

Exceptions

InvalidNameError.

If the output database contains elements of more than one dimensionality or type:	File contains both axisymmetric and nonaxisymmetric elements. File contains both 2D and 3D elements. File contains both rigid and deformable elements.
If more than one part is found on the output database:	PartError: importing of more than one part is not currently supported
If the output database does not contain any valid results for the specified step:	Error. File does not contain any valid frames.
If the specified step and frame do not contain any displacements:	Error. Specified frame does not contain nodal displacements.
If the specified element set is not found on the output database:	Error. Specified element set is not defined in the ODB.
If the step number is invalid:	OdiError: Invalid step index: i. Available step indices: 0 - j.
If the frame number is invalid:	OdiError: Invalid frame index: i. Available frame indices: 0 - j.

PartFromSection3DMeshByPlane(...)

This method creates a Part object by cutting an existing three-dimensional orphan mesh Part object by a plane and places it in the parts repository. This method is valid only for orphan mesh parts composed of 8-node brick elements.

Path

```
mdb.models[modelName].PartFromSection3DMeshByPlane
```

Required arguments

name	A String specifying the repository key.
part	A Part object specifying an existing three-dimensional orphan mesh part.
point1	A Sequence of three Floats specifying a point on the cutting plane. This point is the local origin in the local system of the plane.
point2	A Sequence of three Floats specifying a point in the direction of the normal to the cutting plane. This point must not be coincident with *point1*.
point3	A sequence of three Floats specifying the direction of the local 1-axis in the local system of the plane. This point must not project onto *point1*.

Optional arguments

None.

Return value

A Part object.

Exceptions

InvalidNameError.

If the specified part is not an orphan mesh part:	Cannot reduce dimension of a geometric part.
If the specified part is not three-dimensional:	Cannot reduce dimension of a 2D part.
If the specified part is a rigid body:	Cannot change dimension of a rigid body.
If *point1* and *point2* are coincident:	Cutting plane director has zero length.
If *point3* projects onto *point1*:	Local axis point projects to origin.
If no elements are cut by the specified plane:	Cannot reduce part dimension.

PartFromSubstructure(...)

This method creates a substructure Part object by reading a substructure sim file and places it in the parts repository.

Path

```
mdb.models[name].PartFromSubstructure
```

Required arguments

name	A String specifying the repository key.
substructureFile	A substructure sim file.
odbFile	The output database file corresponding to the substructure sim file.

Optional arguments

None.

Return value

A Part object.

Exceptions

InvalidNameError.

If the specified part is not a substructure:	File specified does not contain a substructure.
If the specified part already exists:	A part with the same name already exists.
If the substructure cannot be imported:	The output database is missing nodes and elements. Nested substructures are not supported. The substructure sim file was generated using a version that is different from the current version.

Part2DGeomFrom2DMesh(...)

This method creates a geometric Part object from the outline of an existing two-dimensional orphan mesh Part object and places it in the parts repository. If the Part2DGeomFrom2DMesh method cannot create a valid two-dimensional shell section from the two-dimensional mesh, the method fails and creates an empty geometry part with a failed base shell feature.

Path

```
mdb.models[modelName].Part2DGeomFrom2DMesh
```

Required arguments

name	A String specifying the repository key.
part	A Part object specifying an existing two-dimensional orphan mesh Part object.
featureAngle	A Float specifying the angle (in degrees) between line segments that triggers a break in the geometry.

Optional arguments

splineCurvatureLimit A Float specifying the traversal angle in degrees of the spline that triggers a break in the geometry. The default value is 90.

twist A Boolean specifying whether to include a twist DEGREE OF FREEDOM in the part (available only when *dimensionality*=AXISYMMETRIC and *type*=DEFORMABLE_BODY). The default value is OFF.

Return value

A Part object.

Exceptions

InvalidNameError.

If the specified part is not an orphan mesh part:	Specified part must be an orphan mesh.
If the Part2DGeomFrom2DMesh method cannot create a valid two-dimensional shell section from the two-dimensional mesh:	Planar shell feature failed
If the specified part is not two-dimensional:	Cannot create a geometry from a 3D part.
If the specified part is a rigid body:	Cannot create a geometry from a rigid body.

`setValues(...)`

This method modifies the Part object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *`Part`* method. In addition, `setValues` has the following optional argument:

geometryRefinement A SymbolicConstant specifying how the part's surface is being refined when faceted for display. Possible values are EXTRA_COARSE, COARSE, MEDIUM, FINE, and EXTRA_FINE.

startNodeLabel A positive Integer specifying the new start node label for the part mesh that currently exists or will be generated. If the part is meshed, Abaqus/CAE changes the node labels while preserving the original order and incrementation.

startElemLabel A positive Integer specifying the new start element label for the part mesh that currently exists or will be generated. If the part is meshed, Abaqus/CAE changes the element labels while preserving the original order and incrementation.

Return value
`None.`

Exceptions
RangeError.

`addGeomToSketch(...)`

This method converts a part into a sketch by projecting all the edges of the part onto the X-Y plane of the sketch. You can use addGeomToSketch with a part of any modeling space.

Required arguments

sketch A *ConstrainedSketch* object.

Optional arguments
None.

Return value
`None.`

Exceptions
None.

`assignThickness(...)`

This method assigns thickness data to shell faces. The thickness can be used while assigning shell and membrane sections to faces.

Required arguments

faces A sequence of *Face* objects specifying the regions where thickness will be applied.

Optional arguments

thickness A Float specifying the thickness along the given *faces* . Either *thickness*, *topFaces*, or *bottomFaces* must be specified.

topFaces A sequence of *Face* objects whose distance to *faces* argument is used to calculate the thickness along the *faces*. The combination of *topFaces* and *bottomFaces* determines the thickness and the offset of the elements. If *bottomFaces* is not specified then the thickness is twice the distance to the *topFaces*. This argument will be ignored if *thickness* is specified. Either *thickness*, *topFaces*, or *bottomFaces* must be specified.

bottomFaces A sequence of *Face* objects whose distance to *faces* is used to calculate the thickness along the *faces*. The combination of *topFaces* and *bottomFaces* determines the thickness and the offset of the elements. If *topFaces* is not specified then the thickness is twice the distance to the *bottomFaces*.

This argument will be ignored if *thickness* is specified. Either *thickness*, *topFaces*, or *bottomFaces* must be specified.

Return value

`None.`

Exceptions

None.

backup()

This method makes a backup copy of the features in the part. Use the restore method to retrieve the part's features from the backup.

Arguments

None.

Return value

`None.`

Exceptions

None.

checkGeometry(...)

This method checks the validity of the geometry of the part and prints a count of all topological entities on the part (faces, edges, vertices, etc.).

Required arguments

None.

Optional arguments

detailed
A Boolean specifying whether detailed output will be printed to the replay file. The default value is OFF.

reportFacetErrors
A Boolean specifying whether faces are checked for proper facetting. The default value is OFF.

level
An Int specifying which level of checking is performed. Values can range from 20 to 70, with higher values reporting less and less important errors. The default value is 20, which reports all critical errors. When the default value is used, the stored validity status is updated to agree with the result of this check.

Return value

`None.`

Exceptions

None.

clearGeometryCache()

This method clears the geometry cache. Clearing the geometry cache reduces the amount of memory being used to cache part features.

Arguments

None.

Return value

`None.`

Exceptions

None.

deleteAllFeatures()

This method deletes all the features in the part.

Arguments

None.

Return value

`None.`

Exceptions

None.

deleteFeatures(...)

This method deletes the given features.

Required arguments

featureNames A sequence of Strings specifying the feature names that will be deleted from the part.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

getAngle(...)

This method returns the angle between the specified entities.

Required arguments

plane1

A *Face*, *MeshFace*, or a *Datum* object specifying the first plane. The *Datum* object must represent a datum plane. The *plane1* and *line1* arguments are mutually exclusive. One of them must be specified.

plane2

A *Face*, *MeshFace*, or a *Datum* object specifying the second plane. The *Datum* object must represent a datum plane. The *plane2* and *line2* arguments are mutually exclusive. One of them must be specified.

line1

An *Edge*, *MeshEdge*, or a *Datum* object specifying the first curve. The *Datum* object must represent a datum axis. The *plane1* and *line1* arguments are mutually exclusive. One of them must be specified.

line2

An *Edge*, *MeshEdge*, or a *Datum* object specifying the second curve. The *Datum* object must represent a datum axis. The *plane2* and *line2* arguments are mutually exclusive. One of them must be specified.

Optional arguments

commonVertex

If the two selected *Edge* objects have more than one vertex in common, this *Vertex* object specifies the vertex at which to evaluate the angle.

Return value

A Float specifying the angle between the specified entities. If you provide a plane as an argument, Abaqus/CAE computes the angle using the normal to the plane.

Exceptions

None.

getArea(...)

This method returns the total surface area of a given face or group of faces.

Required arguments

faces

A sequence of face objects whose area the method will calculate.

Optional arguments

relativeAccuracy A Float specifying that the area computation should stop when the specified relative accuracy has been achieved. The default value is 0.000001 (0.0001%).

Return value

A Float specifying the sum of the calculated areas of the given faces.

Exceptions

None.

`getAssociatedCADPaths()`

This method returns the paths to the associated CAD part and root file. These are available only if the part was imported from one of the supported CAD softwares using the Associative Import capability. The root file can be the assembly file or the part file, depending on which one was imported.

Arguments

None.

Return value

A sequence containing the path to the associated CAD part and assembly file.

Exceptions

None.

`getCADParameters()`

This method returns the names and values of the CAD parameters associated with the part. These are available only if the part was imported from one of the supported CAD softwares using the Associative Import capability, and if the parameter names defined in that CAD software are prefixed with the string ABQ.

Arguments

None.

Return value

A dictionary object representing a map of the name of the parameter and its associated value.

Exceptions

None.

`getCentroid(...)`

Depending on the arguments provided, this method returns the following:

- The location of the centroid of a given face or group of faces.

- The location of the centroid of a given cell or group of cells.

Required arguments

faces A sequence of face objects whose centroid the method will calculate. The arguments *faces* and *cells* are mutually exclusive.

cells A sequence of face objects whose centroid the method will calculate. The arguments *faces* and *cells* are mutually exclusive.

Optional arguments

relativeAccuracy A Float specifying that the centroid computation should stop when the specified relative accuracy has been achieved. The default value is 0.000001 (0.0001%).

Return value

A sequence of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the centroid.

Exceptions

None.

`getCoordinates(...)`

This method returns the coordinates of specified point.

Required arguments

entity A *Vertex*, *Datum* point, *MeshNode*, or *ReferencePoint* specifying the entity to query.

Optional arguments

csys A *DatumCsys* object specifying the desired coordinate system of the returned coordinates. By default, coordinates are given in the global coordinate system.

Return value

A tuple of 3 Floats representing the coordinates of the specified point.

Exceptions

None.

`getCurvature(...)`

This method returns the maximum curvature of a given edge or group of edges. For an arc, the curvature is constant over the entire edge, and equal to the inverse of the radius. For a straight line, the curvature is constant and equal to 0. For a spline edge, the curvature varies over a range, and this method computes the maximum.

Required arguments

edges A sequence of edge objects whose curvature the method will calculate.

Optional arguments

samplePoints An Int specifying the number of points along each edge at which the curvature will be computed. The higher the number of sample points, the better the accuracy of the computation. The default value is 100.

Return value

A Float specifying the maximum curvature.

Exceptions

None.

`getDistance(...)`

Depending on the arguments provided, this method returns one of the following:

- The distance between two points.
- The minimum distance between a point and an edge.
- The minimum distance between two edges.

Required arguments

entity1 A *Vertex*, *Datum* point, *MeshNode*, or *Edge* specifying the first entity from which to measure.

entity2 A *Vertex*, *Datum* point, *MeshNode*, or *Edge* specifying the second entity to which to measure.

Optional arguments

None.

Return value

A Float specifying the distance between *entity1* and *entity2*.

Exceptions

None.

`getLength(...)`

This method returns the length of a given edge or group of edges.

Required arguments

edges A sequence of edge objects whose total length the method will calculate.

Optional arguments

None.

Return value

A Float specifying the total length.

Exceptions

None.

`getPerimeter(...)`

This method returns the total perimeter of a given face or group of faces. All faces need to be on the same part. If the specified faces have shared edges, these edges are excluded from the computation, thus providing the length of the outer perimeter of the specified faces.

Required arguments

faces A sequence of face objects whose perimeter the method will calculate.

Optional arguments

None.

Return value

A Float specifying the perimeter.

Exceptions

None.

`getVolume(...)`

This method returns the volume area of a given cell or group of cells.

Required arguments

cells A sequence of cell objects whose volume the method will calculate.

Optional arguments

relativeAccuracy A Float specifying the relative accuracy of the computation. The default value is 0.000001 (0.0001%).

Return value

A Float specifying the sum of the areas of the given faces.

Exceptions

None.

`getMassProperties(...)`

This method returns the mass properties of a part or region. Only beams, trusses, shells, solids, point, nonstructural mass, and rotary inertia elements are supported.

Required arguments

None.

Optional arguments

regions A MeshElementArray, CellArray, FaceArray, or EdgeArray specifying the regions whose mass properties are to be queried. The whole part is queried by default.

relativeAccuracy A SymbolicConstant specifying the relative accuracy for geometry computation. Possible values are LOW, MEDIUM and HIGH. The default value is LOW.

useMesh A Boolean specifying whether the mesh should be used in the computation if the geometry is meshed. The default value is False.

specifyDensity A Boolean specifying whether a user-specified density should be used in regions with density errors such as undefined material density. The default value is False.

density A double value specifying the user-specified density value to be used in regions with density errors. The user-specified density should be greater than 0.

specifyThickness A Boolean specifying whether a user-specified thickness should be used in regions with thickness errors such as undefined thickness. The default value is False.

thickness A double value specifying the user-specified thickness value to be used in regions with thickness errors. The user-specified thickness should be greater than 0.

miAboutCenterOfMass A Boolean specifying if the moments of inertia should be evaluated about the center of mass. The default value is True.

miAboutPoint A tuple of three floats specifying the coordinates of the point about which to evaluate the moment of inertia. By default if the moments of inertia are not being evaluated about the center of mass, they will be evaluated about the origin.

Return value

A Dictionary object with the following items:

area: None or a Float specifying the sum of the area of the specified faces. The area is computed only for one side for shells.

areaCentroid: None or a tuple of three Floats representing the coordinates of the area centroid.

volume: None or a Float specifying the volume of the specified regions.

volumeCentroid: None or a tuple of three Floats representing the coordinates of the volume centroid.

massFromMassPerUnitSurfaceArea: None or a Float specifying the mass due to mass per unit surface area.

mass: None or a Float specifying the mass of the specified regions. It is the total mass and includes mass from quantities such as mass per unit surface area.

centerOfMass: None or a tuple of three Floats representing the coordinates of the center of mass.

momentOfInertia: None or a tuple of six Floats representing the moments of inertia about the center of mass or about the point specified.

warnings: A tuple of SymbolicConstants representing the problems encountered while computing the mass properties. Possible SymbolicConstants are:

UNSUPPORTED_ENTITIES: Some unsupported entities exist in the specified region. The mass properties are computed only for beams, trusses, shells, solids, point and nonstructural mass elements and rotary inertia elements. The mass properties are not computed for axisymmetric elements, springs, connectors, gaskets or any other elements.

MISSING_THICKNESS: For some regions, the section definitions are missing thickness values.

ZERO_THICKNESS: For some regions, the section definitions have a zero thickness value.

VARIABLE_THICKNESS: The nodal thickness or field thickness specified for some regions has been ignored.

NON_APPLICABLE_THICKNESS: For some regions, the thickness value is not applicable to the corresponding sections specified on the regions.

MISSING_DENSITY: For some regions, the section definitions are missing material density values.

MISSING_MATERIAL_DEFINITION: For some regions, the material definition is missing.

ZERO_DENSITY: For some regions, the section definitions have a zero material density value.

UNSUPPORTED_DENSITY: For some regions, either a negative material density or a temperature-dependent density has been specified, or the material value is missing for one or more plies in the composite section.

SHELL_OFFSETS: For shells, this method does not account for any offsets specified.

MISSING_SECTION_DEFINITION: For some regions, the section definition is missing.

UNSUPPORTED_SECTION_DEFINITION: The section definition provided for some regions is not supported.

REINFORCEMENTS: This method does not account for any reinforcements specified on the model.

SMEARED_PROPERTIES: For regions with composite section assignments, the density is smeared across the thickness. The volume centroid and center of mass computations for a composite shell use a lumped mass approach where the volume and mass is assumed to be lumped in the plane of the shell. As a result of these approximations the volume centroid, center of mass and moments of inertia might be slightly inaccurate for regions with composite section assignments.

UNSUPPORTED_NON_STRUCTURAL_MASS_ENTITIES: This method does not account for any nonstructural mass on wires.

INCORRECT_MOMENT_OF_INERTIA: For geometry regions with non-structural mass per volume, the non-structural mass is assumed to be a point mass at the centroid of the regions. Thus, the moments of inertia might be inaccurate as the distribution of the non-structural mass is not accounted for. Use the mesh for accurately computing the moments of inertia.

MISSING_BEAM_ORIENTATIONS: For some regions with beam section assignments, the beam section orientations are missing.

UNSUPPORTED_BEAM_PROFILES: This method supports the Box, Pipe, Circular, Rectangular, Hexagonal, Trapezoidal, I, L, T, Arbitrary, and Tapered beam profiles. Any other beam profile is not supported.

TAPERED_BEAM_MI: Moment of inertia calculations for tapered beams are not accurate.

SUBSTRUCTURE_INCORRECT_PROPERTIES: The user assigned density and thickness is not considered for substructures.

UNSUPPORTED_NON_STRUCTURAL_MASS_PROPORTIONAL: Nonstructural mass with Mass Proportional distribution is not supported. Results are computed using Volume Proportional distribution.

Exceptions

None.

`getFeatureFaces(...)`

This method returns a sequence of *Face* objects that are created by the given feature.

Required arguments

name A string specifying the feature name.

Optional arguments

None.

Return value

Sequence of *Face* objects.

Exceptions

An exception occurs if a feature with the given name does not exist. Error: Incorrect feature name.

`getFeatureEdges(...)`

This method returns a sequence of *Edge* objects that are created by the given feature.

Required arguments

name A string specifying the feature name.

Optional arguments

None.

Return value

Sequence of *Edge* objects.

Exceptions

An exception occurs if a feature with the given name does not exist.	Error: Incorrect feature name.

getFeatureCells(...)

This method returns a sequence of *Cell* objects that are created by the given feature.

Required arguments

name	A string specifying the feature name.

Optional arguments

None.

Return value

Sequence of *Cell* objects.

Exceptions

An exception occurs if a feature with the given name does not exist.	Error: Incorrect feature name.

getFeatureVertices(...)

This method returns a sequence of *Vertex* objects that are created by the given feature.

Required arguments

name	A string specifying the feature name.

Optional arguments

None.

Return value

Sequence of *Vertex* objects.

Exceptions

An exception occurs if a feature with the given name does not exist.	Error: Incorrect feature name.

isAlignedWithSketch()

This method checks if the normal of an analytical rigid surface part is aligned with that of its sketch.

Arguments

None.

Return value

A Boolean value of True if the part is aligned with the sketch and False if it is not aligned.

Exceptions

If the part is not an analytical rigid part:	AbaqusException: Can only be used with analytical rigid parts.

printAssignedSections()

This method prints information on each section that has been assigned to a region of the part.

Arguments

None.

Return value

`None.`

Exceptions

None.

projectEdgesOntoSketch(...)

This method projects the selected edges of a part onto the specified *ConstrainedSketch* object. The edges appear as sketch geometry after projection. If the plane of projection is not parallel to the specified edge, the resultant sketch geometry might be of a different type. For example, a circular edge can be projected as an ellipse or a line depending on the angle of the plane of projection. By default, the projected edge will be constrained to the background geometry. You can remove this constraint by setting *constrainToBackground* to False.

Required arguments

sketch	The *ConstrainedSketch* object on which the edges are projected.
edges	A sequence of candidate edges to be projected onto the sketch.

Optional arguments

constrainToBackground	A Boolean that determines whether the projected edges need to be constrained to the background geometry. The default is True.

Return value

`None.`

Exceptions

None.

projectReferencesOntoSketch(...)

This method projects the vertices of specified edges, and datum points from the part onto the specified *ConstrainedSketch* object. The vertices and datum points appear on the sketch as reference geometry.

Required arguments

sketch The *ConstrainedSketch* object on which the edges, vertices, and datum points are projected.

Optional arguments

filter A SymbolicConstant specifying how to limit the amount of projection. Possible values are ALL_EDGES and COPLANAR_EDGES. If *filter*=COPLANAR_EDGES, edges that are coplanar to the sketching plane are the only candidates for projection. The default value is ALL_EDGES.

upToFeature A Feature object specifying a marker in the feature-based history of the part. Abaqus/CAE projects onto the sketch only the part entities that were created before the feature specified by this marker. By default, part entities in features created before the sketch you are editing are candidates for projection.

edges A sequence of candidate edges whose vertices need to be projected onto the sketch. By default, all edges specified by the *filter* argument are candidates for projection.

vertices A sequence of candidate vertices to be projected onto the sketch. By default, all vertices are candidates for projection.

Return value

`None.`

Exceptions

None.

queryAttributes(...)

This method prints the following information about a part:

- the name, modeling space, and analysis type; and
- whether twist is included (available only when the modeling space is axisymmetric and the analysis type is deformable); and
- the number of vertices, edges, faces, and cells if applicable.

Required arguments

None.

Optional arguments

printResults A Boolean that specifies whether the above information is to be printed. The default value is True

Return value

A Dictionary object with string keys and integer values that returns the above information with the keys being numVertices, numEdges, numFaces, numCells, numWiredEdges, numShellFaces, and numSolidFaces.

Exceptions

None.

queryCachedStates()

This method displays the position of geometric states relative to the sequence of features in the part cache. The output is displayed in the message area.

Arguments

None.

Return value

`None.`

Exceptions

None.

queryGeometry(...)

This method prints the following information about a part:

- the name, modeling space, and analysis type;
- whether twist is included (available only when the modeling space is axisymmetric and the analysis type is deformable);
- a 3D point representing the minimum of the part's bounding box;
- a 3D point representing the maximum of the part's bounding box;
- a 3D point representing the part's centroid (only on 3D solid parts); and
- the volume (only on 3D solid parts).

Required arguments

None.

Optional arguments

relativeAccuracy A Float specifying that the property computations should stop when the specified relative accuracy has been achieved. The default value is 0.000001 (0.0001%).

printResults A Boolean that specifies whether the above information is to be printed. The default value is True.

Return value

A Dictionary object with string keys, which returns the above information with the keys being name, space, type, volume, centroid, category and boundingBox.

Exceptions

None.

queryRegionsMissingSections()

This method returns all regions in the part that do not have a section assignment but require one for analysis.

Arguments

None.

Return value

A *Region* object, or None.

Exceptions

None.

queryDisjointPlyRegions()

This method provides a list of all composite plys in the current part that have disjoint regions.

Arguments

None.

Return value

`None.`

Exceptions

None.

regenerate()

This method regenerates a part. When you modify features, it might be convenient to postpone regeneration until you make all your changes, since regeneration can be time consuming.

Arguments

None.

Return value
`None.`

Exceptions
None.

regenerationWarnings()

This method prints any regeneration warnings associated with the features.

Arguments
None.

Return value
`None.`

Exceptions
None.

removeInvalidGeometry()

Removes all invalid entities from the part, leaving a valid part. This is not recorded as a feature in the feature list, therefore it should be used on parts that have a single feature (such as an imported part).



Note:

This might remove valid entities that are connected to invalid ones. You can identify invalid entities using the query toolset before using this command.

Arguments
None.

Return value
`None.`

Exceptions
None.

restore()

This method restores the parameters of all features in the assembly to the value they had before a failed regeneration. Use the `restore` method after a failed regeneration, followed by a `regenerate` command.

Arguments
None.

Return value
`None.`

Exceptions

None.

resumeAllFeatures()

This method resumes all the suppressed features in the part.

Arguments

None.

Return value
`None.`

Exceptions

None.

resumeFeatures(...)

This method resumes the specified suppressed features in the part.

Required arguments

featureNames	A tuple of names of features that are to be resumed.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

resumeLastSetFeatures()

This method resumes the last set of features to be suppressed in the part.

Arguments

None.

Return value
`None.`

Exceptions

None.

saveGeometryCache()

This method caches the current geometry. Caching the current geometry improves regeneration performance.

Arguments

None.

Return value

`None.`

Exceptions

None.

setAssociatedCADPaths(...)

This method sets the paths to the associated CAD part and root file. This method is available only if the part was imported from one of the supported CAD softwares using the Associative Import capability. The root file can be the assembly file or the part file, depending on the one that was imported. This method can be used to specify the new paths when the CAD data is moved to a different directory.

Required arguments

None.

Optional arguments

partFile A String specifying the name of the associated CAD part file.

rootFile A String specifying the name of the root associated CAD file. This can be the same as the part file or can be the assembly file, depending on the one that was imported.

Return value

`None.`

Exceptions

None.

suppressFeatures(...)

This method suppresses the given features.

Required arguments

featureNames A tuple of names of features that are to be suppressed in the part.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

writeAcisFile(...)

This method exports the geometry of the part to a named file in ACIS format.

Required arguments

fileName A String specifying the name of the file to which to write.

Optional arguments

version A Float specifying the ACIS version. For example, the Float "12.0" corresponds to ACIS Version 12.0. The default value is the current version of ACIS.

Return value

`None.`

Exceptions

If the part is an orphan mesh part: Cannot export orphan mesh parts to ACIS.

writeCADParameters(...)

This method writes the parameters that were imported from the CAD system to a parameter file.

Required arguments

paramFile A String specifying the parameter file name.

Optional arguments

modifiedParams A tuple of tuples each containing the part name, the parameter name, and the modified parameter value. Default is an empty tuple.

updatePaths A Bool specifying whether to update the path of the CAD model file specified in the *parameterFile* to the current directory, if the CAD model is present in the current directory.

Return value
`None.`

Exceptions
None.

writeIgesFile(...)

This method exports the geometry of the part to a named file in IGES format.

Required arguments

fileName A String specifying the name of the file to which to write.

flavor A SymbolicConstant specifying a particular flavor of IGES. Possible values are STANDARD, AUTOCAD, SOLIDWORKS, JAMA, and MSBO.

Optional arguments
None.

Return value
`None.`

Exceptions
If the part is an orphan mesh part: Cannot export orphan mesh parts to IGES.

writeStepFile(...)

This method exports the geometry of the part to a named file in STEP format.

Required arguments

fileName A String specifying the name of the file to which to write.

Optional arguments
None.

Return value
`None.`

Exceptions
If the part contains no geometry: Parterror: Cannot export orphan mesh parts to STEP.

writeVdaFile(...)

This method exports the geometry of the part to a named file in VDA-FS format.

Required arguments

fileName A String specifying the name of the file to which to write.

Optional arguments

None.

Return value

`None.`

Exceptions

If the part is an orphan mesh part: Cannot export orphan mesh parts to VDA-FS.

copyMeshPattern(...)

This method copies a mesh pattern from a source region consisting of a set of shell elements or element faces onto a target face, mapping nodes and elements in a one-one correspondence between source and target.

Required arguments

None.

Optional arguments

elements A sequence of *MeshElement* objects or a Set object containing elements and specifying the source region.

faces A sequence of *Face* objects that have associated with shell elements or element faces and specifying the source region.

elemFaces A sequence of *MeshFace* objects specifying the source region.

targetFace A *MeshFace* object specifying the target region.

nodes A sequence of MeshNode objects or a Set object containing nodes on the boundary of source region that are to be positioned to the boundary of target face.

coordinates A sequence of three-dimensional coordinate tuples specifying the coordinates for each of the given nodes. When specified, the number of coordinate tuples must match the number of given nodes, and be ordered to correspond to the given nodes in *ascending order* according to index. These coordinates are positions of the nodes of a mesh that will be the target face corresponding to nodes provided.

Return value
`None.`

Exceptions

None.

`smoothNodes(...)`

This method smooths the given nodes of a native mesh, moving them locally to a more optimal location that improves the quality of the mesh

Required arguments

nodes A sequence of MeshNode objects or a Set object containing nodes.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

`Lock()`

This method locks the part. Locking the part prevents any further changes to the part that can trigger regeneration of the part.

Arguments

None.

Return value
`None.`

Exceptions

None.

`Unlock()`

This method unlocks the part. Unlocking the part allows it to be regenerated after any modifications to the part.

Arguments

None.

Return value
`None.`

Exceptions

None.

`LockForUpgrade()`

This method locks the part for upgrade. Locking the part prevents any further changes to the part that can trigger regeneration of the part. When the part is unlocked, all the parts are upgraded and regenrated.

Arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Part object has members with the same names and descriptions as the arguments to the *Part* method.

In addition, the Part object can have the following members:

allInternalSets
A repository of *Set* objects specifying picked regions.

allInternalSurfaces
A repository of *Surface* objects specifying picked regions.

allSets
A repository of *Set* objects specifying the contents of the *allSets* repository is the same as the contents of the *sets* repository.

allSurfaces
A repository of *Surface* objects specifying the contents of the *allSurfaces* repository is the same as the contents of the *surfaces* repository.

cells
A *CellArray* object specifying all the cells in the part.

compositeLayups
A repository of *CompositeLayup* objects.

datums
A repository of *Datum* objects specifying all the datums in the part.

edges
An *EdgeArray* object specifying all the edges in the part.

elemEdges
A repository of *MeshEdge* objects specifying all the element edges in the part. For a given element and a given edge index on a given face within that element, the corresponding *MeshEdge* object can be retrieved from the repository by using the key calculated as (i*32 + j*4 + k), where i, j, and k are zero-based element, face, and edge indices, respectively.

elementEdges
A *MeshEdgeArray* object specifying all the unique element edges in the part.

elementFaces
A *MeshFaceArray* object specifying all the unique element faces in the part.

elements
A *MeshElementArray* object specifying all the elements in the part.

elemFaces
A repository of *MeshFace* objects specifying all the element faces in the part. For a given element and a given face index within that element, the corresponding *MeshFace* object can be retrieved from the repository by using the key calculated as (i*8 + j), where i and j are zero-based element and face indices, respectively.

engineeringFeatures
An *EngineeringFeature* object.

faces
A *FaceArray* object specifying all the faces in the part.

features
A repository of *Feature* objects specifying all the features in the part.

featuresById
A repository of *Feature* objects specifying all *Feature* objects in the part. The *Feature* objects in the featuresById repository are the same as the *Feature* objects in the features repository. However, the key to the objects in the featuresById repository is an integer specifying the *ID*, whereas the key to the objects in the features repository is a string specifying the *name*.

geometryPrecision
A Boolean specifying the precision of the geometry of the part. A part is considered precise when the local precision of all its topologies is less than or equal to the ACIS precision of 10^{-6}.

geometryValidity
A Boolean specifying the validity of the geometry of the part. The value is computed, but it can be set to ON to perform feature and mesh operations on an invalid part. There is no guarantee that such operations will work if the part was originally invalid.

ignoredEdges
An *IgnoredEdgeArray* object specifying all the ignored edges in the part.

ignoredVertices
An *IgnoredVertexArray* object specifying all the ignored vertices in the part.

isOutOfDate
An Int specifying that feature parameters have been modified but that the part has not been regenerated. Possible values are 0 and 1.

materialOrientations
A *MaterialOrientationArray* object.

nodes
A *MeshNodeArray* object specifying all the nodes in the part.

referencePoints
A repository of *ReferencePoint* objects.

retainedNodes
A *MeshNodeArray* object specifying all the retained nodes in the substructure part.

sectionAssignments
A *SectionAssignmentArray* object.

sets
A repository of *Set* objects specifying for more information, see *Set*.

skins
A repository of *Skin* objects specifying the skins created on the part.

stringers
A repository of *Stringer* objects specifying the stringers created on the part.

surfaces
A repository of *Surface* objects specifying for more information, see *Surface*.

timeStamp
A Float specifying when the part was last modified.

vertices
A *VertexArray* object specifying all the vertices in the part.

Feature object

The following commands operate on Feature objects. For more information about the Feature object, see *Feature object*.

Access

```
import part
```

AutoRepair()

This method carries out a sequence of geometry repair operations if it contains invalid entities. It is expected to improve the geometry, but it does not guarantee that the number of invalid entities will decrease. In some cases, it can also increase the number of invalid entities. Since a number of geometry repair operations and validity checks are performed, it could be a slow operation depending on the complexity of the geometry.

Path

mdb.models[*name*].parts[*name*].AutoRepair

Arguments

None.

Return value

A Feature object.

Exceptions

None.

AddCells(...)

This method tries to convert a shell entity to a solid entity. The conversion is not always successful.

Path

mdb.models[*name*].parts[*name*].AddCells

Required arguments

faceList A sequence of Face objects specifying the faces bounding the cell to add.

Optional arguments

flipped A Boolean specifying the direction of feature creation. The possible values are True and False. The default is True indicating that the direction is opposite to the face normal. When multiple faces are selected, Abaqus attempts to create cells on both sides of the selected faces and ignores the *flipped* argument.

Return value

A Feature object.

Exceptions

None.

AnalyticRigidSurf2DPlanar(...)

This method creates a first Feature object for an analytical rigid surface by creating a planar wire from the given *ConstrainedSketch* object.

Path

mdb.models[*name*].parts[*name*].AnalyticRigidSurf2DPlanar

Required arguments

sketch A *ConstrainedSketch* object specifying the planar wire.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

AnalyticRigidSurfExtrude(...)

This method creates a first Feature object for an analytical rigid surface by extruding the given *ConstrainedSketch* object by the given depth, creating a surface.

Path

mdb.models[*name*].parts[*name*].AnalyticRigidSurfExtrude

Required arguments

sketch A *ConstrainedSketch* object specifying the planar wire.

Optional arguments

depth A Float specifying the extrusion depth. The default value is 1.0.

Return value

A Feature object.

Exceptions

None.

AnalyticRigidSurfRevolve(...)

This method creates a first Feature object for an analytical rigid surface by revolving the given *ConstrainedSketch* object by 360° about the *Y*-axis.

Path

mdb.models[*name*].parts[*name*].AnalyticRigidSurfRevolve

Required arguments

sketch A *ConstrainedSketch* object specifying the surface to be revolved.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

AssignMidsurfaceRegion(...)

This method assign a mid-surface property to sequence of *Cell* objects. If a reference representation of the part does not exist, it creates one. It also copies the *cells* to the reference representation and deletes the *cells* from the active representation of the part.

Path

mdb.models[*name*].parts[*name*].AssignMidsurfaceRegion

Required arguments

cellList A sequence of *Cell* objects specifying the regions that will be used for mid-surface construction. These regions will be copied to the reference representation of the part.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

BaseSolidExtrude(...)

This method creates a first Feature object by extruding the given *ConstrainedSketch* object by the given depth, creating a solid. The *ConstrainedSketch* object must define a closed profile.

Path

mdb.models[*name*].parts[*name*].BaseSolidExtrude

Required arguments

sketch
A *ConstrainedSketch* object specifying the plane shape to be extruded.

depth
A Float specifying the extrusion depth. Possible values are $10^{-5} \leq depth \leq 10^{5}$.

Optional arguments

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq draftAngle \leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^{5} \leq pitch \leq 10^{5}$. The arguments *draftAngle* and *pitch* are mutually exclusive.

Return value

A Feature object.

Exceptions

RangeError.

BaseSolidRevolve(...)

This method creates a first Feature object by revolving the given *ConstrainedSketch* object by the given angle, creating a solid. The *ConstrainedSketch* object must define a closed profile and an axis of revolution. The axis is defined by a single construction line.

Path

mdb.models[*name*].parts[*name*].BaseSolidRevolve

Required arguments

sketch

A *ConstrainedSketch* object specifying the shape to be revolved.

angle

A Float specifying the revolve angle in degrees. Possible values are $10^{-4} \leq angle \leq 360$.



Note:

If *pitch*>0, there is no upper limit for *angle*.

Optional arguments

pitch

A Float specifying the pitch. The pitch is the distance traveled along the axial direction, measured between corresponding points on the sketch when it has completed one full revolution about the axis of revolution. Possible values are $0 \leq pitch \leq 10^5$. The default value, 0, implies a normal revolve.

flipRevolveDirection

A Boolean specifying whether to override the direction of feature creation. If *flipRevolveDirection*=OFF, the default direction of revolution is used. If *flipRevolveDirection*=ON, the revolve direction is reversed. The default value is OFF.

flipPitchDirection

A Boolean specifying whether to override the direction of translation. If *flipPitchDirection*=OFF, the direction of translation is given by the direction of the revolve axis. If *flipPitchDirection*=ON, the translation direction is reversed. The default value is OFF.

moveSketchNormalToPath

A Boolean specifying whether to rotate the sketch so that it is normal to the path of revolution when using the *pitch* option. If *moveSketchNormalToPath*=OFF, the sketch plane remains parallel to the revolve axis. If *moveSketchNormalToPath*=ON, the sketch is moved to match the angle created by the *pitch* before being revolved. The default value is OFF.

Return value

A Feature object.

Exceptions

RangeError.

BaseSolidSweep(...)

This method creates a first Feature object by sweeping the given profile *ConstrainedSketch* object along the path defined by the path *ConstrainedSketch* object, creating a solid. The profile *ConstrainedSketch* object must define a closed profile. The origin of the profile sketch is positioned at the start of the sweep path and swept perpendicular to the path. No checks are made for self-intersection.

Path

mdb.models[*name*].parts[*name*].BaseSolidSweep

Required arguments

sketch A *ConstrainedSketch* object specifying the profile to be swept.

path A *ConstrainedSketch* object specifying the path of the sweep.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

BaseShell(...)

This method creates a first Feature object by creating a planar shell from the given *ConstrainedSketch* object. The *ConstrainedSketch* object must define a closed profile.

Path

mdb.models[*name*].parts[*name*].BaseShell

Required arguments

sketch A *ConstrainedSketch* object specifying the planar shell.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

BaseShellExtrude(...)

This method creates a first Feature object by extruding the given *ConstrainedSketch* object by the given depth, creating a shell. The *ConstrainedSketch* object can define either an open or closed profile.

Path

mdb.models[*name*].parts[*name*].BaseShellExtrude

Required arguments

sketch
A *ConstrainedSketch* object specifying the shape to be extruded.

depth
A Float specifying the extrusion depth. Possible values are Floats > 0.

Optional arguments

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 ≤*draftAngle*≤ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

Return value

A Feature object.

Exceptions

RangeError.

BaseShellRevolve(...)

This method creates a first Feature object by revolving the given *ConstrainedSketch* object by the given angle, creating a shell. The *ConstrainedSketch* object can define either an open or closed profile and an axis of revolution. The axis is defined by a single construction line.

Path

mdb.models[*name*].parts[*name*].BaseShellRevolve

Required arguments

sketch
A *ConstrainedSketch* object specifying the shape to be revolved.

angle
A Float specifying the revolve angle in degrees. Possible values are 0 ≤*angle*≤ 360.



Note:

If *pitch*> 0, there is no upper limit for *angle*.

Optional arguments

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction, measured between corresponding points on the sketch when it has completed one full revolution about the axis of revolution. Possible values are $0 \leq pitch \leq 10^5$. The default value, 0, implies a normal revolve.

flipRevolveDirection
A Boolean specifying whether to override the direction of feature creation. If *flipRevolveDirection*=OFF, the default direction of revolution is used. If *flipRevolveDirection*=ON, the revolve direction is reversed. The default value is OFF.

flipPitchDirection
A Boolean specifying whether to override the direction of translation. If *flipPitchDirection*=OFF, the direction of translation is given by the direction of the revolve axis. If *flipPitchDirection*=ON, the translation direction is reversed. The default value is OFF.

moveSketchNormalToPath
A Boolean specifying whether to rotate the sketch so that it is normal to the path of revolution when using the *pitch* option. If *moveSketchNormalToPath*=OFF, the sketch plane remains parallel to the revolve axis. If *moveSketchNormalToPath*=ON, the sketch is moved to match the angle created by the *pitch* before being revolved. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

BaseShellSweep(...)

This method creates a first Feature object by sweeping the given section *ConstrainedSketch* object along the path defined by the path *ConstrainedSketch* object, creating a shell. The *ConstrainedSketch* object can define either an open or closed profile. The origin of the profile sketch is positioned at the start of the sweep path and swept perpendicular to the path. No checks are made for self-intersection.

Path

mdb.models[*name*].parts[*name*].BaseShellSweep

Required arguments

sketch
A *ConstrainedSketch* object specifying the section to be swept.

path
A *ConstrainedSketch* object specifying the path of the sweep.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

BaseWire(...)

This method creates a first Feature object by creating a planar wire from the given *ConstrainedSketch* object.

Path

mdb.models[*name*].parts[*name*].BaseWire

Required arguments

sketch A *ConstrainedSketch* object specifying the planar wire.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

BlendFaces(...)

This method creates a Feature object by creating new faces that blends two sets of faces.

Path

mdb.models[*name*].parts[*name*].BlendFaces

Required arguments

side1 A sequence of *Edge* objects specifying one side of the blend. The edges must form a continuous chain without branches.

side2 A sequence of *Edge* or *Face* objects specifying the second side of the blend. If *side2* contains Edge objects then they must form a continuous chain without branches.

Optional arguments

method

A SymbolicConstant indicating a method for creating blends. This argument is a required argument if *side2* contains *Edge* object and it is ignored if *side2* contains *Face*objects. It can have one of the following values:

- TANGENT: The blend is tangent to the sides.
- SHORTEST_PATH: The blend connects the two sides based on linear interpolation between the two sides.
- SPECIFY_PATH: The blend connects the two sides along a specified path.

path

An Edge object that connects *side1* to *side2* and specifies the path for creating the blend. This argument is required if *method*=SPECIFY_PATH; otherwise, it is ignored.

Return value

A Feature object.

Exceptions

None.

Chamfer(...)

This method creates an additional Feature object by chamfering the given list of edges with a given length.

Path

mdb.models[*name*].parts[*name*].Chamfer

Required arguments

length

A Float specifying the length of the chamfer.

edgeList

A sequence of Edge objects specifying the edges to chamfer.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

Mirror(...)

This method mirrors existing part geometry across a plane to create new geometry.

Path

mdb.models[*name*].parts[*name*].Mirror

Required arguments

mirrorPlane	A *Datum* plane object or a planar *Face* object.
keepOriginal	A boolean specifying whether or not the original part geometry should be retained.

Optional arguments

keepInternalBoundaries	A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

ConvertToAnalytical()

This method attempts to change entities into a simpler form that will speed up processing and make entities available during feature operations.

Path

mdb.models[*name*].parts[*name*].ConvertToAnalytical

Arguments

None.

Return value

A Feature object.

Exceptions

None.

ConvertToPrecise(...)

This method attempts to change imprecise entities so that the geometry becomes precise.

Path

mdb.models[*name*].parts[*name*].ConvertToPrecise

Required arguments

None.

Optional arguments

method A SymbolicConstant specifying the method to be used to convert the part to precise. Possible values are RECOMPUTE_GEOMETRY and TIGHTEN_GAPS. The default value is RECOMPUTE_GEOMETRY.

Return value

A Feature object.

Exceptions

None.

CoverEdges(...)

This method generates a face using the given edges as the face's boundaries. The CoverEdges method generates a face by creating the geometry consisting of the underlying surface, associated edges, and vertices.

Path

mdb.models[*name*].parts[*name*].CoverEdges

Required arguments

edgeList A sequence of Edge objects specifying the edges that bound the new face.

Optional arguments

tryAnalytical A Boolean specifying whether the newly created face should be analytical or not. The default is False.

Return value

A Feature object.

Exceptions

If the given boundary is not a closed loop: Parterror: Cannot find a closed loop

If the given boundary contains a zero length component: Parterror: Cannot find a closed loop

If the underlying surface is too difficult to fit: Parterror: Cannot construct face geometry

Cut(...)

This method creates an additional Feature object by cutting a hole using the given *ConstrainedSketch* object.

Path

mdb.models[*name*].parts[*name*].Cut

Required arguments

sketchPlane
A *Datum* plane object or a planar *Face* object.

sketchPlaneSide
A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch
A *ConstrainedSketch* object specifying the planar cut.

Optional arguments

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM.

Return value

A Feature object.

Exceptions

None.

CutExtrude(...)

This method creates an additional Feature object by extruding the given *ConstrainedSketch* object by the given depth and cutting away material in the solid and shell regions of the part. The *ConstrainedSketch* object must define a closed profile. The CutExtrude method creates a blind cut (using *depth*), an up-to-face cut (using *upToFace*), or a through-all cut (if *depth* and *upToFace* are not specified).

Path

mdb.models[*name*].parts[*name*].CutExtrude

Required arguments

sketchPlane
A *Datum* plane object or a planar *Face* object.

sketchPlaneSide
A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM.

sketch
A *ConstrainedSketch* object specifying the planar sketch to be extruded.

Optional arguments

depth
A Float specifying the extrusion depth. If *depth* is specified, the cut will be a blind cut. The default is to not specify a depth.

upToFace
A Face specifying the face up to which to cut. If *upToFace* is specified, the cut will be an up-to-face cut. The default is to not specify a face.



Note:

If neither *depth* nor *upToFace* is specified, the cut will be a through-all cut.

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

flipExtrudeDirection
A Boolean specifying whether to override the direction of feature creation. If the value is OFF, it means use the direction defined by the *sketchPlaneSide*; if the value is ON, it means use the opposite direction to the one defined by *sketchPlaneSide*. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

CutLoft(...)

This method creates an additional Feature object by lofting between the given sections and cutting away material from the part. You define the sections using a sequence of edges from the part or an *EdgeArray*.

Path

mdb.models[*name*].parts[*name*].CutLoft

Required arguments

loftsections

A sequence of sequences of edges specifying the cross-sections to be lofted. Each outer sequence specifies a section through which the method will pass the loft feature. Each outer sequence can be defined as a sequence of edges or as an *EdgeArray*. The edges specifying a section must form a simple closed profile and must not contain multiple loops.

Optional arguments

startCondition

A SymbolicConstant specifying the tangent direction at the start section of the loft feature. Possible values are NONE, NORMAL, RADIAL, and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *startCondition* argument in conjunction with the *endCondition* argument.

endCondition

A SymbolicConstant specifying the tangent direction at the end section of the loft feature. Possible values are NONE, NORMAL, RADIAL, and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *endCondition* argument in conjunction with the *startCondition* argument.

startTangent

A Float specifying the angle in degrees of the tangent with respect to the plane in which the start section lies. You must specify the *startTangent* argument if *startCondition*=SPECIFIED. Possible values are $0.0 \leq startTangent \leq 180.0$.

startMagnitude

A Float specifying the magnitude of the *startTangent*. You must specify the *startMagnitude* argument if *startCondition*=SPECIFIED. Possible values are $0.0 < startMagnitude < 100.0$.

endTangent

A Float specifying the angle in degrees of the tangent with respect to the plane in which the end section lies. You must specify the *endTangent* argument if *startCondition*=SPECIFIED. Possible values are $0.0 \leq endTangent \leq 180.0$.

endMagnitude

A Float specifying the magnitude of the *endTangent*. This argument is to be used when the *endCondition* argument has the value SPECIFIED. Possible values are $0.0 < endMagnitude < 100.0$.

globalSmoothing

A Boolean specifying whether each path defined in the *paths* argument is applied locally or globally.

- If the path is applied locally, its effect is felt only on faces created from the edges on the *loftSections* through which the *paths* pass through.
- If the path is applied globally, an averaging algorithm is applied over all the paths defined and is distributed over all the faces created.

The default value is ON (globally).

Return value

A Feature object.

Exceptions

None.

CutRevolve(...)

This method creates an additional Feature object by revolving the given *ConstrainedSketch* object by the given angle and cutting away material from the part. The *ConstrainedSketch* object must define a closed profile and an axis of revolution.

Path

mdb.models[*name*].parts[*name*].CutRevolve

Required arguments

sketchPlane
A *Datum* plane object or a planar *Face* object.

sketchPlaneSide
A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM.

sketch
A *ConstrainedSketch* object specifying the planar sketch to be revolved.

angle
A Float specifying the angle in degrees to be revolved.

Optional arguments

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction, measured between corresponding points on the sketch when it has completed one full revolution about the axis of revolution. Possible values are $0 \le pitch \le 10^5$. The default value, 0, implies a normal revolve.

flipRevolveDirection
A Boolean specifying whether to override the direction of feature creation. If *flipRevolveDirection*=OFF, the default direction of revolution is used. If *flipRevolveDirection*=ON, the revolve direction is reversed. The default value is OFF.

flipPitchDirection
A Boolean specifying whether to override the direction of translation. If *flipPitchDirection*=OFF, the direction of translation is given by the direction of the revolve axis. If *flipPitchDirection*=ON, the translation direction is reversed. The default value is OFF.

moveSketchNormalToPath A Boolean specifying whether to rotate the sketch so that it is normal to the path of revolution when using the *pitch* option. If *moveSketchNormalToPath*=OFF, the sketch plane remains parallel to the revolve axis. If *moveSketchNormalToPath*=ON, the sketch is moved to match the angle created by the *pitch* before being revolved. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

CutSweep(...)

This method creates an additional Feature object by sweeping the given *ConstrainedSketch* object along a path which may be a *ConstrainedSketch* or a sequence of *Edge* objects and cutting away material from the part. If the profile section is a *ConstrainedSketch* object, it must define a closed profile. The section sketch can be created at the normal plane at the start of the sweep path or it may be created on a *Datum* plane or a planar *Face*. No checks are made for self-intersection.

Path

mdb.models[*name*].parts[*name*].CutSweep

Required arguments

path Path may either be a *ConstrainedSketch* object or a sequence of *Edge* objects specifying the path of the sweep.

profile Profile may either be a *ConstrainedSketch* object or a *Face* object specifying the section to be swept.

Optional arguments

pathPlane A *Datum* plane object or a planar *Face* object. Only required when path is a *ConstrainedSketch* object.

pathUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the path sketch. Only required when path is a *ConstrainedSketch* object.

pathOrientation A SymbolicConstant specifying the orientation of *pathUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when path is a *ConstrainedSketch* object.

sketchPlane A *Datum* plane object or a planar *Face* object specifying the plane on which to sketch the profile. Not required when profile is a *Face* object. When profile is chosen as a *ConstrainedSketch* object, user may or may not give this as input. If user does not give this as input, the normal plane at the start of the path will be the sketchPlane.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the profile sketch. Only required when profile is a *ConstrainedSketch* object.

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when profile is a *ConstrainedSketch* object.

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq$*pitch*$\leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

profileNormal
A Boolean specifying whether to keep the profile normal same as original or varying through out the sweep path. When *profileNormal*=OFF, the profile normal will vary through out the sweep path. When *profileNormal*=ON, the profile normal will be same as original through out the sweep path. The default value is OFF.

flipSweepDirection
A Boolean specifying whether to flip the direction in which sweep operation will be performed. When *flipSweepDirection*=OFF, sweep operation will be performed in the direction of path direction. When *flipSweepDirection*=ON, sweep operation will be performed in the direction opposite to the path direction. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

ExtendFaces(...)

This method extends faces along its free edges by offsetting the external edges along the surfaces. One of *distance*, *upToReferenceRep*, or *upToFaces* must be used to specify how far the faces need to be extended.

Path

mdb.models[*name*].parts[*name*].ExtendFaces

Required arguments

None.

Optional arguments

faces
A sequence of Face objects specifying the faces to be extended. The faces cannot belong to the reference representation. The *faces* and *extendAlong* arguments are mutually exclusive. One of them must be specified.

extendAlong
A sequence of Edge objects specifying the edges where to extend the faces. Only free edges are considered. The interior edges will be ignored. The *faces* and *extendAlong* arguments are mutually exclusive. One of them must be specified.

distance
A Float indicating the distance to extend the faces along the edges. Either *distance*, *upToReferenceRep*, or *upToFaces* must be specified.

upToFaces
A sequence of Face objects specifying the faces that the selected faces should be extended up to.

trimToExtendedTargetSurfaces
A Boolean indicating that the surfaces of up to target faces should be extended before extending and trimming the selected faces. The default value is True.

upToReferenceRep
A Boolean indicating that the selected faces should be extended along the selected edges and be trimmed along their intersection with the reference representation.

Return value

A Feature object.

Exceptions

None.

FaceFromElementFaces(...)

This method creates a geometry face from a collection of orphan element faces.

Path

mdb.models[*name*].parts[*name*].FaceFromElementFaces

Required arguments

elementFaces
A Region object specifying the collection of orphan element faces.

Optional arguments

stitch
A Boolean specifying whether the created geometry face should be stitched with existing geometry faces. Default value is TRUE.

stitchTolerance
A Float indicating the maximum gap to be stitched. The value should be smaller than the minimum feature size and bigger than the maximum gap expected to be stitched in the model. Otherwise this command may remove small (sliver) edges that are smaller than the

tolerance. If stitch tolerance is not provided then default value of 0.001 will be used for stitching.

analyticFitTolerance A Float indicating the analytical surface fitting tolerance. If analytical tolerance is not provided then default value of 0.015 will be used for analytical surface fitting.

associateFace A Boolean specifying whether the created geometry face should be associated with the mesh. Default value is TRUE.

Return value

A Feature object.

Exceptions

None.

HoleBlindFromEdges(...)

This method creates an additional Feature object by creating a circular blind hole of the given diameter and depth and cutting away material in the solid and shell regions of the part. The center of the hole is offset from two non-parallel straight edges by the given distances.

Path

mdb.models[*name*].parts[*name*].HoleBlindFromEdges

Required arguments

plane A *Datum* plane object or a planar *Face* object.

planeSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

diameter A Float specifying the diameter of the hole.

edge1 An Edge object specifying the edge from which *distance1* is measured.

distance1 A Float specifying the offset from *edge1*.

edge2 An Edge object specifying the edge from which *distance2* is measured.

distance2 A Float specifying the offset from *edge2*.

depth A Float specifying the depth of the hole.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

HoleFromEdges(...)

This method creates an additional Feature object by creating a circular hole of the given diameter in a 2D planar part and cutting away material in the shell and wire regions of the part. The center of the hole is offset from two non-parallel straight edges by the given distances.

Path

mdb.models[*name*].parts[*name*].HoleFromEdges

Required arguments

diameter	A Float specifying the diameter of the hole.
edge1	An Edge object specifying the edge from which *distance1* is measured.
distance1	A Float specifying the offset from *edge1*.
edge2	An Edge object specifying the edge from which *distance2* is measured.
distance2	A Float specifying the offset from *edge2*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

HoleThruAllFromEdges(...)

This method creates an additional Feature object by creating a circular through hole of the given diameter and cutting away material in the solid and shell regions of the part. The center of the hole is offset from two non-parallel straight edges by the given distances.

Path

mdb.models[*name*].parts[*name*].HoleThruAllFromEdges

Required arguments

plane A *Datum* plane object or a planar *Face* object.

planeSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

diameter A Float specifying the diameter of the hole.

edge1 An Edge object specifying the edge from which *distance1* is measured.

distance1 A Float specifying the offset from *edge1*.

edge2 An Edge object specifying the edge from which *distance2* is measured.

distance2 A Float specifying the offset from *edge2*.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

MergeEdges(...)

This method merges edges either by extending the user selection or using only the selected edges.

Path

mdb.models[*name*].parts[*name*].MergeEdges

Required arguments

None.

Optional arguments

edgeList A sequence of Edge objects specifying the edges to be merged.

extendSelection A Boolean specifying whether the user selection needs to be extended to include edges till branching occurs. Branching is said to occur when the vertex of an edge is shared by more than two edges.

Return value

A Feature object.

Exceptions

None.

OffsetFaces(...)

This method creates new faces by offsetting existing faces.

Path

mdb.models[*name*].parts[*name*].OffsetFaces

Required arguments

faceList
A sequence of Face objects specifying the faces that will be offset. The faces may belong to the part or to the reference representation associated with the part.

Optional arguments

distance
A Float indicating the distance to offset the faces. Either *distance* or *targetFaces* must be specified.

targetFaces
A sequence of Face objects whose distance to the faces argument together with the *targetFacesMethod* determines the distance to offset the faces. Either *distance* or *targetFaces* must be specified.

targetFacesMethod
A SymbolicConstant indicating how to calculate the distance to offset. It can have one of the following values:

- HALF_OF_AVERAGE: Offset the faces by a distance equals to half the average distance to target faces.
- CLOSEST_POINT_FRACTION: Offset the faces by a distance equals to the fraction of the distance to the approximate closest point on the selected target faces.
- FARTHEST_POINT_FRACTION: Offset the faces by a distance equals to the fraction of the distance to the approximate farthest point on the selected target faces.

fractionDistance
A Float indicating the fraction of the distance to the closest or the farthest point on the target faces. Its default value is 0.5.

trimToReferenceRep
A Boolean indicating whether to extend the offset faces and trim them along their intersection with the reference representation.

Return value

A Feature object.

Exceptions

None.

RemoveCells(...)

This method converts a solid entity to a shell entity.

Path

mdb.models[*name*].parts[*name*].RemoveCells

Required arguments

cellList	A sequence of Cell objects specifying the cells to remove.

Optional arguments

None.

Return value

A Boolean value.

Exceptions

If the intended volume to be turned into a shell entity is not three-dimensional.	Parterror: Geometry that is not 3-dimensional does not contain cells.

RemoveFaces(...)

This method removes faces from a solid entity or from a shell entity.

Path

mdb.models[*name*].parts[*name*].RemoveFaces

Required arguments

faceList	A sequence of Face objects specifying the faces to remove.

Optional arguments

deleteCells	A Boolean specifying whether all cells are to be deleted when the faces are removed. The default value is False.

Return value

A Feature object.

Exceptions

None.

RemoveFacesAndStitch(...)

This method removes faces from a solid entity and attempts to close the resulting gap by extending the neighboring faces of the solid.

Path

mdb.models[*name*].parts[*name*].RemoveFacesAndStitch

Required arguments

faceList A sequence of Face objects specifying the faces to remove.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

RemoveRedundantEntities(...)

This method removes redundant edges and vertices from a solid or a shell entity. One of the two arguments is required.

Path

mdb.models[*name*].parts[*name*].RemoveRedundantEntities

Required arguments

None.

Optional arguments

vertexList A sequence of Vertex objects specifying the vertices to be removed.

edgeList A sequence of Edge objects specifying the edges to be removed.

removeEdgeVertices A Boolean specifying whether the vertices of the redundant edges need to be removed. The default is True.

Return value

A Feature object.

Exceptions

If the selected entity is not a redundant entity.	Parterror: None of the selected entities are redundant.

RepairFaceNormals(...)

This method works on the entire part or a sequence of shell faces. When the entire part is selected, it aligns all the shell face normals, and inverts all of the solid faces' normals if the solid was originally inside out. When a few shell faces are selected, it inverts the normals of the selected faces.

Path

mdb.models[*name*].parts[*name*].RepairFaceNormals

Required arguments

None.

Optional arguments

faceList	A sequence of Face objects.

Return value

A Feature object.

Exceptions

None.

RepairInvalidEdges(...)

This method repairs invalid edges. It will always attempt to improve edges even if none of selected edges are initially invalid and may leave behind invalid edges that could not be repaired.

Path

mdb.models[*name*].parts[*name*].RepairInvalidEdges

Required arguments

edgeList	A sequence of Edge objects.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

RepairSliver(...)

This method repairs the selected sliver from the selected face. The sliver area is specified using two points. A face partition is carried out at the specified points and the smaller of the two faces is removed.

Path

mdb.models[*name*].parts[*name*].RepairSliver

Required arguments

face A Face object specifying the face on which the sliver is located.

point1 A point specifying the location for partition creation. It can be a Vertex object, an Interesting Point or three coordinates specifying the point on an edge of the *face*.

point2 A point specifying the location for partition creation. It can be a Vertex object, an Interesting Point or three coordinates specifying the point on an edge of the *face*.

Optional arguments

toleranceChecks A Boolean specifying whether to use internal tolerance checks to restrict the size of the sliver face being removed. The default is True.

Return value

A Feature object.

Exceptions

None.

RepairSmallEdges(...)

This method repairs small edges. This method will attempt to replace selected small edges with vertices and extend the adjacent faces and edges. This method might leave behind some small edges that cannot be removed.

Path

mdb.models[*name*].parts[*name*].RepairSmallEdges

Required arguments

edgeList A sequence of Edge objects.

Optional arguments

toleranceChecks A Boolean specifying whether to use internal tolerance checks to restrict the size of the edges being removed. The default is True.

Return value

A Feature object.

Exceptions

None.

RepairSmallFaces(...)

This method repairs small faces. It will attempt to replace the selected small faces with edges or vertices and extend the adjacent faces. This method might leave behind some small faces that cannot be removed.

Path

mdb.models[*name*].parts[*name*].RepairSmallFaces

Required arguments

faceList A sequence of Face objects.

Optional arguments

toleranceChecks A Boolean specifying whether to use internal tolerance checks to restrict the size of the faces being removed. The default is True.

Return value

A Feature object.

Exceptions

None.

ReplaceFaces(...)

This method replaces the selected faces with a single face. If one single face is selected, that alone is replaced with a new face.

Path

mdb.models[*name*].parts[*name*].ReplaceFaces

Required arguments

faceList A sequence of Face objects to be replaced.

Optional arguments

stitch A Boolean specifying whether the newly created face needs to be stitched to the existing geometry. The default is True.

Return value

A Feature object.

Exceptions

None.

Round(...)

This method creates an additional Feature object by rounding (filleting) the given list of entities with the given radius.

Path

mdb.models[*name*].parts[*name*].Round

Required arguments

radius A Float specifying the radius of the fillets.

edgeList A sequence of Edge objects. Solid and Shell edges of a part can be rounded. The operation will fail for non-manifold edges. The *edgeList* and *vertexList* arguments are mutually exclusive. One of them must be specified.

vertexList A sequence of Vertex objects. Vertices that are connected to two wire edges can be rounded. The operation will fail for a vertex connected to a face. The *edgeList* and *vertexList* arguments are mutually exclusive. One of them must be specified.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

Shell(...)

This method creates an additional Feature object by creating a planar shell from the given *ConstrainedSketch* object. The *ConstrainedSketch* object must define a closed profile.

Path

mdb.models[*name*].parts[*name*].Shell

Required arguments

sketchPlane A *Datum* plane object or a planar *Face* object.

sketchPlaneSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch A *ConstrainedSketch* object specifying the planar shell.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

None.

ShellExtrude(...)

This method creates an additional Feature object by extruding the given *ConstrainedSketch* object by the given depth, creating a shell protrusion. The *ConstrainedSketch* object can define either an open or closed profile.

Path

mdb.models[*name*].parts[*name*].ShellExtrude

Required arguments

sketchPlane A *Datum* plane object or a planar *Face* object.

sketchPlaneSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch A *ConstrainedSketch* object specifying the planar sketch to be extruded.

Optional arguments

depth

A Float specifying the extrusion depth. The default is to not specify a depth. Either *depth* or *upToFace* must be used to define the extrusion depth.

upToFace

A Face specifying the face up to which to extrude. If *upToFace* is specified, the extrusion will be an up-to-face extrusion. The default is to not specify a face. Either *depth* or *upToFace* must be used to define the extrusion depth.

sketchOrientation

A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

draftAngle

A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch

A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

flipExtrudeDirection

A Boolean specifying whether to override the direction of feature creation. If the value is OFF, it means use the direction defined by the *sketchPlaneSide*; if the value is ON, it means use the opposite direction to the one defined by *sketchPlaneSide*. The default value is OFF.

keepInternalBoundaries

A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

ShellLoft(...)

This method creates an additional Feature object by lofting between the given sections and adding shell faces to the part. You define the sections using a sequence of edges from the part or an *EdgeArray*.

Path

mdb.models[*name*].parts[*name*].ShellLoft

Required arguments

loftsections

A sequence of sequences of edges specifying the cross-sections to be lofted. Each outer sequence specifies a section through which the method will pass the loft feature. Each outer sequence can

be defined as a sequence of edges or as an *EdgeArray*. The edges specifying a section must form a simple closed profile and must not contain multiple loops.

Optional arguments

startCondition

A SymbolicConstant specifying the tangent direction at the start section of the loft feature. Possible values are NONE, NORMAL, RADIAL and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *startCondition* argument in conjunction with the *endCondition* argument.

endCondition

A SymbolicConstant specifying the tangent direction at the end section of the loft feature. Possible values are NONE, NORMAL, RADIAL and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *endCondition* argument in conjunction with the *startCondition* argument.

startTangent

A Float specifying the angle in degrees of the tangent with respect to the plane in which the start section lies. You must specify the *startTangent* argument if *startCondition*=SPECIFIED. Possible values are $0.0 \le startTangent \le 180.0$.

startMagnitude

A Float specifying the magnitude of the *startTangent*. You must specify the *startMagnitude* argument if *startCondition*=SPECIFIED. Possible values are $0.0 < startMagnitude < 100.0$.

endTangent

A Float specifying the angle in degrees of the tangent with respect to the plane in which the end section lies. You must specify the *endTangent* argument if *startCondition*=SPECIFIED. Possible values are $0.0 \le endTangent \le 180.0$.

endMagnitude

A Float specifying the magnitude of the *endTangent*. This argument is to be used when the *endCondition* argument has the value SPECIFIED. Possible values are $0.0 < endMagnitude < 100.0$.

paths

A sequence of sequences of edges that pass through each section in the loft feature. Each sequence specifies a path followed by the face or an edge created by a loft feature. Each path must start at the first section, end at the last section, and pass through each section. In addition, the order of the sequences must be the same as the order of the sections in the *loftsections* argument. Each path must not self-intersect and must be tangent continuous. In addition, the paths must not intersect each other. You cannot use the *paths* argument in conjunction with the *startCondition* and *endCondition* arguments.

globalSmoothing

A Boolean specifying whether each path defined in the *paths* argument is applied locally or globally.

- If the path is applied locally, its effect is felt only on faces created from the edges on the *loftsections* through which the *paths* pass through.
- If the path is applied globally, an averaging algorithm is applied over all the paths defined and is distributed over all the faces created.

The default value is ON (globally).

keepInternalBoundaries A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

ShellRevolve(...)

This method creates an additional Feature object by revolving the given *ConstrainedSketch* object by the given angle, creating a shell protrusion. The *ConstrainedSketch* object can define either an open or closed profile and an axis of revolution. The axis is defined by a single construction line. For a description of the plane positioning arguments, see `SolidExtrude`.

Path

mdb.models[*name*].parts[*name*].ShellRevolve

Required arguments

sketchPlane A *Datum* plane object or a planar *Face* object.

sketchPlaneSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch A *ConstrainedSketch* object specifying the planar sketch to be revolved.

angle A Float specifying the angle in degrees to be revolved.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

pitch A Float specifying the pitch. The pitch is the distance traveled along the axial direction, measured between corresponding points on the sketch when it has completed one full revolution about the axis of revolution. Possible values are $0 \le pitch \le 10^5$. The default value, 0, implies a normal revolve.

flipRevolveDirection A Boolean specifying whether to override the direction of feature creation. If *flipRevolveDirection*=OFF, the default direction of revolution is used. If *flipRevolveDirection*=ON, the revolve direction is reversed. The default value is OFF.

flipPitchDirection A Boolean specifying whether to override the direction of translation. If *flipPitchDirection*=OFF, the direction of translation is given by the direction of the revolve axis. If *flipPitchDirection*=ON, the translation direction is reversed. The default value is OFF.

moveSketchNormalToPath A Boolean specifying whether to rotate the sketch so that it is normal to the path of revolution when using the *pitch* option. If *moveSketchNormalToPath*=OFF, the sketch plane remains parallel to the revolve axis. If *moveSketchNormalToPath*=ON, the sketch is moved to match the angle created by the *pitch* before being revolved. The default value is OFF.

keepInternalBoundaries A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

ShellSweep(...)

This method creates an additional Feature object by sweeping the given *ConstrainedSketch* object or a sequence of *Edge* objects along a path which may be a *ConstrainedSketch* or a sequence of *Edge* objects, creating a shell swept protrusion. The section can be an open or a closed profile. The section sketch can be created at the normal plane at the start of the sweep path or it may be created on a *Datum* plane or a planar *Face*. No checks are made for self-intersection.

Path

mdb.models[*name*].parts[*name*].ShellSweep

Required arguments

path Path may either be a *ConstrainedSketch* object or a sequence of *Edge* objects specifying the path of the sweep.

profile Profile may either be a *ConstrainedSketch* object or a sequence of *Edge* objects specifying the section to be swept.

Optional arguments

pathPlane A *Datum* plane object or a planar *Face* object. Only required when path is a *ConstrainedSketch* object.

pathUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the path sketch. Only required when path is a *ConstrainedSketch* object.

pathOrientation
A SymbolicConstant specifying the orientation of *pathUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when path is a *ConstrainedSketch* object.

sketchPlane
A *Datum* plane object or a planar *Face* object specifying the plane on which to sketch the profile. Not required when profile is a *Face* object. When profile is chosen as *ConstrainedSketch* object, user may or may not give this as input. If user does not give this as input, the normal plane at the start of the path will be the sketchPlane.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the profile sketch. Only required when profile is a *ConstrainedSketch* object.

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when profile is a *ConstrainedSketch* object.

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

profileNormal
A Boolean specifying whether to keep the profile normal same as original or varying through out the sweep path. When *profileNormal*=OFF, the profile normal will vary through out the sweep path. When *profileNormal*=ON, the profile normal will be same as original through out the sweep path. The default value is OFF.

flipSweepDirection
A Boolean specifying whether to flip the direction in which sweep operation will be performed. When *flipSweepDirection*=OFF, sweep operation will be performed in the direction of path direction. When *flipSweepDirection*=ON, sweep operation will be performed in the direction opposite to the path direction. The default value is OFF.

keepInternalBoundaries
A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

SolidExtrude(...)

This method creates an additional Feature object by extruding the given *ConstrainedSketch* object by the given depth, creating a solid protrusion. The *ConstrainedSketch* object must define a closed profile.

Path

mdb.models[*name*].parts[*name*].SolidExtrude

Required arguments

sketchPlane
A *Datum* plane object or a planar *Face* object.

sketchPlaneSide
A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch
A *ConstrainedSketch* object specifying the planar sketch to be extruded.

Optional arguments

depth
A Float specifying the extrusion depth. The default is to not specify a depth. Either *depth* or *upToFace* must be used to define the extrusion depth.

upToFace
A Face specifying the face up to which to extrude. If *upToFace* is specified, the extrusion will be an up-to-face extrusion. The default is to not specify a face. Either *depth* or *upToFace* must be used to define the extrusion depth.

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

draftAngle
A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

flipExtrudeDirection
A Boolean specifying whether to override the direction of feature creation. If the value is OFF, it means use the direction defined by the *sketchPlaneSide*; if the value is ON, it means use the opposite direction to the one defined by *sketchPlaneSide*. The default value is OFF.

keepInternalBoundaries A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

SolidLoft(...)

This method creates an additional Feature object by lofting between the given sections and adding material to the part. You define the sections using a sequence of edges from the part or an *EdgeArray*.

Path

mdb.models[*name*].parts[*name*].SolidLoft

Required arguments

loftsections A sequence of sequences of edges specifying the cross-sections to be lofted. Each outer sequence specifies a section through which Abaqus will pass the loft feature. Each outer sequence can be defined as a sequence of edges or as an *EdgeArray*. The edges specifying a section must form a simple closed profile and must not contain multiple loops.

Optional arguments

startCondition A SymbolicConstant specifying the tangent direction at the start section of the loft feature. Possible values are NONE, NORMAL, RADIAL and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *startCondition* argument in conjunction with the *endCondition* argument.

endCondition A SymbolicConstant specifying the tangent direction at the end section of the loft feature. Possible values are NONE, NORMAL, RADIAL and SPECIFIED. You can specify this argument only if the start and end sections are planar. You cannot use this argument in conjunction with the *path* argument. You must use the *endCondition* argument in conjunction with the *startCondition* argument.

startTangent A Float specifying the angle in degrees of the tangent with respect to the plane in which the start section lies. You must specify the *startTangent* argument if *startCondition*=SPECIFIED. Possible values are $0.0 \leq startTangent \leq 180.0$.

startMagnitude A Float specifying the magnitude of the *startTangent*. You must specify the *startMagnitude* argument if *startCondition*=SPECIFIED. Possible values are $0.0 < startMagnitude < 100.0$.

endTangent
A Float specifying the angle in degrees of the tangent with respect to the plane in which the end section lies. You must specify the *endTangent* argument if *startCondition*=SPECIFIED. Possible values are 0.0 ≤*endTangent*≤ 180.0.

endMagnitude
A Float specifying the magnitude of the *endTangent*. This argument is to be used when the *endCondition* argument has the value SPECIFIED. Possible values are 0.0 <*endMagnitude*< 100.0.

paths
A sequence of sequences of edges that pass through each section in the loft feature. Each sequence specifies a path followed by the face or an edge created by a loft feature. Each path must start at the first section, end at the last section, and pass through each section. In addition, the order of the sequences must be the same as the order of the sections in the *loftsections* argument. Each path must not self-intersect and must be tangent continuous. In addition, the paths must not intersect each other. You cannot use the *paths* argument in conjunction with the *startCondition* and *endCondition* arguments.

globalSmoothing
A Boolean specifying whether each path defined in the *paths* argument is applied locally or globally.

- If the path is applied locally, its effect is felt only on faces created from the edges on the *loftsections* through which the *paths* pass through.
- If the path is applied globally, an averaging algorithm is applied over all the paths defined and is distributed over all the faces created.

The default value is ON (globally).

keepInternalBoundaries
A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

SolidRevolve(...)

This method creates an additional Feature object by revolving the given *ConstrainedSketch* object by the given angle, creating a solid protrusion. The *ConstrainedSketch* object must define a closed profile and an axis of revolution. The axis is defined by a single construction line.

Path

mdb.models[*name*].parts[*name*].SolidRevolve

Required arguments

sketchPlane
A *Datum* plane object or a planar *Face* object.

sketchPlaneSide
A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge
An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch
A *ConstrainedSketch* object specifying the planar sketch to be revolved.

angle
A Float specifying the angle in degrees to be revolved.

Optional arguments

sketchOrientation
A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

pitch
A Float specifying the pitch. The pitch is the distance traveled along the axial direction, measured between corresponding points on the sketch when it has completed one full revolution about the axis of revolution. Possible values are $0 \le pitch \le 10^5$. The default value, 0, implies a normal revolve.

flipRevolveDirection
A Boolean specifying whether to override the direction of feature creation. If *flipRevolveDirection*=OFF, the default direction of revolution is used. If *flipRevolveDirection*=ON, the revolve direction is reversed. The default value is OFF.

flipPitchDirection
A Boolean specifying whether to override the direction of translation. If *flipPitchDirection*=OFF, the direction of translation is given by the direction of the revolve axis. If *flipPitchDirection*=ON, the translation direction is reversed. The default value is OFF.

moveSketchNormalToPath
A Boolean specifying whether to rotate the sketch so that it is normal to the path of revolution when using the *pitch* option. If *moveSketchNormalToPath*=OFF, the sketch plane remains parallel to the revolve axis. If *moveSketchNormalToPath*=ON, the sketch is moved to match the angle created by the *pitch* before being revolved. The default value is OFF.

keepInternalBoundaries
A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

SolidSweep(...)

This method creates an additional Feature object by sweeping the given *ConstrainedSketch* object or a *Face* object along a path which may be a *ConstrainedSketch* or a sequence of *Edge* objects, creating a solid swept protrusion. If the profile section is a *ConstrainedSketch* object, it must define a closed profile. The section sketch can be created at the normal plane at the start of the sweep path or it may be created on a *Datum* plane or a planar *Face*. No checks are made for self-intersection.

Path

mdb.models[*name*].parts[*name*].SolidSweep

Required arguments

path

Path may either be a *ConstrainedSketch* object or a sequence of *Edge* objects specifying the path of the sweep.

profile

Profile may either be a *ConstrainedSketch* object or a *Face* object specifying the section to be swept.

Optional arguments

pathPlane

A *Datum* plane object or a planar *Face* object. Only required when path is a *ConstrainedSketch* object.

pathUpEdge

An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the path sketch. Only required when path is a *ConstrainedSketch* object.

pathOrientation

A SymbolicConstant specifying the orientation of *pathUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when path is a *ConstrainedSketch* object.

sketchPlane

A *Datum* plane object or a planar *Face* object specifying the plane on which to sketch the profile. Not required when profile is a *Face* object. When profile is chosen as *ConstrainedSketch* object, user may or may not give this as input. If user does not give this as input, the normal plane at the start of the path will be the sketchPlane.

sketchUpEdge

An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the profile sketch. Only required when profile is a *ConstrainedSketch* object.

sketchOrientation

A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. Default value is RIGHT. Only required when profile is a *ConstrainedSketch* object.

draftAngle

A Float specifying the draft angle in degrees. Possible values are -90.0 $\leq$*draftAngle*$\leq$ 90.0. By convention, for a positive draft angle an outer loop will draft outward and an inner loop will draft inward. The opposite is true for a negative draft angle. The default value, 0, implies a normal extrude. The arguments *draftAngle* and *pitch* are mutually exclusive.

pitch A Float specifying the pitch. The pitch is the distance traveled along the axial direction by the sketch when the sketch has completed one full revolution about the twist axis. Pitch can be specified as positive or negative to achieve right-handed or left-handed twist about the twist axis, respectively. The default value, 0, implies a normal extrude. Possible values are $-10^5 \leq pitch \leq 10^5$. The arguments *draftAngle* and *pitch* are mutually exclusive.

profileNormal A Boolean specifying whether to keep the profile normal same as original or varying through out the sweep path. When *profileNormal*=OFF, the profile normal will vary through out the sweep path. When *profileNormal*=ON, the profile normal will be same as original through out the sweep path. The default value is OFF.

flipSweepDirection A Boolean specifying whether to flip the direction in which sweep operation will be performed. When *flipSweepDirection*=OFF, sweep operation will be performed in the direction of path direction. When *flipSweepDirection*=ON, sweep operation will be performed in the direction opposite to the path direction. The default value is OFF.

keepInternalBoundaries A Boolean specifying whether internal boundaries will be retained. The default value is OFF.

Return value

A Feature object.

Exceptions

None.

Stitch(...)

This method attempts to create a valid part by binding together free and imprecise edges of all the faces of a part. If *edgeList* is not given, a global stitch will be performed. If *stitchTolerance* is not specified, a value of 1.0 will be used.

Path

mdb.models[*name*].parts[*name*].Stitch

Required arguments

None.

Optional arguments

edgeList A sequence of Edge objects specifying the edges that need to be stitched.

stitchTolerance A Float indicating the maximum gap to be stitched. The value should be smaller than the minimum feature size and bigger than the maximum gap expected to be stitched in the model. Otherwise this command may remove small (sliver) edges that are smaller than the tolerance.

Return value

A Feature object.

Exceptions

None.

Wire(...)

This method creates an additional Feature object by creating a planar wire from the given *ConstrainedSketch* object. The *ConstrainedSketch* object must define a closed profile.

Path

mdb.models[*name*].parts[*name*].Wire

Required arguments

sketchPlane A *Datum* plane object or a planar *Face* object specifying the plane on which to sketch.

sketchPlaneSide A SymbolicConstant specifying the direction of feature creation. Possible values are SIDE1 and SIDE2.

sketchUpEdge An Edge object or a Datum axis object specifying the vertical (*Y*) direction of the sketch.

sketch A *ConstrainedSketch* object specifying the planar sketch to be revolved.

Optional arguments

sketchOrientation A SymbolicConstant specifying the orientation of *sketchUpEdge* on the sketch. Possible values are RIGHT, LEFT, TOP, and BOTTOM. The default value is RIGHT.

Return value

A Feature object.

Exceptions

None.

WireSpline(...)

This method creates an additional Feature object by creating a spline wire that passes through a sequence of given points. Each point can be a datum point, a vertex, an interesting point, or a tuple.

Path

mdb.models[*name*].parts[*name*].WireSpline

Required arguments

points
A sequence of Vertex, Datum point, or InterestingPoint objects specifying the points through which the spline wire will pass. *points* can also be a sequence of tuples of Floats. You must specify at least two values in the sequence.

Optional arguments

mergeType
A SymbolicConstant specifying the merge behavior of the wire with existing geometry. If *mergeType* is MERGE, Abaqus merges the wire into solid regions of the part if the wire passes through them. If *mergeType* is IMPRINT, Abaqus imprints the spline wire on existing geometry as edges. If *mergeType* is SEPARATE, Abaqus neither merges nor imprints the spline wire with existing geometry. It creates the wire separately. The default value is IMPRINT.

smoothClosedSpline
A Boolean specifying the behavior of Abaqus when the points defining a spline wire form a closed loop (the start and end points are the same). If *smoothClosedSpline*=ON, Abaqus creates a smooth spline wire where the tangencies at the end point meet smoothly. If *smoothClosedSpline*=OFF, Abaqus does not automatically create a smooth end condition. The default value in OFF.

Return value

A Feature object.

Exceptions

None.

WirePolyLine(...)

This method creates an additional Feature object by creating a polyline wire that passes through a sequence of given points. Each point can be a datum point, a vertex, an interesting point, or a tuple.

Path

mdb.models[*name*].parts[*name*].WirePolyLine

Required arguments

points
A sequence of Vertex, Datum point, or InterestingPoint objects specifying the points through which the polyline wire will pass. *points* can also be a sequence of tuples of Floats. You must specify at least two values in the sequence.

Optional arguments

mergeType
A SymbolicConstant specifying the merge behavior of the wire with existing geometry. If *mergeType* is MERGE, Abaqus merges the wire into solid regions of the part if the wire passes through them. If *mergeType* is IMPRINT, Abaqus imprints the wire on existing geometry as edges. If *mergeType* is SEPARATE, Abaqus neither merges nor imprints the spline wire with existing geometry. It creates the wire separately. The default value is IMPRINT.

meshable A Boolean specifying whether the wire should be available for selection in meshing operations. If *meshable*=OFF, the wire can be used for connector section assignment. The default value is ON.

Return value

A Feature object.

Exceptions

None.

WireFromEdge(...)

This method creates an additional Feature object by creating a Wire by selecting one or more *Edge* objects of a Solid or Shell part.

Path

mdb.models[*name*].parts[*name*].WireFromEdge

Required arguments

edgeList A list of *Edge* objects specifying the edges from which the wire is to be created.

Optional arguments

None.

Return value

A Feature object.

Exceptions

None.

AcisFile object

The AcisFile object is a file object used to open ACIS-, STEP-, and IGES-format files.

Access

```
import part
```

openAcis(...)

This method creates an AcisFile object from a file containing ACIS-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openAcis

Required arguments

fileName A String specifying the path to the ACIS file to open.

Optional arguments

scaleFromFile A Boolean specifying whether to scale, rotate, and translate the part using the transform read from the ACIS file. The default value is OFF.

Return value

An AcisFile object.

Exceptions

File is from a newer version of ACIS than the CAE kernel. Texterror: ACIS File version exceeds Kernel.

The data in the ACIS file are corrupted. Texterror: Failed to read ACIS file.

openCatia(...)

This method creates an AcisFile object from a file containing V5-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openCatia

Required arguments

fileName A String specifying the path to the CATIA file to open.

Optional arguments

topology
A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID , SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID .

convertUnits
A SymbolicConstant specifying whether the original units should be retained. Possible values are ON and OFF. The default value is OFF.

combineBodies
A Boolean specifying whether to combine the bodies in the CATPart file. If the bodies to be combined touch or overlap, invalid entities would result. For CATProduct files, this option will be ignored.

Return value

An AcisFile object.

Exceptions

None.

openEnf(...)

This method creates an AcisFile object from a file containing Elysium Neutral File-format geometry that was created by CATIA V5, I-DEAS, or Pro/ENGINEER. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openEnf

Required arguments

fileName
A String specifying the path to the Elysium Neutral File that was created by I-DEAS, Pro/ENGINEER, or CATIA V5.

fileType
A String specifying the type of CAD system that created the file. Possible values are “ideas”, “proe”, or “catiav5” or a combination similar to “proe/ideas/catiav5” if the type is unknown.

Optional arguments

topology
A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID , SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

convertUnits
A Boolean specifying if the dimensions of the part should be converted to millimeters. The default value is OFF.

Return value

An AcisFile object.

Exceptions

None.

openIges(...)

This method creates an AcisFile object from a file containing IGES-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openIges

Required arguments

fileName A String specifying the path to the IGES file to open.

Optional arguments

trimCurve A SymbolicConstant specifying the method used to define the trim curves that bound parametric surfaces. Possible values are:

- DEFAULT, use either of the following as specified by the contents of the IGES file.
- PARAMETRIC_DATA, use the parameter space of the surface being trimmed.
- THREED_DATA, use real space—the coordinate system of the part along with an indication that the trim curve lies on the parametric surface.

The default value is DEFAULT.

scaleFromFile A SymbolicConstant specifying whether the imported geometry needs to be scaled using the units information available in the IGES file. Possible values are ON and OFF. The default value is OFF. When the argument is set to ON, the geometry is scaled to millimeters with respect to the unit system specified in the IGES file.

msbo A Boolean specifying if the IGES file contains MSBO (Manifold Solid B-Rep Object) entities. The default value is False.

includedLayers A sequence of Ints specifying the levels or layers of entities that will be translated from the IGES file to build the part. The default is to include all the layers.

topology A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID , SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

uniteWires A SymbolicConstant specifying whether the imported wires need to be united or not. Possible values are ON and OFF. The default value is ON. When importing a sketch, this value is set to OFF.

Return value

An AcisFile object.

Exceptions

The data in the IGES file are corrupted. Texterror: Failed to read IGES file.

openParasolid(...)

This method creates an AcisFile object from a file containing Parasolid-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openParasolid

Required arguments

fileName A String specifying the path to the Parasolid file to open.

Optional arguments

topology A SymbolicConstant specifying the topology of the data to be read from the file and of the part to be created. Possible values are SOLID , SHELL, and WIRE. If *topology*=SOLID, Abaqus/CAE attempts to attach cells to create a solid. If *topology*=SHELL, Abaqus/CAE builds the body as a shell entity and not as a solid entity. The default value is SOLID.

Return value

An AcisFile object.

Exceptions

None.

openStep(...)

This method creates an AcisFile object from a file containing STEP-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openStep

Required arguments

fileName A String specifying the path to the STEP file to open.

Optional arguments

scale A Float specifying the scaling factor to apply to the imported geometric entities. The default value is 1.0.

Return value

An AcisFile object.

Exceptions

The data in the STEP file are corrupted. Texterror: Failed to read STEP file.

openVda(...)

This method creates an AcisFile object from a file containing VDA-FS-format geometry. This object is subsequently used by the `PartFromGeometryFile` method.

Path

mdb.openVda

Required arguments

fileName A String specifying the path to the VDA-FS file to open.

Optional arguments

None.

Return value

An AcisFile object.

Exceptions

The data in the VDA-FS file are corrupted. Texterror: Failed to read VDA file.

writeAcisFile(...)

This method exports the assembly to a named file in ACIS format.

Required arguments

fileName A String specifying the name of the file to which to write.

Optional arguments

version A Float specifying the ACIS version. For example, the Float "12.0" corresponds to ACIS Version 12.0. The default value is the current version of ACIS.

Return value

`None.`

Exceptions

None.

Members

The AcisFile object has the following member:

numberOfParts An Int specifying the number of parts in the object.

Partition commands

The partition commands are used to partition edges, faces, and cells into new regions.
A partition command can be invoked for a Part object or for an Assembly object. The partition commands create Feature objects and are described in *Feature object*.

Path and Probe commands

Path commands are used to define a line through your model by specifying a series of points. You can view results along the path in the form of an *X*–*Y* plot. Probe commands are used to determine model data and analysis results at selected locations.

In this section:

- *Path object*
- *CurrentProbeValues object*
- *FreeBody object*
- *NodeQuery object*
- *ProbeOptions object*
- *ProbeReport object*
- *SelectedProbeValues object*
- *Spectrum object*
- *Stream object*

Path object

The Path object defines a line through your model by specifying a series of nodes or points.

Access

```
import visualization
session.paths[name]
```

Path(...)

This method creates a Path object.

Path

```
session.Path
```

Required arguments

name

A String specifying the repository key.

type

A SymbolicConstant specifying the type of path being created. Possible values are NODE_LIST, POINT_LIST, EDGE_LIST, CIRCUMFERENTIAL, and RADIAL.

expression

A sequence specifying the nodes or points that make up the path. The definition of the path expression depends on the *type* argument.

- If *type*=NODE_LIST, *expression* must be a sequence of sequences. Each inner sequence contains two items, the first item is a String specifying the name of a part instance, and the second item can be either a sequence of Ints or a sequence of Strings, each specifying a range of Ints.
- If *type*=POINT_LIST, *expression* must be a sequence of tuples of three Floats, specifying the coordinates of each point.
- If *type*=EDGE_LIST, *expression* must be a sequence of sequences. Each inner sequence contains two items, the first item is a String specifying the name of the part instance, and the second item is a sequence of tuples of four Ints that uniquely identify an element edge. The four Ints are:
  1. The element label.
  2. The element face index (one-based).
  3. The face edge index (one-based).
  4. The edge direction. A positive number specifies that the edge direction runs from the edge start node to the edge end node. A negative number specifies the opposite.
- When *type*=CIRCUMFERENTIAL or RADIAL, *expression* must be a sequence of three tuples of three Floats, specifying the coordinates of the points used to define a coordinate system.

circleDefinition

A SymbolicConstant specifying the method in which the circle is being defined. This argument is valid only when *type*=CIRCUMFERENTIAL or RADIAL. Possible values are ORIGIN_AXIS and POINT_ARC.

When the value is ORIGIN_AXIS, the first two points in *expression* are points on the rotational axis and the third point lies on the x-axis. When the value is POINT_ARC, the three points in *expression* are points lying on the arc of the circle.

numSegments
An Int specifying the number of equal segments in the path. This argument is valid only when *type*=CIRCUMFERENTIAL or RADIAL.

startAngle
A Float specifying the start angle of the circumferential path. This argument is valid only when *type*=CIRCUMFERENTIAL.

endAngle
A Float specifying the end angle of the circumferential path. This argument is valid only when *type*=CIRCUMFERENTIAL.

radius
The SymbolicConstant CIRCLE_RADIUS or a Float specifying the radius of the circumferential path. This argument is valid only when *type*=CIRCUMFERENTIAL.

radialAngle
A Float specifying the angle between the radial path and the *X*-axis of the specified coordinate system. This argument is valid only when *type*=RADIAL.

startRadius
The SymbolicConstant CIRCLE_RADIUS or a Float specifying the start radius of the radial path. This argument is valid only when *type*=RADIAL.

endRadius
The SymbolicConstant CIRCLE_RADIUS or a Float specifying the end radius of the radial path. This argument is valid only when *type*=RADIAL.

Optional arguments

None.

Return value

A Path object.

Exceptions

KeyError and PthError.

Possible error messages for PthError exception: ModelError, ErrorUnsupportedNodeData, ErrorUnsupportedPointData, ErrorIncorrectPathData, KeyError, ErrorEmptyPathName, ErrorPathNotFound, and ErrorNoOdbPathDisplay.

ValueError: When *type*=CIRCUMFERENTIAL or RADIAL, the three points specified in *expression* are collinear.

Members

The Path object has members with the same names and descriptions as the arguments to the *Path* method.

In addition, the Path object can have the following member:

coordinates
If *type*=NODE_LIST, *coordinates* is a sequence of tuples of three Floats. This can be used as the value for the *expression* argument when creating a Path object of *type* = POINT_LIST.

CurrentProbeValues object

The CurrentProbeValues object has no constructor. The CurrentProbeValues object is created when you import the Visualization module.

Access

```
import visualization
session.currentProbeValues
```

Members

The CurrentProbeValues object has the following member:

values A tuple of Floats specifying the values obtained while probing. These values are updated constantly as the user moves the mouse over the object being probed.

FreeBody object

The FreeBody object defines a section across which resultant forces and moments are computed.

Access

```
import visualization
session.freeBodies[name]
```

FreeBodyFromEdges(...)

This method creates a FreeBody object and places it in the freeBodies repository.

Path

```
session.FreeBodyFromEdges
```

Required arguments

name A string name for the free body.

edges A *DisplayGroup* leaf object that specifies the physical constituents of the free body.

Optional arguments

summationLoc A SymbolicConstant specifying the location of the summation point. Possible values are CENTROID, NODAL_AVERAGE, and SPECIFY. The default value is CENTROID.

summationPoint A tuple of 3 floats specifying the summation point.

componentResolution A SymbolicConstant specifying the component resolution. Possible values are NORMAL_TANGENTIAL and CSYS. The default value is NORMAL_TANGENTIAL.

csysName A string specifying the name of the coordinate system.

Return value

A FreeBody object.

Exceptions

None.

FreeBodyFromFaces(...)

This method creates a FreeBody object and places it in the freeBodies repository.

Path

```
session.FreeBodyFromFaces
```

Required arguments

name
A string name for the free body.

faces
A *DisplayGroup* leaf object that specifies the physical constituents of the free body.

Optional arguments

summationLoc
A SymbolicConstant specifying the location of the summation point. Possible values are CENTROID, NODAL_AVERAGE, and SPECIFY. The default value is CENTROID.

summationPoint
A tuple of 3 floats specifying the summation point.

componentResolution
A SymbolicConstant specifying the component resolution. Possible values are NORMAL_TANGENTIAL and CSYS. The default value is NORMAL_TANGENTIAL.

csysName
A string specifying the name of the coordinate system.

Return value

A FreeBody object.

Exceptions

None.

FreeBodyFromNodesElements(...)

This method creates a FreeBody object and places it in the freeBodies repository.

Path

```
session.FreeBodyFromNodesElements
```

Required arguments

name
A string name for the free body.

elements
A *DisplayGroup* leaf object that specifies the physical constituents of the free body.

nodes
A *DisplayGroup* leaf object that specifies the physical constituents of the free body.

Optional arguments

summationLoc
A SymbolicConstant specifying the location of the summation point. Possible values are CENTROID, NODAL_AVERAGE, and SPECIFY. The default value is CENTROID.

summationPoint	A tuple of 3 floats specifying the summation point.
componentResolution	A SymbolicConstant specifying the component resolution. Possible values are NORMAL_TANGENTIAL and CSYS. The default value is NORMAL_TANGENTIAL.
csysName	A string specifying the name of the coordinate system.

Return value

A FreeBody object.

Exceptions

None.

getFreeBodyData(...)

This method returns the force and moment data of a FreeBody object.

Path

```
session.freeBodies[name].getFreeBodyData
```

Required arguments

None.

Optional arguments

step	An Int specifying the step from which to obtain values. The default value is the current step.
frame	An Int specifying the frame from which to obtain values. The default value is the current frame.
allActiveStepFrame	A Boolean specifying whether to obtain the values from the specified step and frame or from all active steps and frames.

Return value

A tuple of tuples of dictionaries, for each requested step and frame, containing the force and moment data.

Exceptions

None.

Members

The FreeBody object has no members.

NodeQuery object

The NodeQuery object specifies nodes and their coordinates in a path. The NodeQuery object has no constructor or methods. Abaqus creates the *nodeQuery* member when you import the `visualization` module.

Access

```
import visualization
session.nodeQuery
```

Members

The NodeQuery object has the following members:

nodeId

An Int specifying the ID of the most recently queried node. If the last query was unsuccessful, *nodeID*=−1.

nodePos

A tuple of Floats specifying the *X*-, *Y*-, and *Z*-coordinates of the most recently queried node.

ProbeOptions object

The ProbeOptions object is used to store settings associated with probing a model or an *X–Y* plot. The ProbeOptions object has no constructor. Abaqus creates the *defaultProbeOptions* and the *probeOptions* members when you import the Visualization module. When probing is initiated for the first time, the values in the *probeOptions* member are initialized using the values from the *defaultProbeOptions* member.

Access

```
import visualization
session.defaultProbeOptions
session.probeOptions
```

`setValues(...)`

This method modifies the settings on the ProbeOptions object.

Required arguments

None.

Optional arguments

options

A ProbeOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

probeEntity

A SymbolicConstant specifying the entity being probed when *probeObject*="ODB". Possible values are NODE and ELEMENT. The default value is ELEMENT.

probeOutputPosition

A SymbolicConstant specifying the output position to probe for field output results when *probeObject*="ODB". Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, and ELEMENT_CENTROID.

When *probeEntity*=NODE, the only possible value is NODAL. When *probeEntity*=ELEMENT, the only possible values are INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, and ELEMENT_CENTROID, and the default is INTEGRATION_POINT.

partInstance

A Boolean specifying whether to display the part instance information. This member is valid when *probeObject*=“ODB.” The default value is ON.

elementID

A Boolean specifying whether to display the element ID information. This member is valid when *probeObject*=“ODB” and *probeEntity*=ELEMENT. The default value is ON.

elementType

A Boolean specifying whether to display the element type information. This member is valid when *probeObject*=“ODB” and *probeEntity*=ELEMENT. The default value is ON.

elementConnectivity

A Boolean specifying whether to display the element connectivity. This member is valid when *probeObject*=“ODB” and *probeEntity*=ELEMENT. The default value is ON.

elementFieldResults A Boolean specifying whether to display the element field output results. This member is valid when *probeObject*="ODB", *probeEntity*=ELEMENT, and *isFieldOutputAvailable*=ON. The default value is ON.

nodeId A Boolean specifying whether to display the node ID when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

baseCoordinates A Boolean specifying whether to display the base coordinates of a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

deformedCoordinates A Boolean specifying whether to display the deformed coordinates of a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

attachedElements A Boolean specifying whether to display the elements attached to a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

nodeFieldResults A Boolean specifying whether to display the node field output results. This member is valid when *probeObject*="ODB", *probeEntity*=NODE, and *isFieldOutputAvailable*=ON. The default value is ON.

legend A Boolean specifying whether to display the legend for a curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

xValue A Boolean specifying whether to display the *x*-coordinate value of the point on the curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

yValue A Boolean specifying whether to display the *y*-coordinate value of the point on the curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

sequenceID A Boolean specifying whether to display the sequence ID of the point on the curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

interpolateXy A Boolean specifying whether to interpolate values within a line segment when *probeObject*="XYPlot." When *interpolateXy*=OFF, probing returns the nearest *X–Y* data point on the curve. When *interpolateXy*=ON, probing interpolates data to return a value at the nearest point on the curve. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The ProbeOptions object has the following members:

probeEntity

A SymbolicConstant specifying the entity being probed when *probeObject*="ODB". Possible values are NODE and ELEMENT. The default value is ELEMENT.

probeOutputPosition

A SymbolicConstant specifying the output position to probe for field output results when *probeObject*="ODB". Possible values are NODAL, INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, and ELEMENT_CENTROID.

When *probeEntity*=NODE, the only possible value is NODAL. When *probeEntity*=ELEMENT, the only possible values are INTEGRATION_POINT, ELEMENT_FACE, ELEMENT_NODAL, and ELEMENT_CENTROID, and the default is INTEGRATION_POINT.

partInstance

A Boolean specifying whether to display the part instance information. This member is valid when *probeObject*="ODB." The default value is ON.

elementID

A Boolean specifying whether to display the element ID information. This member is valid when *probeObject*="ODB" and *probeEntity*=ELEMENT. The default value is ON.

elementType

A Boolean specifying whether to display the element type information. This member is valid when *probeObject*="ODB" and *probeEntity*=ELEMENT. The default value is ON.

elementConnectivity

A Boolean specifying whether to display the element connectivity. This member is valid when *probeObject*="ODB" and *probeEntity*=ELEMENT. The default value is ON.

elementFieldResults

A Boolean specifying whether to display the element field output results. This member is valid when *probeObject*="ODB", *probeEntity*=ELEMENT, and *isFieldOutputAvailable*=ON. The default value is ON.

nodeId

A Boolean specifying whether to display the node ID when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

baseCoordinates

A Boolean specifying whether to display the base coordinates of a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

deformedCoordinates

A Boolean specifying whether to display the deformed coordinates of a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

attachedElements

A Boolean specifying whether to display the elements attached to a node when *probeObject*="ODB" and *probeEntity*=NODE. The default value is ON.

nodeFieldResults

A Boolean specifying whether to display the node field output results. This member is valid when *probeObject*="ODB", *probeEntity*=NODE, and *isFieldOutputAvailable*=ON. The default value is ON.

legend

A Boolean specifying whether to display the legend for a curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

xValue

A Boolean specifying whether to display the *x*-coordinate value of the point on the curve being probed. This member is valid when *probeObject*="XYPlot." The default value is ON.

yValue

A Boolean specifying whether to display the *y*-coordinate value of the point on the curve being probed. This member is valid when *probeObject*=“XYPlot.” The default value is ON.

sequenceID

A Boolean specifying whether to display the sequence ID of the point on the curve being probed. This member is valid when *probeObject*=“XYPlot.” The default value is ON.

interpolateXy

A Boolean specifying whether to interpolate values within a line segment when *probeObject*=“XYPlot.” When *interpolateXy*=OFF, probing returns the nearest *X–Y* data point on the curve. When *interpolateXy*=ON, probing interpolates data to return a value at the nearest point on the curve. The default value is OFF.

isFieldOutputAvailable

A Boolean specifying whether field output is available for probing when *probeObject*=“XYPlot. This member is read-only.”

probeObject

A String specifying the type of the displayed object being probed. Possible values are "ODB" and "XYPlot". This member is read-only.

ProbeReport object

The ProbeReport object is used to store settings associated with tabular reports of probe data. The ProbeReport object has no constructor. Abaqus creates the *defaultProbeReport* and the *probeReport* members when you import the Visualization module. When probing is initiated for the first time, the values in the *probeReport* member are initialized using the values from the *defaultProbeReport* member.

Access

```
import visualization
session.defaultProbeReport
session.probeReport
```

`setValues(...)`

This method modifies the ProbeReport object.

Required arguments

None.

Optional arguments

options

`None` or a ProbeReport object specifying values to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

numColumns

An Int specifying the number of columns in the report file. This argument is valid only when *pageWidth*=SPECIFY. The default value is 80.

numDigits

An Int specifying the number of significant digits to be written for decimal values. The default value is 6.

numFormat

A SymbolicConstant specifying the number format to be used when formatting decimal values. Possible values are AUTOMATIC, ENGINEERING, and SCIENTIFIC. The default value is ENGINEERING.

pageWidth

A SymbolicConstant specifying the page width format. Possible values are NO_LIMIT and SPECIFY. The default value is NO_LIMIT.

printTotal

A Boolean specifying whether to print the total value of either the field output result (when *probeObject*="ODB") or the *x*- and *y*-coordinates (when *probeObject*="XYPlot"). The default value is OFF.

printMinMax

A Boolean specifying whether to print the minimum and maximum values of either the field output result (when *probeObject*="ODB") or the *x*- and *y*-coordinates (when *probeObject*="XYPlot"). The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The ProbeReport object has members with the same names and descriptions as the arguments to the *setValues* method.

SelectedProbeValues object

The SelectedProbeValues object has no constructor. The SelectedProbeValues object is created when you import the Visualization module.

Access

```
import visualization
session.selectedProbeValues
```

Members

The SelectedProbeValues object has the following members:

length

An Int specifying the length of the *values* member.

fieldOutputAvailable

A Boolean specifying whether any probe values have been selected (as is necessary prior to writing to a file).

values

A tuple of tuples of Floats specifying the selected probe values.

lastValues

A tuple of Floats specifying the last sequence of the *values* member.

Spectrum object

The Spectrum object defines a color spectrum for the contour display.

Access

```
import visualization
session.spectrums[name]
```

Spectrum(...)

This method creates a Spectrum object and places it in the spectrums repository.

Path

```
session.Spectrum
```

Required arguments

name	A string name for the spectrum.
colors	A sequence of strings indicating the colors of the spectrum.

Optional arguments

None.

Return value

A Spectrum object.

Exceptions

None.

Members

The Spectrum object has no members.

Stream object

TheStream object defines a set of streamlines in fluid mechanics.

Access

```
import visualization
session.streams[name]
```

Stream(...)

This method creates aStream object and places it in the streams repository.

Path

```
session.Stream
```

Required arguments

name	A string name for the stream.
numPointsOnRake	An integer specifying the number of points along the rake.

Optional arguments

pointA	A tuple of 3 floats specifying the starting point of the rake. Alternatively, a string representation of the node selected in the viewport.
pointB	A tuple of 3 floats specifying the end point of the rake. Alternatively, a string representation of the node selected in the viewport.
path	A*Path* object that specifies the rake.

Return value

A Stream object.

Exceptions

None.

Members

The Stream object has no members.

Plot Options commands

Plot options commands are used to control the appearance of plots in the Visualization module. Plots can be undeformed, deformed, contour, symbol, or material orientation.

In this section:

- *BasicOptions object*
- *DGCommonOptions object*
- *DGContourOptions object*
- *DisplayOptions object*
- *DGDisplayBodyOptions object*
- *FreeBodyOptions object*
- *HistoryVariable object*
- *OdbData object*
- *OdbDataDatumCsys object*
- *OdbDataElementSet object*
- *OdbDataFrame object*
- *OdbDataInstance object*
- *OdbDataMaterial object*
- *OdbDataNodeSet object*
- *OdbDataSection object*
- *OdbDataStep object*
- *OdbDataSurfaceSet object*
- *CouplingConstraint object*
- *RigidBodyConstraint object*
- *TieConstraint object*
- *MpcConstraint object*
- *OdbDiagnosticData object*
- *OdbAnalysisError object*
- *OdbAnalysisWarning object*
- *OdbJobTime object*
- *OdbNumericalProblemSummary object*
- *OdbDiagnosticStep object*
- *OdbAuxiliaryData object*
- *OdbDiagnosticIncrement object*
- *OdbDiagnosticAttempt object*
- *OdbContactDiagnostics object*
- *OdbDisplayOptions object*
- *OptionArg object*
- *DGOrientationOptions object*
- *StreamOptions object*
- *DGSuperimposeOptions object*
- *DGSymbolOptions object*

- *ViewCutOptions object*
- *DetailPlotOptions object*
- *PlyStackPlotOptions object*
- *MdbData object*
- *MdbDataFrame object*
- *MdbDataInstance object*
- *MdbDataStep object*

BasicOptions object

The BasicOptions object stores values and attributes associated with an OdbDisplay object.

The BasicOptions object has no constructor command. Abaqus creates the *defaultOdbDisplay.basicOptions* member when you import the Visualization module. Abaqus creates a *basicOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.basicOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the attributes from the previous active viewport. The previous active viewport contains the attributes from the *defaultOdbDisplay* object for the session. The attributes from the *defaultOdbDisplay* object are copied from the previous active viewport to create the new viewport.

BasicOptions objects are accessed in one of two ways:

- The default basic options: these settings are used as defaults when other *basicOptions* members are created. These settings can be set to customize user preferences.
- The basic options associated with a particular viewport.

Access

```
import visualization
session.defaultOdbDisplay.basicOptions
session.viewports[name].layers[name].odbDisplay.basicOptions
session.viewports[name].odbDisplay.basicOptions
```

`setValues(...)`

This method modifies the BasicOptions object.

Required arguments

None.

Optional arguments

options

A BasicOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

cameraCsysName

A String specifying the name of the coordinate system driving the moving camera.

cameraMovesWithCsys

A Boolean specifying whether the camera moves with the coordinate system. The default value is OFF.

cameraFollowsRotation

A Boolean specifying whether the camera, when it moves, follows the rotation of the coordinate system. The default value is OFF.

averagingThreshold

A Float specifying the nodal averaging threshold percentage. Possible values are 0 <*averagingThreshold*< 100. The default value is 75.0.

quantityToPlot

A SymbolicConstant specifying the quantity to plot. Possible values are FIELD_OUTPUT and DISCONTINUITIES. The default value is FIELD_OUTPUT.

curveRefinementLevel
A SymbolicConstant specifying the refinement level for drawing curves. Possible values are EXTRA_COARSE, COARSE, MEDIUM, FINE, and EXTRA_FINE. The default value is COARSE.

noResultsColor
A String specifying the color of elements that do not have any results. The default value is "White".

featureAngle
A Float specifying the feature angle to be used when *visibleEdges*=FEATURE. The default value is 30.0.

otherSymbolSize
An Int specifying the size of various glyph symbols (boundary conditions, coupling constraints, etc.). The default value is 6.

renderBeamProfiles
A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor
A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

renderShellThickness
A Boolean specifying whether to render the shell thickness. The default value is OFF.

shellScaleFactor
A Float specifying the shell thickness scale factor. The shellScaleFactor must be greater than zero. The default value is 1.0.

accountForDeactivatedElems
A Boolean specifying whether to account for deactivated elements. The default value is ON.

bcDisplay
A Boolean specifying whether to display boundary conditions. The default value is OFF.

connectorDisplay
A Boolean specifying whether to display connectors. The default value is OFF.

highlightConnectorPts
A Boolean specifying whether to highlight connector points. The default value is ON.

showConnectorAxes
A Boolean specifying whether to display connector orientations or coordinate systems. The default value is ON.

showConnectorType
A Boolean specifying whether to display the text that describes the connector type. The default value is ON.

pointElements
A Boolean specifying whether to display point type elements. The default value is ON.

referencePoints
A Boolean specifying whether to display reference points. *referencePoints* is valid only when *pointElements*=ON. The default value is ON.

massElements
A Boolean specifying whether to display mass, heat capacity and inertia elements. *massElements* is valid only when *pointElements*=ON. The default value is OFF.

springElements

A Boolean specifying whether to display spring and dashpot elements. *springElements* is valid only when *pointElements*=ON. The default value is OFF.

spotWelds

A Boolean specifying whether to display spot weld and distributed coupling elements. *spotWelds* is valid only when *pointElements*=ON. The default value is OFF.

tracerParticles

A Boolean specifying whether to display tracer particles. *tracerParticles* is valid only when *pointElements*=ON. The default value is OFF.

pc3dElements

A Boolean specifying whether to display PC3D elements. *pc3dElements* is valid only when *pointElements*=ON. The default value is ON.

pd3dElements

A Boolean specifying whether to display PD3D elements. *pd3dElements* is valid only when *pointElements*=ON. The default value is ON.

sweepArs

A Boolean specifying whether to sweep the analytical surfaces. The default value is ON or OFF, depending on the characteristics of your model.

sweepElem

A Boolean specifying whether to sweep the deformable elements. The default value is ON or OFF, depending on the characteristics of your model.

sweepStartAngleArs

A Float specifying the starting angle (in degrees) from which to sweep the analytical surfaces when *sweepArs*=ON. The default value is 0.0.

sweepStartAngleElem

A Float specifying the starting angle (in degrees) from which to sweep the model when *sweepElem*=ON. The default value is 0.0.

sweepEndAngleArs

A Float specifying the angle (in degrees) through which to sweep the analytical surfaces when *sweepArs*=ON. The default value is 360.0.

sweepEndAngleElem

A Float specifying the angle (in degrees) through which to sweep the model when *sweepElem*=ON. The default value is 180.0.

numSweepSegmentsArs

An Int specifying the number of segments to display when *sweepArs*=ON. The default value is 10 or 20, depending on characteristics of your model.

numSweepSegmentsElem

An Int specifying the number of segments to display when *sweepElem*=ON. The default value is 10 or 20, depending on characteristics of your model.

numericForm

A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, COMPLEX_MAG_AT_ANGLE, COMPLEX_ENVELOPE_MAX_ABS, COMPLEX_ENVELOPE_MAX, and COMPLEX_ENVELOPE_MIN. The default value is REAL.

complexAngle

A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_MAG_AT_ANGLE. The default value is 0.0.

sectionResults

A SymbolicConstant specifying which of the section points to use. Possible values are USE_BOTTOM, USE_TOP, USE_BOTTOM_AND_TOP, and USE_ENVELOPE. The default value is USE_BOTTOM.

envelopeCriteria

A SymbolicConstant specifying the envelope criterion. Possible values are MAX_VALUE, MIN_VALUE, and MAX_ABS_VALUE. The default value is MAX_ABS_VALUE.

envelopeDataPosition

A SymbolicConstant specifying the output position for envelope calculations. Possible values are CENTROID, ELEMENT_NODAL, and INTEGRATION_POINT. The default value is INTEGRATION_POINT.

plyResultLocation

A SymbolicConstant specifying the ply result location. Possible values are BOTTOM, MIDDLE, TOP, and TOP_AND_BOTTOM. The default value is MIDDLE.

sectionPointScheme

A SymbolicConstant specifying the section point scheme. Possible values are CATEGORY_BASED and PLY_BASED. The default value is CATEGORY_BASED.

sweepSectors

A Boolean specifying whether to sweep the cyclic symmetry sectors. The default value is OFF.

sectorSelectionType

A SymbolicConstant specifying how sectors will be selected for sweeping. Possible values are SELECT_BY_NUMBER, SELECT_BY_ANGLE, and SELECT_ALL. The default value is SELECT_BY_NUMBER.

selectedSectorNumbers

A sequence of Ints specifying which sectors to display when *sectorSelectionType*=SELECT_BY_NUMBER. Possible values are 1 $\leq$*selectedSectorNumbers*$\leq$ the number of sectors. The default value is (1).

sweepSectorStartAngle

A Float specifying the angle (in degrees) from which to sweep cyclic symmetry sectors when *sweepSectors*=ON. Possible values are multiples of the sector angle such that 0 $\leq$*sweepSectorStartAngle*$\leq$ 360. The default value is 0.0.

sweepSectorEndAngle

A Float specifying the angle (in degrees) through which to sweep cyclic symmetry sectors when *sweepSectors*=ON. Possible values are multiples of the sector angle such that 0 $<$*sweepSectorEndAngle*$\leq$ 360. The default value is 360.0.

extrudeArs

A Boolean specifying whether to extrude analytical surfaces. The default value is ON or OFF depending on characteristics of your model.

extrudeArsDepthAutoCompute

A Boolean specifying whether to use automatic depth determination when extruding analytical surfaces. The default value is ON. The default value is ON.

extrudeElem

A Boolean specifying whether to extrude deformable elements. The default value is ON or OFF depending on characteristics of your model.

extrudeArsDepth

A Float specifying the depth (in model units) by which the analytical surfaces are to be extruded when *extrudeArs*=ON. The default value is 1.0.

extrudeElemDepth

A Float specifying the depth (in model units) by which the deformable elements are to be extruded when *extrudeElem*=ON. The default value is 1.0.

mirrorPatternOrder

A SymbolicConstant specifying the order of operations to create the mirror pattern. Possible values are MIRROR_RECT_CIRC, RECT_MIRROR_CIRC, MIRROR_CIRC_RECT, RECT_CIRC_MIRROR, CIRC_MIRROR_RECT, and CIRC_RECT_MIRROR. The default value is MIRROR_RECT_CIRC.

mirrorCsysName

The SymbolicConstant GLOBAL or a String specifying the name of the mirror's coordinate system. The default value is GLOBAL.

mirrorAboutXyPlane

A Boolean specifying whether to mirror about the XY plane. The default value is OFF.

mirrorAboutXzPlane

A Boolean specifying whether to mirror about the XZ plane. The default value is OFF.

mirrorAboutYzPlane

A Boolean specifying whether to mirror about the YZ plane. The default value is OFF.

mirrorDisplayBodies

A Boolean specifying whether to mirror display bodies. The default value is OFF.

patternCsysName

The SymbolicConstant GLOBAL or a String specifying the name of the pattern's coordinate system. The default value is GLOBAL.

patternNumX

An Int specifying the number of patterns to create in the X-direction for a rectangular pattern. The default value is 1.

patternNumY

An Int specifying the number of patterns to create in the Y-direction for a rectangular pattern. The default value is 1.

patternNumZ

An Int specifying the number of patterns to create in the Z-direction for a rectangular pattern. The default value is 1.

patternOffsetX

A Float specifying the offset of the pattern along the X-axis for a rectangular pattern. The default value is 0.0.

patternOffsetY

A Float specifying the offset of the pattern along the Y-axis for a rectangular pattern. The default value is 0.0.

patternOffsetZ

A Float specifying the offset of the pattern along the Z-axis for a rectangular pattern. The default value is 0.0.

patternRotationAxis

A SymbolicConstant specifying the axis of rotation for a circular. Possible values are XAXIS, YAXIS, and ZAXIS. The default value is ZAXIS.

patternTotalAngle

A Float specifying the total angle of a circular pattern. The default value is 360.0.

patternNumCircular

An Int specifying the number of patterns to create in a circular pattern. The default value is 1.

couplingDisplay

A Boolean specifying whether to display coupling constraints. The default value is ON.

coordSystemDisplay

A Boolean specifying whether to display coordinate systems. The default value is OFF.

scratchCoordSystemDisplay

A Boolean specifying whether to display coordinate systems that represent user-defined orientations. The default value is OFF.

transformationType

A SymbolicConstant specifying the transformation to apply to the PrimaryVariable. Possible values are DEFAULT, NODAL, USER_SPECIFIED, ANGULAR, and LAYUP_ORIENTATION. The default value is DEFAULT.

If *transformationType*=NODAL, Abaqus will transform nodal vector fields into any orientation defined in the analysis with the **TRANSFORM* option. Setting *transformationType*=NODAL has no effect on element-based results.

If *transformationType*=USER_SPECIFIED, Abaqus will transform tensor and vector fields into the coordinate system specified by *datumCsys*.

If *transformationType*=LAYUP_ORIENTATION, Abaqus will transform tensor and vector fields into the layup orientation defined in the composite section. The odb should contain the field SORIENT in order to use this option.

datumCsys

A *DatumCsys* object specifying the coordinate system to use for results transformation when *transformationType*=USER_SPECIFIED.

rigidTransformPrimary

A Boolean specifying whether to perform a rigid transformation of nodal vector datasets based on the active user specific system The default value is OFF.

rigidTransformDeformed

A Boolean specifying whether to perform a rigid transformation of current deformedVariable based on the active user specific system The default value is OFF.

transformOnDeformed

A Boolean specifying whether to include the effects of deformation on the transformation calculations The default value is ON.

averageElementOutput

A Boolean specifying whether to average the element output. The default value is ON.

averageOnlyDisplayed

A Boolean specifying whether to average only values on displayed elements. The default value is ON.

computeOutput

A SymbolicConstant specifying the order or the computations to be performed. Possible values are EXTRAPOLATE_AVERAGE_COMPUTE, EXTRAPOLATE_COMPUTE_AVERAGE, EXTRAPOLATE_COMPUTE, EXTRAPOLATE_COMPUTE_DISCONTINUITIES, and RAW_DATA. The default value is EXTRAPOLATE_AVERAGE_COMPUTE.

regionBoundaries A SymbolicConstant specifying the type of averaging region boundaries. Possible values are NONE, ODB_REGIONS, ELEMENT_SET, and DISPLAY_GROUPS. The default value is ODB_REGIONS.

useRegionBoundaries A Boolean specifying whether to use region boundaries when averaging. The default value is ON.

userRegions A sequence of Strings specifying either element set or display group names (depending on the value of regionBoundaries) defining the averaging region boundaries. The default value is an empty sequence.

includeFeatureBoundaries A Boolean specifying whether to include additional averaging boundaries for shells and membranes based on feature edges. The default value is ON.

Return value

`None.`

Exceptions

If *featureAngle* is not in the valid range: RangeError: featureAngle must be a float in the range 0-90, inclusive

Members

The BasicOptions object can have the following members:

regionAveraging A Boolean specifying whether to ignore region boundaries when computing values. The default value is ON.

cameraMovesWithCsys A Boolean specifying whether the camera moves with the coordinate system. The default value is OFF.

cameraFollowsRotation A Boolean specifying whether the camera, when it moves, follows the rotation of the coordinate system. The default value is OFF.

averagingThreshold A Float specifying the nodal averaging threshold percentage. Possible values are 0 <*averagingThreshold*< 100. The default value is 75.0.

quantityToPlot A SymbolicConstant specifying the quantity to plot. Possible values are FIELD_OUTPUT and DISCONTINUITIES. The default value is FIELD_OUTPUT.

extrapAlgorithm A SymbolicConstant specifying the extrapolation algorithm. This member is for internal use only. The only possible value is EXTRAP_COMPUTE_AVERAGE.

curveRefinementLevel A SymbolicConstant specifying the refinement level for drawing curves. Possible values are EXTRA_COARSE, COARSE, MEDIUM, FINE, and EXTRA_FINE. The default value is COARSE.

featureAngle
A Float specifying the feature angle to be used when *visibleEdges*=FEATURE. The default value is 30.0.

otherSymbolSize
An Int specifying the size of various glyph symbols (boundary conditions, coupling constraints, etc.). The default value is 6.

renderBeamProfiles
A Boolean specifying whether to render the beam profiles. The default value is OFF.

beamScaleFactor
A Float specifying the beam profile scale factor. The beamScaleFactor must be greater than zero. The default value is 1.0.

renderShellThickness
A Boolean specifying whether to render the shell thickness. The default value is OFF.

shellScaleFactor
A Float specifying the shell thickness scale factor. The shellScaleFactor must be greater than zero. The default value is 1.0.

accountForDeactivatedElems
A Boolean specifying whether to account for deactivated elements. The default value is ON.

bcDisplay
A Boolean specifying whether to display boundary conditions. The default value is OFF.

connectorDisplay
A Boolean specifying whether to display connectors. The default value is OFF.

highlightConnectorPts
A Boolean specifying whether to highlight connector points. The default value is ON.

showConnectorAxes
A Boolean specifying whether to display connector orientations or coordinate systems. The default value is ON.

showConnectorType
A Boolean specifying whether to display the text that describes the connector type. The default value is ON.

pointElements
A Boolean specifying whether to display point type elements. The default value is ON.

referencePoints
A Boolean specifying whether to display reference points. *referencePoints* is valid only when *pointElements*=ON. The default value is ON.

massElements
A Boolean specifying whether to display mass, heat capacity and inertia elements. *massElements* is valid only when *pointElements*=ON. The default value is OFF.

springElements
A Boolean specifying whether to display spring and dashpot elements. *springElements* is valid only when *pointElements*=ON. The default value is OFF.

spotWelds
A Boolean specifying whether to display spot weld and distributed coupling elements. *spotWelds* is valid only when *pointElements*=ON. The default value is OFF.

tracerParticles
A Boolean specifying whether to display tracer particles. *tracerParticles* is valid only when *pointElements*=ON. The default value is OFF.

pc3dElements
A Boolean specifying whether to display PC3D elements. *pc3dElements* is valid only when *pointElements*=ON. The default value is ON.

pd3dElements
A Boolean specifying whether to display PD3D elements. *pd3dElements* is valid only when *pointElements*=ON. The default value is ON.

sweepArs
A Boolean specifying whether to sweep the analytical surfaces. The default value is ON or OFF, depending on the characteristics of your model.

sweepElem
A Boolean specifying whether to sweep the deformable elements. The default value is ON or OFF, depending on the characteristics of your model.

sweepStartAngleArs
A Float specifying the starting angle (in degrees) from which to sweep the analytical surfaces when *sweepArs*=ON. The default value is 0.0.

sweepStartAngleElem
A Float specifying the starting angle (in degrees) from which to sweep the model when *sweepElem*=ON. The default value is 0.0.

sweepEndAngleArs
A Float specifying the angle (in degrees) through which to sweep the analytical surfaces when *sweepArs*=ON. The default value is 360.0.

sweepEndAngleElem
A Float specifying the angle (in degrees) through which to sweep the model when *sweepElem*=ON. The default value is 180.0.

numSweepSegmentsArs
An Int specifying the number of segments to display when *sweepArs*=ON. The default value is 10 or 20, depending on characteristics of your model.

numSweepSegmentsElem
An Int specifying the number of segments to display when *sweepElem*=ON. The default value is 10 or 20, depending on characteristics of your model.

numericForm
A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, COMPLEX_MAG_AT_ANGLE, COMPLEX_ENVELOPE_MAX_ABS, COMPLEX_ENVELOPE_MAX, and COMPLEX_ENVELOPE_MIN. The default value is REAL.

complexAngle
A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_MAG_AT_ANGLE. The default value is 0.0.

sectionResults
A SymbolicConstant specifying which of the section points to use. Possible values are USE_BOTTOM, USE_TOP, USE_BOTTOM_AND_TOP, and USE_ENVELOPE. The default value is USE_BOTTOM.

envelopeCriteria
A SymbolicConstant specifying the envelope criterion. Possible values are MAX_VALUE, MIN_VALUE, and MAX_ABS_VALUE. The default value is MAX_ABS_VALUE.

envelopeDataPosition
A SymbolicConstant specifying the output position for envelope calculations. Possible values are CENTROID, ELEMENT_NODAL, and INTEGRATION_POINT. The default value is INTEGRATION_POINT.

plyResultLocation
A SymbolicConstant specifying the ply result location. Possible values are BOTTOM, MIDDLE, TOP, and TOP_AND_BOTTOM. The default value is MIDDLE.

sectionPointScheme
A SymbolicConstant specifying the section point scheme. Possible values are CATEGORY_BASED and PLY_BASED. The default value is CATEGORY_BASED.

sweepSectors
A Boolean specifying whether to sweep the cyclic symmetry sectors. The default value is OFF.

sectorSelectionType
A SymbolicConstant specifying how sectors will be selected for sweeping. Possible values are SELECT_BY_NUMBER, SELECT_BY_ANGLE, and SELECT_ALL. The default value is SELECT_BY_NUMBER.

sweepSectorStartAngle
A Float specifying the angle (in degrees) from which to sweep cyclic symmetry sectors when *sweepSectors*=ON. Possible values are multiples of the sector angle such that $0 \leq$ *sweepSectorStartAngle* ≤ 360. The default value is 0.0.

sweepSectorEndAngle
A Float specifying the angle (in degrees) through which to sweep cyclic symmetry sectors when *sweepSectors*=ON. Possible values are multiples of the sector angle such that $0 <$ *sweepSectorEndAngle* ≤ 360. The default value is 360.0.

extrudeArs
A Boolean specifying whether to extrude analytical surfaces. The default value is ON or OFF depending on characteristics of your model.

extrudeArsDepthAutoCompute
A Boolean specifying whether to use automatic depth determination when extruding analytical surfaces. The default value is ON. The default value is ON.

extrudeElem
A Boolean specifying whether to extrude deformable elements. The default value is ON or OFF depending on characteristics of your model.

extrudeArsDepth
A Float specifying the depth (in model units) by which the analytical surfaces are to be extruded when *extrudeArs*=ON. The default value is 1.0.

extrudeElemDepth
A Float specifying the depth (in model units) by which the deformable elements are to be extruded when *extrudeElem*=ON. The default value is 1.0.

mirrorPatternOrder
A SymbolicConstant specifying the order of operations to create the mirror pattern. Possible values are MIRROR_RECT_CIRC, RECT_MIRROR_CIRC, MIRROR_CIRC_RECT, RECT_CIRC_MIRROR, CIRC_MIRROR_RECT, and CIRC_RECT_MIRROR. The default value is MIRROR_RECT_CIRC.

mirrorAboutXyPlane
A Boolean specifying whether to mirror about the XY plane. The default value is OFF.

mirrorAboutXzPlane
A Boolean specifying whether to mirror about the XZ plane. The default value is OFF.

mirrorAboutYzPlane
A Boolean specifying whether to mirror about the YZ plane. The default value is OFF.

mirrorDisplayBodies
A Boolean specifying whether to mirror display bodies. The default value is OFF.

patternNumX
An Int specifying the number of patterns to create in the X-direction for a rectangular pattern. The default value is 1.

patternNumY
An Int specifying the number of patterns to create in the Y-direction for a rectangular pattern. The default value is 1.

patternNumZ
An Int specifying the number of patterns to create in the Z-direction for a rectangular pattern. The default value is 1.

patternOffsetX
A Float specifying the offset of the pattern along the X-axis for a rectangular pattern. The default value is 0.0.

patternOffsetY
A Float specifying the offset of the pattern along the Y-axis for a rectangular pattern. The default value is 0.0.

patternOffsetZ
A Float specifying the offset of the pattern along the Z-axis for a rectangular pattern. The default value is 0.0.

patternRotationAxis
A SymbolicConstant specifying the axis of rotation for a circular. Possible values are XAXIS, YAXIS, and ZAXIS. The default value is ZAXIS.

patternTotalAngle
A Float specifying the total angle of a circular pattern. The default value is 360.0.

patternNumCircular
An Int specifying the number of patterns to create in a circular pattern. The default value is 1.

couplingDisplay
A Boolean specifying whether to display coupling constraints. The default value is ON.

coordSystemDisplay
A Boolean specifying whether to display coordinate systems. The default value is OFF.

scratchCoordSystemDisplay
A Boolean specifying whether to display coordinate systems that represent user-defined orientations. The default value is OFF.

transformationType
A SymbolicConstant specifying the transformation to apply to the PrimaryVariable. Possible values are DEFAULT, NODAL, USER_SPECIFIED, ANGULAR, and LAYUP_ORIENTATION. The default value is DEFAULT.

If *transformationType*=NODAL, Abaqus will transform nodal vector fields into any orientation defined in the analysis with the **TRANSFORM* option. Setting *transformationType*=NODAL has no effect on element-based results.

If *transformationType*=USER_SPECIFIED, Abaqus will transform tensor and vector fields into the coordinate system specified by *datumCsys*.

If *transformationType*=LAYUP_ORIENTATION, Abaqus will transform tensor and vector fields into the layup orientation defined in the composite section. The odb should contain the field SORIENT in order to use this option.

rigidTransformPrimary

A Boolean specifying whether to perform a rigid transformation of nodal vector datasets based on the active user specific system The default value is OFF.

rigidTransformDeformed

A Boolean specifying whether to perform a rigid transformation of current deformedVariable based on the active user specific system The default value is OFF.

transformOnDeformed

A Boolean specifying whether to include the effects of deformation on the transformation calculations The default value is ON.

modelCanExtrude

A Boolean specifying whether the model contains any elements or surfaces that can be extruded.

sweepModelType

An Int specifying the types of sweepable elements and surfaces contained in the model, if any.

averageElementOutput

A Boolean specifying whether to average the element output. The default value is ON.

averageOnlyDisplayed

A Boolean specifying whether to average only values on displayed elements. The default value is ON.

regionBoundaries

A SymbolicConstant specifying the type of averaging region boundaries. Possible values are NONE, ODB_REGIONS, ELEMENT_SET, and DISPLAY_GROUPS. The default value is ODB_REGIONS.

useRegionBoundaries

A Boolean specifying whether to use region boundaries when averaging. The default value is ON.

includeFeatureBoundaries

A Boolean specifying whether to include additional averaging boundaries for shells and membranes based on feature edges. The default value is ON.

numSectors

An Int specifying the number of sectors of a cyclic symmetric model. The value is automatically computed from the cyclic symmetric model. This value is read-only.

sectorAngle

A Float specifying the sector angle of a cyclic symmetric model. The value is automatically computed from the cyclic symmetric model. This value is read-only.

automaticExtrudeDepth

A Float specifying the automatic extrude depth used to extrude analytical rigid surfaces in the default setting. This value is read-only.

cameraCsysName

A String specifying the name of the coordinate system driving the moving camera.

noResultsColor

A String specifying the color of elements that do not have any results. The default value is "White".

mirrorCsysName

The SymbolicConstant GLOBAL or a String specifying the name of the mirror's coordinate system. The default value is GLOBAL.

patternCsysName

The SymbolicConstant GLOBAL or a String specifying the name of the pattern's coordinate system. The default value is GLOBAL.

datumCsys

A *DatumCsys* object specifying the coordinate system to use for results transformation when *transformationType*=USER_SPECIFIED.

selectedSectorNumbers

A tuple of Ints specifying which sectors to display when *sectorSelectionType*=SELECT_BY_NUMBER. Possible values are 1 $\leq$*selectedSectorNumbers*$\leq$ the number of sectors. The default value is (1).

userRegions

A tuple of Strings specifying either element set or display group names (depending on the value of regionBoundaries) defining the averaging region boundaries. The default value is an empty sequence.

DGCommonOptions object

The DGCommonOptions object stores values and attributes that are common to all plot states. The DGCommonOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.commonOptions* member when a display group instance is created, using values from *odbDisplay.commonOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.commonOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.commonOptions
```

Members

The DGCommonOptions object can have the following members:

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is SHADED.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

normals

A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay

A SymbolicConstant specifying whether to draw element normals or surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

normalArrowLength

A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness

A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle

A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency

A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor

A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

labelFont

A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabelColor

A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabelColor

A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabelColor

A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbolColor

A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_1}$-direction. The default value is "Blue".

beamN2Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_2}$-direction. The default value is "Red".

beamTangentColor

A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

DGContourOptions object

The DGContourOptions object stores values and attributes associated with a contour plot. The DGContourOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.contourOptions* member when a display group instance is created, using values from *odbDisplay.contourOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.contourOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.contourOptions
```

Members

The DGContourOptions object can have the following members:

contourMethod

A SymbolicConstant specifying the contour rendering method. Possible values are TEXTURE_MAPPED and TESSELLATED. The default value is TEXTURE_MAPPED.

tickmarkPlots

A Boolean specifying whether tick mark plots should be displayed on line-type elements. If *tickmarkPlots*=ON, Abaqus displays a tick mark plot. If *tickmarkPlots*=OFF, Abaqus displays contours on the element faces. The default value is OFF.

contourStyle

A SymbolicConstant specifying the interval style of the contour plot. Possible values are CONTINUOUS and UNIFORM. The default value is UNIFORM.

contourEdges

A Boolean specifying whether to plot the edges of each contour interval when *contourType*=BANDED or ISOSURFACE. The default value is OFF.

contourEdgeStyle

A SymbolicConstant specifying the edge line style to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

contourEdgeThickness

A SymbolicConstant specifying the edge line thickness to be used to plot the edge of the contour intervals when *contourType*=BANDED or ISOSURFACE. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

averagedOrientationDisplay

A Boolean specifying the display of the nodal averaged coordinate systems used when averaging element vector or tensor data. The default value is OFF.

matchingPlyLabels

A Boolean specifying whether the label for the matching ply shows up in the viewport. The default value is OFF.

colorByMatchingPlies

A Boolean specifying whether the contour color is driven by the matching ply. The default value is OFF.

tickmarkAxisLength

A SymbolicConstant specifying the length of the tick mark plot axes. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

tickmarkBaseValue

A Float specifying the base contour value defining the tick mark axis contour value that intersects the elements. Possible values are *autoMinValue*≤*tickmarkBaseValue*≤*autoMaxValue*. The default value is 0.0.

tickmarkOrientation

A SymbolicConstant specifying the orientation of the tick mark plots. Possible values are N1 and N2. The default value is N2.

edgeColorLine

A String specifying the edge color to be used when *contourType*=LINE. The default value is "White".

edgeColorBandedQuilt

A String specifying the edge color to be used when *contourType*=BANDED or QUILT. The default value is "Black".

contourEdgeColor

A String specifying the color to be used to plot the contour edges when *contourType*=BANDED or ISOSURFACE. The default value is "Grey60".

tickmarkCurveColor

A String specifying the color to be used to plot the tick mark curve. The default value is "Cyan".

intervalLineAttributes

A tuple of tuples of SymbolicConstants specifying the line style and line thickness for each interval in the plot when *contourType*=LINE. The size of the outer sequence must be equal to *numIntervals*-1. The inner sequence consists of two SymbolicConstants specifying the line style and line thickness. For possible values, refer to the *edgeLineStyle* and *edgeLineThickness* members of the *DGCommonOptions* object. The default is ((SOLID, VERY_THIN),).

DisplayOptions object

The DisplayOptions object stores a plot state.

Access

```
import visualization
session.viewports[name].layers[name].odbDisplay.display
session.viewports[name].odbDisplay.display
```

`setValues(...)`

This method modifies the DisplayOptions object.

Required arguments

None.

Optional arguments

options
A DisplayOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

plotState
A sequence of SymbolicConstants specifying the plot state of the display. Possible values are UNDEFORMED, DEFORMED, CONTOURS_ON_UNDEF, CONTOURS_ON_DEF, SYMBOLS_ON_UNDEF, SYMBOLS_ON_DEF, ORIENT_ON_UNDEF, and ORIENT_ON_DEF. The default value is (UNDEFORMED).

Return value

`None.`

Exceptions

None.

Members

The DisplayOptions object can have the following member:

plotState
A tuple of SymbolicConstants specifying the plot state of the display. Possible values are UNDEFORMED, DEFORMED, CONTOURS_ON_UNDEF, CONTOURS_ON_DEF, SYMBOLS_ON_UNDEF, SYMBOLS_ON_DEF, ORIENT_ON_UNDEF, and ORIENT_ON_DEF. The default value is (UNDEFORMED).

DGDisplayBodyOptions object

The DGDisplayBodyOptions object stores values and attributes applied to display bodies. The DGDisplayBodyOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.displayBodyOptions* member when a display group instance is created, using values from *odbDisplay.displayBodyOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.displayBodyOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.displayBodyOptions
```

Members

The DGDisplayBodyOptions object can have the following members:

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is EXTERIOR.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

translucency

A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor

A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "White".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

FreeBodyOptions object

The FreeBodyOptions object stores values and attributes associated with a free body plot. The FreeBodyOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.freeBodyOptions* member when you import the Visualization module. Abaqus creates a *FreeBodyOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.freeBodyOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

FreeBodyOptions objects are accessed in one of two ways:

- The default free body options. These settings are used as defaults when other *freeBodyOptions* members are created. These settings can be set to customize user preferences.
- The free body options associated with a particular viewport.

Access

```
import visualization
session.defaultOdbDisplay.freeBodyOptions
session.viewports[name].layers[name].odbDisplay.freeBodyOptions
session.viewports[name].odbDisplay.freeBodyOptions
```

`setValues(...)`

This method modifies the FreeBodyOptions object.

Required arguments

None.

Optional arguments

comp1ColorF
A String specifying color of the first force component. The default value is "#FF1932".

comp1ColorM
A String specifying color of the first moment component. The default value is "#0000FF".

comp2ColorF
A String specifying color of the second force component. The default value is "#FF1932".

comp2ColorM
A String specifying color of the second moment component. The default value is "#0000FF".

comp3ColorF
A String specifying color of the third force component. The default value is "#FF1932".

comp3ColorM
A String specifying color of the third moment component. The default value is "#0000FF".

resultantColorF
A String specifying color of the resultant force. The default value is "#FF1932".

resultantColorM A String specifying color of the resultant moment. The default value is "#0000FF".

textColorF A String specifying text color for force. The default value is "Yellow".

textColorM A String specifying text color for moment. The default value is "Yellow".

textFontF A String specifying text font for force. The default value is "verdana".

textFontM A String specifying text font for moment. The default value is "verdana".

numberFormatF A SymbolicConstant specifying the number format for force. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

numberFormatM A SymbolicConstant specifying the number format for moment. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

scaleModeF A SymbolicConstant specifying the size scaling mode for force. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

scaleModeM A SymbolicConstant specifying the size scaling mode for moment. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

vectorDisplay A SymbolicConstant specifying the vector display mode. Possible values are RESULTANT and COMPONENT. The default value is RESULTANT.

numDigitsF An Int specifying the number of digits in the force label. The default value is 3.

numDigitsM An Int specifying the number of digits in the moment label. The default value is 3.

sizePercentageF A Float specifying the size of the force symbol as a percentage of the screen or model. The default value is 10.0.

sizePercentageM A Float specifying the size of the moment symbol as a percentage of the screen or model. The default value is 10.0.

thresholdF A Float specifying the force threshold value. The default value is 10^{-6}.

thresholdM A Float specifying the moment threshold value. The default value is 10^{-6}.

drawLabelF A Boolean specifying whether to draw force labels. The default value is ON.

drawLabelM A Boolean specifying whether to draw moment labels. The default value is ON.

showComp1F A Boolean specifying whether to show the first force component. The default value is ON.

showComp1M A Boolean specifying whether to show the first moment component. The default value is ON.

showComp2F A Boolean specifying whether to show the second force component. The default value is ON.

showComp2M A Boolean specifying whether to show the second moment component. The default value is ON.

showComp3F A Boolean specifying whether to show the third force component. The default value is ON.

showComp3M A Boolean specifying whether to show the third moment component. The default value is ON.

showForce A Boolean specifying whether to show forces. The default value is ON.

showMoment A Boolean specifying whether to show moments. The default value is ON.

constantLengthArrow A Boolean specifying whether to use a constant length for all arrows. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The FreeBodyOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

HistoryVariable object

The HistoryVariable object stores history data.

Access

```
import visualization
session.odbData[name].historyVariables[i]
```

Members

The HistoryVariable object has the following members:

name

A String specifying the history request label. This String is read-only.

legendLabel

A String specifying the legend text. This String is read-only.

steps

A tuple of (String, Int, SymbolicConstant) tuples specifying the steps. This sequence is read-only. Each inner sequence contains the following elements:

- *stepLabel*: A String specifying the step label.
- *stepNumber*: An Int specifying the step number.
- *procedureDomain*: A SymbolicConstant specifying the analysis type of the step, which can have these values: “TIME,” “FREQUENCY,” or “MODAL.”

OdbData object

The OdbData object stores non persistent values and attributes associated with an open odb for the given session. The OdbData object has no constructor. Abaqus creates the *odbData* repository when you import the Visualization module. Abaqus creates a OdbData object when an odb is opened.

Access

```
import visualization
session.odbData[name]
```

setValues(...)

This method modifies the OdbData object.

Required arguments

None.

Optional arguments

activeFrames

A sequence specifying the active frames, or the SymbolicConstant ALL. Each item in the sequence is a tuple defining the stepName followed by a sequence of expressions specifying frame numbers. The expression can be any of the following:

- An Int specifying a single frame number; for example, `1`.
- A String specifying a single frame number ; for example, `'7'`.
- A String specifying a sequence of frame numbers; for example, `'3:5'` and `'3:15:3'`.

For these expressions a negative number will indicate reverse numbering: -1 is the last frame of the step, -2 is the one before the last frame. Frame numbering starts at 0.

stepPeriods

A sequence of (String, Float) sequences specifying the stepName and the stepPeriod. Alternatively, this member may take the value ODB_VALUES.

Return value

`None.`

Exceptions

None.

Members

The OdbData object can have the following members:

activeFrames

A tuple specifying the active frames, or the SymbolicConstant ALL. Each item in the sequence is a tuple defining the stepName followed by a sequence of expressions specifying frame numbers. The expression can be any of the following:

- An Int specifying a single frame number; for example, `1`.

- A String specifying a single frame number ; for example, `'7'`.
- A String specifying a sequence of frame numbers; for example, `'3:5'` and `'3:15:3'`.

For these expressions a negative number will indicate reverse numbering: -1 is the last frame of the step, -2 is the one before the last frame. Frame numbering starts at 0.

stepPeriods

A tuple of (String, Float) tuples specifying the stepName and the stepPeriod. Alternatively, this member may take the value ODB_VALUES.

historyVariables

A repository of *HistoryVariable* objects specifying the history request label. The repository is read-only.

steps

A repository of *OdbDataStep* objects specifying the list of steps. The repository is read-only.

instances

A repository of *OdbDataInstance* objects specifying the list of instances. The repository is read-only.

materials

A repository of *OdbDataMaterial* objects specifying the list of materials. The repository is read-only.

sections

A repository of *OdbDataSection* objects specifying the list of sections. The repository is read-only.

elementSets

A repository of *OdbDataElementSet* objects specifying the list of element sets. The repository is read-only.

nodeSets

A repository of *OdbDataNodeSet* objects specifying the list of node sets. The repository is read-only.

surfaceSets

A repository of *OdbDataSurfaceSet* objects specifying the list of surface sets. The repository is read-only.

datumCsyses

A repository of *OdbDataDatumCsys* objects specifying the list of coordinate systems defined in the model. The repository is read-only.

kinematicCouplings

A repository of *CouplingConstraint* objects specifying the list of kinematic couplings. The repository is read-only.

distributingCouplings

A repository of *CouplingConstraint* objects specifying the list of distributing couplings. The repository is read-only.

shellSolidCouplings

A repository of *CouplingConstraint* objects specifying the list of shellsolid couplings. The repository is read-only.

rigidbodies

A repository of *RigidBodyConstraint* objects specifying the list of rigid body constraints. The repository is read-only.

multiPointConstraints A repository of *MpcConstraint* objects specifying the list of multipoint constraints. The repository is read-only.

ties A repository of *TieConstraint* objects specifying the list of Tie constraints. The repository is read-only.

diagnosticData An *OdbDiagnosticData* object.

OdbDataDatumCsys object

The OdbDataDatumCsys object stores coordinate system data.

Access

```
import visualization
session.odbData[name].datumCsyses[i]
```

Members

The OdbDataDatumCsys object can have the following members:

name
A String specifying the coordinate system name. This String is read-only.

type
A SymbolicConstant specifying the coordinate system type. This String is read-only. Possible values are CARTESIAN, CYLINDRICAL, and SPHERICAL.

xAxis
A tuple of three Floats specifying a sequence of three floats specifying the x-Axis vector. The default value is (1, 0, 0).

yAxis
A tuple of three Floats specifying a sequence of three floats specifying the y-Axis vector. The default value is (0, 1, 0).

zAxis
A tuple of three Floats specifying a sequence of three floats specifying the z-Axis vector. The default value is (0, 0, 1).

origin
A tuple of three Floats specifying a sequence of three floats specifying the origin. The default value is (0, 0, 0).

OdbDataElementSet object

The OdbDataElementSet object stores element set data.

Access

```
import visualization
session.odbData[name].elementSets[i]
```

Members

The OdbDataElementSet object has the following members:

name

A String specifying the set name. This attribute is read-only.

elements

A String-to-tuple-of-Ints Dictionary specifying the elements in the set. This attribute is read-only.

OdbDataFrame object

The OdbDataFrame object.

Access

```
import visualization
session.odbData[name].steps[i].frames[i]
```

`setValues(...)`

This method modifies the OdbDataFrame object.

Required arguments

activateFrame A Boolean specifying whether to activate the frame.

Optional arguments

update A Boolean specifying whether to update the model. The default value is ON

Return value

`None.`

Exceptions

None.

Members

The OdbDataFrame object has no members.

OdbDataInstance object

The OdbDataInstance object instance data.

Access

```
import visualization
session.odbData[name].instances[i]
```

Members

The OdbDataInstance object has the following member:

name A String specifying the instance name. This attribute is read-only.

OdbDataMaterial object

The OdbDataMaterial object stores material data.

Access

```
import visualization
session.odbData[name].materials[i]
```

Members

The OdbDataMaterial object has the following members:

name

A String specifying the set name. This attribute is read-only.

elements

A String-to-tuple-of-Ints Dictionary specifying the elements in the set. This attribute is read-only.

OdbDataNodeSet object

The OdbDataNodeSet object stores node set data.

Access

```
import visualization
session.odbData[name].nodeSets[i]
```

Members

The OdbDataNodeSet object has the following members:

name
A String specifying the set name. This attribute is read-only.

nodes
A String-to-tuple-of-Ints Dictionary specifying the nodes in the set. This attribute is read-only.

OdbDataSection object

The OdbDataSection object stores section data.

Access

```
import visualization
session.odbData[name].sections[i]
```

Members

The OdbDataSection object has the following members:

name

A String specifying the set name. This attribute is read-only.

elements

A String-to-tuple-of-Ints Dictionary specifying the elements in the set. This attribute is read-only.

OdbDataStep object

The OdbDataStep object.

Access

```
import visualization
session.odbData[name].steps[i]
```

`setValues(...)`

This method modifies the OdbDataStep object.

Required arguments

activateFrames A Boolean specifying whether to activate all the frames in the step.

Optional arguments

update A Boolean specifying whether to update the model. The default value is ON

Return value

`None.`

Exceptions

None.

Members

The OdbDataStep object has the following member:

frames An *OdbDataFrameArray* object specifying the list of frames. The list is read-only.

OdbDataSurfaceSet object

The OdbDataSurfaceSet object stores surface set data.

Access

```
import visualization
session.odbData[name].surfaceSets[i]
```

Members

The OdbDataSurfaceSet object has the following members:

name

A String specifying the set name. This attribute is read-only.

elements

A String-to-tuple-of-Ints Dictionary specifying the elements in the set. This attribute is read-only.

facets

A String-to-tuple-of-Ints Dictionary specifying the facets corresponding to the *elements*. This attribute is read-only.

CouplingConstraint object

The CouplingConstraint object.

Access

```
import visualization
session.odbData[name].kinematicCouplings[i]
session.odbData[name].distributingCouplings[i]
session.odbData[name].shellSolidCouplings[i]
```

`constraintData(...)`

This method returns node numbers of the surface being controlled by the control point.

Required arguments

None.

Optional arguments

None.

Return value

Tuple-of-Ints Dictionary specifying the node numbers on the controlled surface.

Exceptions

None.

Members

The CouplingConstraint object has the following members:

name A String specifying the coupling name. This attribute is read-only.

type A String specifying the type of coupling. This attribute is read-only.

RigidBodyConstraint object

The RigidBodyConstraint object.

Access

```
import visualization
session.odbData[name].rigidbodies[i]
```

`constraintData(...)`

This method returns constraint data if any are associated with the object.

Required arguments

None.

Optional arguments

None.

Return value

A String Value: NONE in the absence of constraint data.

Exceptions

None.

Members

The RigidBodyConstraint object has the following members:

name A String specifying the rigidbody constraint name. This attribute is read-only.

TieConstraint object

The TieConstraint object.

Access

```
import visualization
session.odbData[name].ties[i]
```

`constraintData(...)`

This method returns constraint data if any are associated with the object

Required arguments

None.

Optional arguments

None.

Return value

A tuple containing coordinates of the nodes pertaining to the constraint.

Exceptions

None.

Members

The TieConstraint object has the following members:

name A String specifying the tie constraint name. This attribute is read-only.

MpcConstraint object

The MpcConstraint object.

Access

```
import visualization
session.odbData[name].multiPointConstraints[i]
```

`constraintData(...)`

This method returns constraint data if any are associated with the object.

Required arguments

None.

Optional arguments

None.

Return value

A tuple containing coordinates of the nodes pertaining to the constraint.

Exceptions

None.

Members

The MpcConstraint object has the following members:

name A String specifying the multipoint constraint name. This attribute is read-only.

OdbDiagnosticData object

The OdbDiagnosticData object.

Access

```
import visualization
session.odbData[name].diagnosticData
```

Members

The OdbDiagnosticData object has the following members:

analysisErrors
A repository of *OdbAnalysisError* objects.

analysisWarnings
A repository of *OdbAnalysisWarning* objects.

steps
A repository of *OdbDiagnosticStep* objects.

jobTime
An *OdbJobTime* object.

numericalProblemSummary
An *OdbNumericalProblemSummary* object.

isXplDoublePrecision
A boolean specifying whether or not double precision is used for the analysis. This attribute is read-only.

jobStatus
A String specifying the job status after the analysis. This attribute is read-only.

numDomains
An int specifying the number of domains. This attribute is read-only.

numberOfAnalysisErrors
An int specifying the number of analysis errors encountered. This attribute is read-only.

numberOfAnalysisWarnings
An int specifying the number of analysis warnings encountered. This attribute is read-only.

numberOfSteps
An int specifying the number of steps present in the analysis. This attribute is read-only.

OdbAnalysisError object

The OdbAnalysisError object stores the description of different errors encountered during the analysis.

Access

```
import visualization
session.odbData[name].diagnosticData.analysisErrors[i]
```

Members

The OdbAnalysisError object has the following members:

incrementNumber
An int specifying the increment number where the analysis was aborted. This attribute is read-only.

iterationNumber
An int specifying the iteration number where the analysis was aborted. This attribute is read-only.

attemptNumber
An int specifying the attempt number on which the analysis was aborted. This attribute is read-only.

category
String specifying the category of error. This attribute is read-only.

data
An *OdbAuxiliaryData* object.

description
String specifying the cause of the error. This attribute is read-only.

detailStrings
String specifying the exact nature of the problem. This attribute is read-only.

knowledgeItem
String specifying the exact reason for the error encountered. This attribute is read-only.

numberOfVariations
An int specifying the number of variations. This attribute is read-only.

stepNumber
An int specifying the step number on which the error was encountered. This attribute is read-only.

OdbAnalysisWarning object

The OdbAnalysisWarning object stores the description of different warnings encountered during the analysis.

Access

```
import visualization
session.odbData[name].diagnosticData.analysisWarnings[i]
```

Members

The OdbAnalysisWarning object has members that are similar to the *OdbAnalysisError* object.

OdbJobTime object

The OdbJobTime object stores the analysis time of a job.

Access

```
import visualization
session.odbData[name].diagnosticData.jobTime
```

Members

The OdbJobTime object has the following members:

systemTime A float specifying the systemtime for the analysis. This attribute is read-only.

userTime A float specifying the usertime for the analysis. This attribute is read-only.

wallclockTime A float specifying the wallclocktime for the analysis. This attribute is read-only.

OdbNumericalProblemSummary object

The OdbNumericalProblemSummary object stores the numerical problem summary of a job.

Access

```
import visualization
session.odbData[name].diagnosticData.numericalProblemSummary
```

Members

The OdbNumericalProblemSummary object has the following members:

convergedNegativeEigenValues
A boolean specifying whether negative eigenvalues converged during the analysis. This attribute is read-only.

convergedNumericalSingularities
A boolean specifying whether numerical singularities converged during the analysis. This attribute is read-only.

convergedZeroPivots
A boolean specifying whether pivot points converged during the analysis. This attribute is read-only.

numberOfZeroPivots
An int specifying the number of zero pivots. This attribute is read-only.

numberOfNumericalSingularities
An int specifying the number of numerical singularities. This attribute is read-only.

numberOfNegativeEigenValues
An int specifying the number of negative eigenvalues. This attribute is read-only.

OdbDiagnosticStep object

The OdbDiagnosticStep object stores step data.

Access

```
import visualization
session.odbData[name].diagnosticData.steps[i]
```

`extractData(...)`

This method creates a temporary *XYData* object, with increments on the x-axis and requested output on the y-axis.

Required arguments

incrementStatistics An enum specifying the requested output variable for the data table. Possible enum values are NUM_ATTEMPTS (the number of attempts), NUM_SDI (the number of severe discontinuity iterations), NUM_EQI (the number of equivalent iterations), NUM_ITERS (the number of iterations), STEP_TIME (the cumulative step time until that increment) or INC_SIZE (the step time for each increment).

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The OdbDiagnosticStep object has the following members:

activeXplStatus A Tuple of the status values. This attribute is read-only.

characteristicElementLength A float specifying the characteristic element length for the step. This attribute is read-only.

contactDiagnostics A repository of *OdbContactDiagnostic* objects.

explicitIncrementStatus A sequence of string specifying the explicit increment status. This attribute is read-only.

extrapolation A String specifying the method (Linear or logarithmic) used for extrapolation. This attribute is read-only.

incrementationScheme A String specifying the method of incrementation (Auto or fixed). This attribute is read-only.

incrementsCompleted An int specifying the number of completed increments. This attribute is read-only.

increments A repository of *OdbDiagnosticIncrement* objects.

initialTimeIncrement A float specifying the initial increment size for the step. This attribute is read-only.

isNlgeom A boolean specifying whether or not the effects of geometric nonlinearities are considered. This attribute is read-only.

isPerturbation A boolean specifying whether or not the step is a perturbation step. This attribute is read-only.

isStabilized A boolean specifying whether or not stabilization for the system in any form is considered. This attribute is read-only.

isRiks A boolean specifying whether the step is static riks. This attribute is read-only.

isUnsymm A boolean specifying whether the matrix storage is unsymmetric. This attribute is read-only.

matrixSolver A string specifying the method of solving (Direct or Iterative). This attribute is read-only.

maximumNumberOfIncrements An int specifying the maximum number of allowed increments in the step. This attribute is read-only.

maximumTimeIncrement A float specifying the size of the allowed maximum time increment in the step. This attribute is read-only.

minimumTimeIncrement A float specifying the size of the allowed minimum time increment in the step. This attribute is read-only.

name A string specifying the name of the step. This attribute is read-only.

number An int specifying the step number. This attribute is read-only.

numberOfContactDiagnostics An int specifying the number of contact diagnostics encountered. This attribute is read-only.

numberOfIncrements An int specifying the number of increments taken in the step to complete the solution. This attribute is read-only.

numberOfXplStatus An int specifying the number of the explicit status. This attribute is read-only.

stabilizeFactor
A float specifying the stabilize factor. This attribute is read-only.

stepTimeCompleted
A float specifying the time taken for the completion of the step. This attribute is read-only.

timePeriod
A float specifying the duration for the step. This attribute is read-only.

OdbAuxiliaryData object

The OdbAuxiliaryData object stores auxiliary data related to the steps in the analysis.

Access

```
        import visualization
        session.odbData[name].diagnosticData.analysisErrors[i].data[i]
        session.odbData[name].diagnosticData.analysisWarnings[i].data[i]

session.odbData[name].diagnosticData.steps[i].contactDiagnostics[i].data[i]
```

Members

The OdbAuxiliaryData object has the following members:

elementInstanceNames
A tuple consisting of element instance names. This attribute is read-only.

elementLabels
A tuple consisting of element label names. This attribute is read-only.

hasValues
A boolean specifying whether the OdbAuxiliaryData object has values in it. This attribute is read-only.

nodeInstanceNames
A tuple consisting of node instance names. This attribute is read-only.

nodeLables
A tuple consisting of node label names. This attribute is read-only.

OdbDiagnosticIncrement object

The OdbDiagnosticIncrement object.

Access

```
import visualization
session.odbData[name].diagnosticData.steps[i].increments[i]
```

Members

The OdbDiagnosticIncrement object has the following members:

attempts
A repository of *OdbDiagnosticAttempt* objects.

initialSize
A float specifying the size of the initial increment. This attribute is read-only.

isConverged
A boolean specifying whether the solution converged for the particular increment. This attribute is read-only.

numberOfAttempts
An int specifying the number of attempts for the particular increment. This attribute is read-only.

numberOfElementDiagnostics
An int specifying the number of element diagnostics encountered for the particular increment. This attribute is read-only.

size
A float specifying the size of the particular increment. This attribute is read-only.

stepTimeCompleted
A float specifying the amount of step time completed in the particular increment. This attribute is read-only.

OdbDiagnosticAttempt object

The OdbDiagnosticAttempt object.

Access

```
import visualization
session.odbData[name].diagnosticData.steps[i].increments[i].attempts[i]
```

Members

The OdbDiagnosticAttempt object has the following members:

autoStabilize
A boolean specifying the state of Auto-stablilization. This attribute is read-only.

isConverged
A boolean specifying the state of convergence for the attempt. This attribute is read-only.

isCutBack
A boolean specifying the state of cutback. This attribute is read-only.

needsReordering
A boolean specifying whether or not reordering is needed. This attribute is read-only.

numberOfCutbackDiagnostics
An int specifying the number of cutback diagnostics. This attribute is read-only.

numberOfIterations
An int specifying the number of iterations for the particular attempt. This attribute is read-only.

numberOfSevereDiscontinuityIterations
An int specifying the number of iterations with severe discontinuities This attribute is read-only.

size
A float specifying the size of the increment of the particular attempt. This attribute is read-only.

OdbContactDiagnostics object

The OdbDiagnosticContact object.

Access

```
import visualization
session.odbData[name].diagnosticData.steps[i].contactDiagnostics[i]
```

Members

The OdbContactDiagnostic object has the following members:

data
A repository of *OdbAuxiliaryData* objects. This attribute is read-only.

description
A string specifying the opening/overclosure status of the contact. This attribute is read-only.

detailStrings
A sequence of strings specifying the nature of each of the contact pair. This attribute is read-only.

type
A string specifying the type of contact initialization. This attribute is read-only.

defaultFormats
A string specifying the default format value. This attribute is read-only.

elementDescriptions
A string specifying the element description. This attribute is read-only.

nodeDescriptions
A string specifying the node description. This attribute is read-only.

OdbDisplayOptions object

The OdbDisplayOptions object stores the display options associated with an *OdbInstance* object. This object does not have a constructor. Abaqus creates the OdbDisplayOptions object when an *OdbInstance* object is created using the display options associated with the current viewport at the time of creation.

Access

```
import assembly
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions
import visualization
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions
import part
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions
```

Members

The OdbDisplayOptions object has the following members:

commonOptions

A *DGCommonOptions* object.

superimposeOptions

A *DGSuperimposeOptions* object.

contourOptions

A *DGContourOptions* object.

symbolOptions

A *DGSymbolOptions* object.

materialOrientationOptions

A *DGOrientationOptions* object.

displayBodyOptions

A *DGDisplayBodyOptions* object.

OptionArg object

The OptionArg object is used to store values and attributes as a temporary object to be associated with a viewCutOptions object. The OptionArg object has only a constructor command.

Access

```
import visualization
session.defaultOdbDisplay.viewCutOptions.aboveOptions
session.defaultOdbDisplay.viewCutOptions.belowOptions
session.defaultOdbDisplay.viewCutOptions.onOptions
session.viewports[name].layers[name].odbDisplay.viewCutOptions\
.aboveOptions
session.viewports[name].layers[name].odbDisplay.viewCutOptions\
.belowOptions
session.viewports[name].layers[name].odbDisplay.viewCutOptions\
.onOptions
session.viewports[name].odbDisplay.viewCutOptions.aboveOptions
session.viewports[name].odbDisplay.viewCutOptions.belowOptions
session.viewports[name].odbDisplay.viewCutOptions.onOptions
```

OptionArg(...)

This method creates an OptionArg object.

Path

visualization.OptionArg

Required arguments

None.

Optional arguments

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is WIREFRAME.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is FEATURE.

NONE can be used only when *renderStyle*=SHADED.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the undeformed plot when *renderStyle*=WIREFRAME or HIDDEN. The default value is "Green".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the undeformed plot when *renderStyle*=FILLED or SHADED. The default value is "Black".

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

fillColor A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "Green".

translucency A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor A Float specifying the translucency factor when *translucency* = ON. Possible values are 0.0 ≤*translucencyFactor*≤ 1.0. The default value is 0.3.

Return value

An OptionArg object.

Exceptions

RangeError.

Members

The OptionArg object has members with the same names and descriptions as the arguments to the *OptionArg* method.

DGOrientationOptions object

The DGOrientationOptions object stores values and attributes associated with a material orientation plot. The DGOrientationOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.materialOrientationOptions* member when a display group instance is created, using values from *odbDisplay.materialOrientationOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.materialOrientationOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.materialOrientationOptions
```

Members

The DGOrientationOptions object can have the following members:

showAxis1

A Boolean specifying whether axis 1 of the material orientation triad should be displayed. The default value is ON.

showAxis2

A Boolean specifying whether axis 2 of the material orientation triad should be displayed. The default value is ON.

showAxis3

A Boolean specifying whether axis 3 of the material orientation triad should be displayed. The default value is ON.

symbolSize

A Float specifying the size of the material orientation triad. The default value is 6.0.

lineThickness

A SymbolicConstant specifying the thickness of the material orientation triad. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

orientation

A SymbolicConstant specifying the orientation used for composites. Possible values are PLY and LAYUP. The default value is PLY.

arrowheadStyle

A SymbolicConstant specifying the arrowhead style for the material orientation triad. Possible values are NONE, FILLED, and WIRE. The default value is NONE.

scaleMode

A SymbolicConstant specifying the scaling basis for the material orientation triad. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

axis1Color

A String specifying the color of axis 1 of the material orientation triad. The default value is "Cyan".

axis2Color

A String specifying the color of axis 2 of the material orientation triad. The default value is "Yellow".

axis3Color

A String specifying the color of axis 3 of the material orientation triad. The default value is "Red".

StreamOptions object

The StreamOptions object stores values and attributes associated with a stream plot. The StreamOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.streamOptions* member when you import the Visualization module. Abaqus creates a *StreamOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.streamOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

StreamOptions objects are accessed in one of two ways:

- The default stream options. These settings are used as defaults when other *streamOptions* members are created. These settings can be set to customize user preferences.
- The stream options associated with a particular viewport.

Access

```
import visualization
session.defaultOdbDisplay.streamOptions
session.viewports[name].layers[name].odbDisplay.streamOptions
session.viewports[name].odbDisplay.streamOptions
```

`setValues(...)`

This method modifies the StreamOptions object.

Required arguments

None.

Optional arguments

colorMethod A SymbolicConstant specifying the method for coloring stream lines. Possible values are UNIFORM, SPECTRUM, and CONTINUOUS. The default value is UNIFORM.

uniformColor A String specifying uniform color of the stream lines. The default value is "#FFFF00".

lineThickness A Float specifying the stream line thickness. The default value is 6.0.

showArrow A Boolean specifying whether to show arrows. The default value is OFF.

numArrows An Int specifying the number of arrows on each stream line. The default value is 10.

Return value

`None.`

Exceptions

None.

Members

The StreamOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

DGSuperimposeOptions object

The DGSuperimposeOptions object stores values and attributes associated with the undeformed shape when the display contains both the deformed shape and the undeformed shape. The DGSuperimposeOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.superimposeOptions* member when a display group instance is created, using values from *odbDisplay.superimposeOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.superimposeOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.superimposeOptions
```

Members

The DGSuperimposeOptions object can have the following members:

renderStyle

A SymbolicConstant specifying the render style of the plot. Possible values are WIREFRAME, FILLED, HIDDEN, and SHADED. The default value is WIREFRAME.

visibleEdges

A SymbolicConstant specifying which edges to plot. Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE. The default value is FEATURE.

NONE can be used only when *renderStyle*=SHADED.

edgeLineStyle

A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeLineThickness

A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

colorCodeOverride

A Boolean specifying whether to allow color coded items in the output database to override the edge and fill color settings. The default value is ON.

elemLabels

A Boolean specifying whether to plot the element labels. The default value is OFF.

faceLabels

A Boolean specifying whether to plot the face labels. The default value is OFF.

nodeLabels

A Boolean specifying whether to plot the node labels. The default value is OFF.

nodeSymbols

A Boolean specifying whether to plot the node symbols. The default value is OFF.

nodeSymbolType

A SymbolicConstant specifying the node symbol types. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER

The default value is HOLLOW_CIRCLE.

nodeSymbolSize

A SymbolicConstant specifying the node symbol size. Possible values are SMALL, MEDIUM, and LARGE. The default value is SMALL.

elementShrink

A Boolean specifying whether elements are displayed in a shrunk format. The default value is OFF.

elementShrinkFactor

An Int specifying the percentage to shrink the elements when *elementShrink*=ON. Possible values are $0 \leq$ *elementShrinkPercentage* ≤ 90. The default value is 5.

coordinateScale

A Boolean specifying whether to scale coordinates. The default value is OFF.

normals

A Boolean specifying whether to draw arrows that indicate the directions of element and surface normals. The default value is OFF.

normalDisplay

A SymbolicConstant specifying whether to draw element normals and surface normals. Possible values are ELEMENT and SURFACE. The default value is ELEMENT.

normalArrowLength

A SymbolicConstant specifying the length of the normal arrows. Possible values are SHORT, MEDIUM, and LONG. The default value is MEDIUM.

normalLineThickness

A SymbolicConstant specifying the thickness of the normal arrows. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

normalArrowheadStyle

A SymbolicConstant specifying the arrowhead style of the normal arrows. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

translucency

A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor

A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

edgeColorWireHide

A String specifying the color to be used to plot the edges of the model when *renderStyle*=WIREFRAME or HIDDEN. The default value is "Green".

edgeColorFillShade

A String specifying the color to be used to plot the edges of the model when *renderStyle*=FILLED or SHADED. The default value is "Black".

fillColor

A String specifying the color to be used to fill elements when *renderStyle*=FILLED or SHADED. The default value is "Green".

labelFont

A String specifying the label font to be used for all model labels. The default value is "-*-courier-medium-r-normal-*-*-120-*-*-m-*-*-*".

elemLabelColor

A String specifying the color to be used to plot the element labels. The default value is "Cyan".

faceLabelColor

A String specifying the color to be used to plot the face labels. The default value is "Red".

nodeLabelColor

A String specifying the color to be used to plot the node labels. The default value is "Yellow".

nodeSymbolColor

A String specifying the color to be used to plot the node symbols. The default value is "Yellow".

faceNormalColor

A String specifying the color to be used to plot the normal to a nonbeam element or to a surface. The default value is "Red".

beamN1Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_1}$-direction. The default value is "Blue".

beamN2Color

A String specifying the color to be used to plot an arrow along the beam $\boldsymbol{n_2}$-direction. The default value is "Red".

beamTangentColor

A String specifying the color to be used to plot an arrow along the tangent to a beam. The default value is "White".

coordinateScaleFactors

A tuple of three Floats specifying the coordinate scaling in each of the three coordinate directions when *coordinateScale*=ON. The default value is (1, 1, 1).

DGSymbolOptions object

The DGSymbolOptions object stores values and attributes associated with a symbol plot. The DGSymbolOptions object has no constructor command. Abaqus creates an *odbDisplayOptions.symbolOptions* member when a display group instance is created, using values from *odbDisplay.symbolOptions*.

Access

```
session.viewports[name].assemblyDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].assemblyDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].odbDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].layers[name].partDisplay\
.displayGroupInstances[name].odbDisplayOptions.symbolOptions
session.viewports[name].odbDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
session.viewports[name].partDisplay.displayGroupInstances[name]\
.odbDisplayOptions.symbolOptions
```

Members

The DGSymbolOptions object can have the following members:

vectorLineThickness
A SymbolicConstant specifying the vector line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

vectorColorMethod
A SymbolicConstant specifying the vector color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

tensorColorMethod
A SymbolicConstant specifying the tensor color method. Possible values are UNIFORM and SPECTRUM. The default value is SPECTRUM.

vectorArrowheadStyle
A SymbolicConstant specifying the vector arrowhead style. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

arrowSymbolSize
An Int specifying the length of vector and tensor symbols. The default value is 6.

vectorIntervalNumber
An Int specifying the number of color intervals for vector symbols. The default value is 12.

symbolDensity
A Float specifying the factor for randomized sampling. The default value is 1.0.

constantLengthArrows
A Boolean specifying whether to use constant-length arrows for vector symbols. The default value is OFF.

tensorIntervalNumber
An Int specifying the number of color intervals for tensor symbols. The default value is 12.

tensorLineThickness
A SymbolicConstant specifying the line thickness of the tensor symbols. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

tensorArrowheadStyle
A SymbolicConstant specifying the arrowhead style of the tensor symbols. Possible values are NONE, FILLED, and WIRE. The default value is WIRE.

numberFormatT
A SymbolicConstant specifying the number format for tensor. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

numberFormatV
A SymbolicConstant specifying the number format for vector. Possible values are SCIENTIFIC, FIXED, and ENGINEERING. The default value is SCIENTIFIC.

arrowScaleMode
A SymbolicConstant specifying the arrow scaling mode. Possible values are MODEL_SIZE and SCREEN_SIZE. The default value is MODEL_SIZE.

drawLabelT
A Boolean specifying whether to draw tensor labels. The default value is OFF.

drawLabelV
A Boolean specifying whether to draw vector labels. The default value is OFF.

numDigitsT
An Int specifying the number of digits in the tensor label. The default value is 2.

numDigitsV
An Int specifying the number of digits in the vector label. The default value is 2.

vectorColor
A String specifying the vector color. The default value is "Red".

vectorColorSpectrum
A String specifying the vector color spectrum name. The default value is "Rainbow".

tensorColorSpectrum
A String specifying the tensor color spectrum name. The default value is "Rainbow".

textColorT
A String specifying the text color for tensor. The default value is "Yellow".

textColorV
A String specifying the text color for vector. The default value is "Yellow".

textFontT
A String specifying the text font for tensor. The default value is "verdana".

textFontV
A String specifying the text font for vector. The default value is "verdana".

tensorMaxPrinColor
A String specifying the color of the maximum principal tensor symbols. The default value is "Red".

tensorMinPrinColor
A String specifying the color of the minimum principal tensor symbols. The default value is "Cyan".

tensorMidPrinColor
A String specifying the color of the intermediate principal tensor symbols. The default value is "Yellow".

tensorSelectedPrinColor A String specifying the color of the selected principal tensor symbols. The default value is "Red".

ViewCutOptions object

The ViewCutOptions object stores values and attributes associated with a view cut plot. The ViewCutOptions object has no constructor command. Abaqus creates a *defaultOdbDisplay.viewCutOptions* member when you import the Visualization module. Abaqus creates an *viewCutOptions* member when it creates the *OdbDisplay* object, using the values from *defaultOdbDisplay.viewCutOptions*. Abaqus creates the *odbDisplay* member when a viewport is created, using the values from *defaultOdbDisplay*.

ViewCutOptions objects are accessed in one of two ways:

- The default view cut options. These settings are used as defaults when other *viewCutOptions* members are created. These settings can be set to customize user preferences.
- The view cut options associated with a particular viewport.

Access

```
import visualization
session.defaultOdbDisplay.viewCutOptions
session.viewports[name].layers[name].odbDisplay.viewCutOptions
session.viewports[name].odbDisplay.viewCutOptions
```

`setValues(...)`

This method modifies the ViewCutOptions object.

Required arguments

None.

Optional arguments

options

A ViewCutOptions object from which values are to be copied. If other arguments are also supplied to `setValues`, they will override the values in *options*. The default value is `None`.

belowOptions

`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model below the cut. The default value is `None`.

useBelowOptions

A Boolean specifying whether to use the options defined for displaying the model below the cut. The default value is OFF.

onOptions

`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model on the cut. The default value is `None`.

useOnOptions

A Boolean specifying whether to use the options defined for displaying the model on the cut. The default value is OFF.

aboveOptions

`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model above the cut. The default value is `None`.

useAboveOptions
A Boolean specifying whether to use the options defined for displaying the model above the cut. The default value is OFF.

freeBodyCutThru
A SymbolicConstant specifying the domain through which the free body cuts. Possible values are CURRENT_DISPLAY_GROUP and WHOLE_MODEL. The default value is CURRENT_DISPLAY_GROUP.

freeBodyStepThru
A SymbolicConstant specifying the domain through which the free body steps. Possible values are ACTIVE_CUT_RANGE and PREDEFINED_PATH. The default value is ACTIVE_CUT_RANGE.

numCutFreeBody
An Int specifying number of free bodies per view cut. The default value is 1.

displaySlicing
A Boolean specifying whether to display slicing. The default value is OFF.

slicingAtPathNodes
A Boolean specifying whether to slice at path nodes. The default value is OFF.

freeBodySumOnPath
A Boolean specifying whether to put the summation point at the path. The default value is ON.

cutFreeBodyMin
A Float specifying free body minimum value. The default value is 0.0.

cutFreeBodyMax
A Float specifying free body maximum value. The default value is 0.0.

showHeatFlowRate
A SymbolicConstant specifying whether to show the heat flow rate when available. Possible values are ON and OFF. The default value is ON.

summationLoc
A SymbolicConstant specifying the summation location for the free body cut. Possible values are CENTROID and SPECIFY. The default value is CENTROID.

componentResolution
A SymbolicConstant specifying the component resolution choice for the free body cut. Possible values are NORMAL_TANGENTIAL and CSYS. The default value is NORMAL_TANGENTIAL.

csysName
The SymbolicConstant GLOBAL or a String specifying the coordinate system name for the free body cut's component resolution. The default value is GLOBAL.

pathName
A String specifying the name of the path along which slicing occurs. The default value is an empty string.

summationPoint
A sequence of three Floats specifying the summation point for the free body cut. The default value is (0, 0, 0).

yAxis
A sequence of three Floats specifying the Y axis for free body component resolution. The default value is (0, 1, 0).

Return value
`None.`

Exceptions

RangeError.

Members

The ViewCutOptions object can have the following members:

useBelowOptions
A Boolean specifying whether to use the options defined for displaying the model below the cut. The default value is OFF.

useOnOptions
A Boolean specifying whether to use the options defined for displaying the model on the cut. The default value is OFF.

useAboveOptions
A Boolean specifying whether to use the options defined for displaying the model above the cut. The default value is OFF.

numCutFreeBody
An Int specifying number of free bodies per view cut. The default value is 1.

displaySlicing
A Boolean specifying whether to display slicing. The default value is OFF.

slicingAtPathNodes
A Boolean specifying whether to slice at path nodes. The default value is OFF.

freeBodySumOnPath
A Boolean specifying whether to put the summation point at the path. The default value is ON.

cutFreeBodyMin
A Float specifying free body minimum value. The default value is 0.0.

cutFreeBodyMax
A Float specifying free body maximum value. The default value is 0.0.

freeBodyCutThru
A SymbolicConstant specifying the domain through which the free body cuts. Possible values are CURRENT_DISPLAY_GROUP and WHOLE_MODEL. The default value is CURRENT_DISPLAY_GROUP.

freeBodyStepThru
A SymbolicConstant specifying the domain through which the free body steps. Possible values are ACTIVE_CUT_RANGE and PREDEFINED_PATH. The default value is ACTIVE_CUT_RANGE.

showHeatFlowRate
A SymbolicConstant specifying whether to show the heat flow rate when available. Possible values are ON and OFF. The default value is ON.

summationLoc
A SymbolicConstant specifying the summation location for the free body cut. Possible values are CENTROID and SPECIFY. The default value is CENTROID.

componentResolution
A SymbolicConstant specifying the component resolution choice for the free body cut. Possible values are NORMAL_TANGENTIAL and CSYS. The default value is NORMAL_TANGENTIAL.

belowOptions
`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model below the cut. The default value is `None`.

onOptions

`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model on the cut. The default value is `None`.

aboveOptions

`None` or an *OptionArg* object specifying values to be used for defining the options applicable on the model above the cut. The default value is `None`.

csysName

The SymbolicConstant GLOBAL or a String specifying the coordinate system name for the free body cut's component resolution. The default value is GLOBAL.

pathName

A String specifying the name of the path along which slicing occurs. The default value is an empty string.

summationPoint

A tuple of three Floats specifying the summation point for the free body cut. The default value is (0, 0, 0).

yAxis

A tuple of three Floats specifying the Y axis for free body component resolution. The default value is (0, 1, 0).

DetailPlotOptions object

The DetailPlotOptions object stores values and attributes associated with a *Viewport* object. The DetailPlotOptions object has no constructor command. Abaqus creates the *detailPlotOptions* member whenever a *Viewport* is created.

Access

```
session.viewports[name].detailPlotOptions
```

Members

The DetailPlotOptions object has the following member:

plyStackPlotOptions
A *PlyStackPlotOptions* object.

PlyStackPlotOptions object

The PlyStackPlotOptions object stores values and attributes associated with a*Viewport* object. The PlyStackPlotOptions object has no constructor command. Abaqus creates the *detailPlotOptions.plyStackPlotPlotOptions* member whenever a *Viewport* is created.

Access

```
session.viewports[name].detailPlotOptions.plyStackPlotOptions
```

`setValues(...)`

This method modifies the PlyStackPlotOptions object.

Required arguments

None.

Optional arguments

renderStyle
A SymbolicConstant specifying how the plies in the viewport are rendered. Possible values are WIRE_FRAME, FILLED, and SHADED. The default value is SHADED.

showEdges
A Boolean specifying whether to draw the edges for the plies. The default value is OFF.

edgeStyle
A SymbolicConstant specifying the edge line style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

edgeThickness
A SymbolicConstant specifying the edge line thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

plyDisplay
A SymbolicConstant specifying whether to display the plies in an unsymmetric staircase fashion or symmetric staircase fashion. Possible values are UNSYMMETRIC and SYMMETRIC. The default value is UNSYMMETRIC.

startLayer
An Int specifying the start ply. The default value is 1.

numLayers
An Int specifying the number of plies to show. The default value is 30.

evenNumPlyColor
A String specifying the ply color for the even numbered plies. The numbering is based on the displayed plies and not on the actual ply number in the layup. The default value is "#A2B3FF".

oddNumPlyColor
A String specifying the ply color for the odd numbered plies. The numbering is based on the displayed plies and not on the actual ply number in the layup. The default value is "#ECC9AD".

showFibers
A Boolean specifying whether to show the fibers. The default value is ON.

fiberColor
A String specifying the fiber color. The default value is "Red".

fiberStyle
A SymbolicConstant specifying the fiber style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

fiberThickness
A SymbolicConstant specifying the fiber thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

fiberSpacing
A Float specifying the spacing between the fibers. The default value is 0.1.

showReferencePlane
A Boolean specifying whether to show the reference surface. The default value is OFF.

referenceSurfaceColor
A String specifying the reference surface color. The default value is "White".

translucency
A Boolean specifying whether to set translucency. The default value is OFF.

translucencyFactor
A Float specifying the translucency factor when *translucency*=ON. Possible values are $0.0 \leq$ *translucencyFactor* ≤ 1.0. The default value is 0.3.

translucencySort
A Boolean specifying whether to use depth sorting. The default value is OFF.

showReferenceOutline
A Boolean specifying whether to show the reference outline. The default value is OFF.

referenceOutlineColor
A String specifying the reference outline color. The default value is "Red".

referenceOutlineStyle
A SymbolicConstant specifying the reference outline style. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

referenceOutlineThickness
A SymbolicConstant specifying the reference outline thickness. Possible values are VERY_THIN, THIN, MEDIUM, and THICK. The default value is VERY_THIN.

allLabelsFont
A String specifying the font for all the labels. The default value is "-*-verdana-medium-r-normal-*-*-80-*-*-p-*-*-*" on Windows and "-*-verdana-medium-r-normal-*-*-90-*-*-p-*-*-*" on Linux platforms. The default value is an empty string.

showMaterialNames
A Boolean specifying whether to show the material names. The default value is OFF.

materialNamesColor
A String specifying the material names text color. The default value is "Green".

showOrientationNames
A Boolean specifying whether to show the orientation names or the fiber angles. The default value is OFF.

orientationNamesColor
A String specifying the orientation names fiber angles text color. The default value is "Orange".

showStateBlock	A Boolean specifying whether to show the state block. The default value is ON.
stateBlockColor	A String specifying the state block text color. The default value is "White".
showPlyNames	A Boolean specifying whether to show the ply names. The default value is ON.
plyNamesColor	A String specifying the ply names text color. The default value is "Yellow".
showThicknessLabels	A Boolean specifying whether to show the thickness labels. The default value is ON.
thicknessLabelsColor	A String specifying the thickness labels text color. The default value is "Red".
showIntPoints	A Boolean specifying whether to show the through thickness integration points. The default value is OFF.
intPointsColor	A String specifying the through thickness integration points color. The default value is "Blue".
sizeX	A Float specifying the size of the ply in the X-direction. The default value is 1.5.
sizeY	A Float specifying the size of the ply in the Y-direction. The default value is 1.0.
sizeZ	A Float specifying the size of the ply in the Z-direction. The default value is 0.8.

Return value

`None.`

Exceptions

None.

Members

The PlyStackPlotOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

MdbData object

The MdbData object has no constructor. Abaqus creates an MdbData object when a cae file is opened or a new model is created. There is one MdbData for each model in session. MdbData is updated when it is displayed in a viewport.

Access

```
import visualization
session.mdbData[name]
```

Members

The MdbData object has the following members:

stepPeriods

A tuple of (String, Float) tuples specifying the stepName and the stepPeriod.

steps

A repository of *MdbDataStep* objects specifying the list of steps. The repository is read-only.

instances

A repository of *MdbDataInstance* objects specifying the list of instances. The repository is read-only.

MdbDataFrame object

The MdbDataFrame object. There is one and only MdbDataFrame in a *MdbDataStep*.

Access

```
import visualization
session.mdbData[name].steps[i].frames[i]
```

Members

The MdbDataFrame object has no members.

MdbDataInstance object

The MdbDataInstance object instance data. It corresponds to same named part instance with a mesh in the cae model.

Access

```
import visualization
session.mdbData[name].instances[i]
```

Members

The MdbDataInstance object has the following member:

name A String specifying the instance name. This attribute is read-only.

MdbDataStep object

The MdbDataStep object.It corresponds to same named step in the cae model.

Access

```
import visualization
session.mdbData[name].steps[i]
```

Members

The MdbDataStep object has the following member:

frames

A *MdbDataFrameArray* object specifying the list of frames. The list is read-only. There is only one frame in a step.

Plug-in registration commands

Plug-in commands register kernel and GUI plug-ins in either the **Plug-ins** menu or in a toolbox.
Plug-in registration commands for accessing both kernel and GUI plug-ins are located in the *Abaqus GUI Toolkit Reference Guide*.

Predefined field commands

A PredefinedField object stores the non-propagating data of a predefined field as well as a number of instances of the corresponding PredefinedFieldState object.
The PredefinedFieldState object stores the propagating data of the predefined field in a single step. A specific type of PredefinedField object and a specific type of PredefinedFieldState object are designed for each type of predefined field. Instances of the PredefinedFieldState object are created and deleted internally by its corresponding PredefinedField object.

In this section:

- *PredefinedField object*
- *Field object*
- *FieldState object*
- *FluidCavityPressure object*
- *FluidElectricPotential object*
- *IMAField object*
- *IMARegion object*
- *InitialState object*
- *IonConcentration object*
- *KinematicHardening object*
- *MaterialAssignment object*
- *PredefinedFieldState object*
- *SolidElectricPotential object*
- *Stress object*
- *Temperature object*
- *TemperatureState object*
- *Velocity object*

PredefinedField object

The PredefinedField object is the base object for the objects in the predefined field repository. The methods and members of the PredefinedField object are common to all objects derived from PredefinedField.

An instance of any PredefinedField object can be obtained through the predefined field repository of the *Model* object. An instance of any PredefinedFieldState object can be obtained through the predefined field repository of the *Step* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

move(...)

This method moves a specific *PredefinedFieldState* object from one step to a different step.

Required arguments

fromStepName	A String specifying the name of the step from which the *PredefinedFieldState* object is moved.
toStepName	A String specifying the name of the step to which the *PredefinedFieldState* object is moved.

Optional arguments

None.

Return value

None.

Exceptions

TextError.

resume()

This method resumes the predefined field that was previously suppressed.

Arguments

None.

Return value

None.

Exceptions

None.

suppress()

This method suppresses the predefined field.

Arguments

None.

Return value

`None.`

Exceptions

None.

delete(...)

This method allows you to delete existing fields.

Required arguments

indices A sequence of Ints specifying the index of each field to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The PredefinedField object can have the following members:

name A String specifying the repository key.

region A *Region* object specifying the region to which the predefined field is applied. *Region* is ignored if the predefined field has an *instances* member available. *Region* is also ignored if the predefined field has a *distributionType* member available, and *distributionType*=FROM_FILE or FROM_FILE_AND_USER_DEFINED.

Field object

The Field object stores the data for field predefined fields.

The Field object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

Field(...)

This method creates a Field object.

Path

```
mdb.models[name].Field
```

Required arguments

name A String specifying the repository key.

createStepName A String specifying the name of the step in which the predefined field is created.

region A *Region* object specifying the region to which the predefined field is applied. *Region* is ignored if the predefined field has a *distributionType* member available, and *distributionType*=FROM_FILE.

Optional arguments

outputVariable A String specifying the scalar nodal output variable that will be read from an output database and used to initialize a specified predefined field. This argument is a required argument if *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

fieldVariableNum An Int specifying the field variable number.

distributionType A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, USER_DEFINED, FROM_FILE, FIELD, FROM_FILE_AND_USER_DEFINED, and DISCRETE_FIELD. The default value is UNIFORM.

crossSectionDistribution A SymbolicConstant specifying how the predefined field is distributed over the cross-section of the region. Possible values are

- CONSTANT_THROUGH_THICKNESS
- GRADIENTS_THROUGH_SHELL_CS
- GRADIENTS_THROUGH_BEAM_CS
- POINTS_THROUGH_SECTION

The default value is CONSTANT_THROUGH_THICKNESS.

field

A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this predefined field. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

amplitude

A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the predefined field has no amplitude reference. The default value is UNSET.



Note:

amplitude should be given only if it is valid for the specified step.

fileName

A String specifying the name of the file from which the Field values are to be read when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

beginStep

An Int specifying the first step from which Field values are to be read or the SymbolicConstant FIRST_STEP or LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

beginIncrement

An Int specifying the first increment of the step set in *beginStep* or the SymbolicConstants STEP_START or STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endStep

An Int specifying the last step from which Field values are to be read or the SymbolicConstants FIRST_STEP and LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endIncrement

An Int specifying the last increment of the step set in *endStep* or the SymbolicConstants STEP_START and STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

interpolate

A SymbolicConstant specifying whether to interpolate a field read from an output database or results file. Possible values are OFF, ON, or MIDSIDE_ONLY. The default value is OFF.

magnitudes

A Sequence of Doubles specifying the Field values when *distributionType*=UNIFORM or FIELD. The value of the *magnitudes* argument is a function of the *crossSectionDistribution* argument, as shown in the following list:

- If *crossSectionDistribution*=CONSTANT_THROUGH_THICKNESS, *magnitudes* is a Double specifying the Field.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_SHELL_CS, *magnitudes* is a sequence of Doubles specifying the mean value and the gradient in the thickness direction.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_BEAM_CS, *magnitudes* is a sequence of Doubles specifying the mean value, the gradient in the N1 direction, and the gradient in the N2 direction.

- If *crossSectionDistribution*=POINTS_THROUGH_SECTION, *magnitudes* is a sequence of Doubles specifying the Field at each point.

Return value

A Field object.

Exceptions

None.

move(...)

This method moves the *FieldState* object from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the *PredefinedFieldState* is moved.

toStepName A String specifying the name of the step to which the *PredefinedFieldState* is moved.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

setValues(...)

This method modifies the data for an existing Field object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Field* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing Field object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the predefined field is modified.

Optional arguments

The optional arguments to `setValuesInStep` are the same as the optional arguments to the *Field* method, except for the *distributionType* and *crossSectionDistribution* arguments.

Return value

`None.`

Exceptions

None.

Members

The Field object can have the following members:

name A String specifying the repository key.

distributionType A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, USER_DEFINED, FROM_FILE, FIELD, FROM_FILE_AND_USER_DEFINED, and DISCRETE_FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this predefined field. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the predefined field is applied. *Region* is ignored if the predefined field has an *instances* member available. *Region* is also ignored if the predefined field has a *distributionType* member available, and *distributionType*=FROM_FILE or FROM_FILE_AND_USER_DEFINED.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=FIELD
- **FIELD*

FieldState object

The FieldState object stores the propagating data of a field in a step. One instance of this object is created internally by the *Field object* for each step.

The FieldState object has no constructor or methods.

The FieldState object is derived from the *PredefinedFieldState object*.

Access

```
import load
mdb.models[name].steps[name].predefinedFieldStates[name]
```

Members

The FieldState object can have the following members:

outputVariable A String specifying the scalar nodal output variable that will be read from an output database and used to initialize a specified predefined field. This argument is a required argument if *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

fileNameState A SymbolicConstant specifying the propagation state of the *fileName* member. Possible values are UNSET, SET, and UNCHANGED.

beginStep A SymbolicConstant or an Int specifying the first step from which field values are to be read. This argument is valid only when *distribution*=FROM_FILE or *distribution*=FROM_FILE_AND_USER_DEFINED. Possible values are FIRST_STEP, LAST_STEP, and NONE. The default value is NONE.

beginStepState A SymbolicConstant specifying the propagation state of the *beginStep* member. Possible values are UNSET, SET, and UNCHANGED.

beginIncrement `None` or an Int specifying the first increment of the step set in *beginStep* or the SymbolicConstants STEP_START or STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

beginIncrementState A SymbolicConstant specifying the propagation state of the *beginIncrement* member. Possible values are UNSET, SET, and UNCHANGED.

endStep `None` or an Int specifying the last step from which field values are to be read or the SymbolicConstants FIRST_STEP and LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endStepState A SymbolicConstant specifying the propagation state of the *endStep* member. Possible values are UNSET, SET, and UNCHANGED.

endIncrement `None` or an Int specifying the last increment of the step set in *endStep* or the SymbolicConstants STEP_START and STEP_END. This argument is valid only when

distributionType=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endIncrementState

A SymbolicConstant specifying the propagation state of the *endIncrement* member. Possible values are UNSET, SET, and UNCHANGED.

amplitudeState

A SymbolicConstant specifying the propagation state of the *amplitudeState* member. Possible values are UNSET, SET, and UNCHANGED.

fileName

A String specifying the name of the file from which the field values are to be read when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

amplitude

The SymbolicConstant UNSET or a String specifying the name of the amplitude reference. The SymbolicConstant UNSET should be used if the predefined field has no amplitude reference. The default value is UNSET.



Note: *amplitude* should be given only if it is valid for the specified step.

magnitudesState

A tuple of SymbolicConstants specifying the propagation state of each item of the *magnitudes* member. Possible values are UNSET, SET, and UNCHANGED.

magnitudes

A tuple of Floats specifying the field values when *distributionType*=UNIFORM or *distributionType*=FIELD. The value of the *magnitudes* argument is a function of the *crossSectionDistribution* argument, as shown in the following list:

- If *crossSectionDistribution*=CONSTANT_THROUGH_THICKNESS, *magnitudes* is a Double specifying the field.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_SHELL_CS, *magnitudes* is a sequence of Doubles specifying the mean value and the gradient in the thickness direction.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_BEAM_CS, *magnitudes* is a sequence of Doubles specifying the mean value, the gradient in the N1 direction, and the gradient in the N2 direction.
- If *crossSectionDistribution*=POINTS_THROUGH_SECTION, *magnitudes* is a sequence of Doubles specifying the field at each point.

status

A SymbolicConstant specifying the propagation state of the *PredefinedFieldState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- DEACTIVATED_TO_INITIAL
- NO_LONGER_ACTIVE
- RESET_TO_INITIAL

- TO_BE_COMPUTED
- PROPAGATED_FROM_COMPUTED
- BUILT_INTO_BASE_STATE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

This member exists in all *PredefinedFieldState* objects, but different predefined fields use different subsets of the entire list of possible values depending on propagation rules.

FluidCavityPressure object

The FluidCavityPressure object stores the data for initial fluid cavity pressures. The base class*region* argument can not be specifed with this object.

The FluidCavityPressure object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

FluidCavityPressure(...)

This method creates a FluidCavityPressure object.

Path

```
mdb.models[name].FluidCavityPressure
```

Required arguments

name	A String specifying the repository key.
fluidCavity	A String specifying the name of a Fluid Cavity Interaction.
fluidPressure	A Float specifying the initial fluid pressure.

Optional arguments

None.

Return value

A FluidCavityPressure object.

Exceptions

None.

setValues(...)

This method modifies the FluidCavityPressure object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidCavityPressure* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FluidCavityPressure object has members with the same names and descriptions as the arguments to the *FluidCavityPressure* method.

In addition, the FluidCavityPressure object can have the following member:

region A *Region* object on which the *fluidCavity* interaction is specified.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=FLUID PRESSURE

FluidElectricPotential object

The FluidElectricPotential object stores the data for the fluid electric potential predefined field.

The FluidElectricPotential object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

FluidElectricPotential(...)

This method creates a FluidElectricPotential predefined field object.

Path

```
mdb.models[name].FluidElectricPotential
```

Required arguments

name	A String specifying the repository key.
region	A *Region* object specifying the region to which the predefined field is applied.

Optional arguments

distributionType	A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.
field	A String specifying the name of the AnalyticalField or DiscreteField object associated with this predefined field. The field argument applies only when *distributionType* is FIELD or *distributionType* is DISCRETE_FIELD. The default value is an empty string.
value	A Float specifying the initial value of fluid electric potential.

Return value

A FluidElectricPotential object.

Exceptions

None.

setValues(...)

This method modifies the FluidElectricPotential object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FluidElectricPotential* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The FluidElectricPotential object has members with the same names and descriptions as the arguments to the *FluidElectricPotential* method.

Corresponding analysis keywords

- **INSTANCE*

IMAField object

A IMAField is an object used to define material instance name volume fractions for the *MaterialAssignment* predefined field.

Access

```
import load
mdb.models[name].predefinedFields[name].fieldList
```

Members

The IMAField object can have the following members:

region

A *Region* object specifying the sub-region of the selected part instance to which the volume fractions will be applied.

discFieldList

A tuple of Strings specifying the name of the discrete fields that contain the volume fraction data. The length of the tuple corresponds to the number of material instance names, as established by the assigned Eulerian section.

IMARegion object

A IMARegion is an object used to define material instance name volume fractions for the *MaterialAssignment* predefined field.

Access

```
import load
mdb.models[name].predefinedFields[name].assignmentList
```

Members

The IMARegion object can have the following members:

region

A *Region* object specifying the sub-region of the selected part instance to which the volume fractions will be applied.

fractionList

A tuple of Floats specifying the volume fractions, per material instance name. The length of the tuple corresponds to the number of material instance names, as established by the assigned Eulerian section.

InitialState object

The InitialState object stores the data for an initial state predefined field.

The InitialState object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

InitialState(...)

This method creates an InitialState predefined field object.

Path

```
mdb.models[name].InitialState
```

Required arguments

name A String specifying the repository key.

instances A *PartInstanceArray* object specifying the instances to which the predefined field is applied.

fileName A String specifying the name of the job that generated the initial state data.

Optional arguments

endStep The SymbolicConstant LAST_STEP or an Int specifying the step from which the initial state values are to be read or the SymbolicConstant LAST_STEP. The default value is LAST_STEP.

endIncrement The SymbolicConstant STEP_END or an Int specifying the increment, interval or iteration of the step set in *endStep* or the SymbolicConstant STEP_END. The default value is STEP_END.

updateReferenceConfiguration A Boolean specifying whether to update the reference configuration based on the import data. The default value is OFF.

Return value

An InitialState object.

Exceptions

None.

setValues(...)

This method modifies the InitialState object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *InitialState* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The InitialState object has members with the same names and descriptions as the arguments to the *InitialState* method.

Corresponding analysis keywords

- **INSTANCE*

IonConcentration object

The IonConcentration object stores the data for an ion concentration predefined field.

The IonConcentration object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

IonConcentration(...)

This method creates an IonConcentration predefined field object.

Path

```
mdb.models[name].IonConcentration
```

Required arguments

name	A String specifying the repository key.
region	A *Region* object specifying the region to which the predefined field is applied.

Optional arguments

distributionType	A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.
field	A String specifying the name of the AnalyticalField or DiscreteField object associated with this predefined field. The field argument applies only when *distributionType* is FIELD or *distributionType* is DISCRETE_FIELD. The default value is an empty string.
value	A Float specifying the initial value of ion concentration.

Return value

An IonConcentration object.

Exceptions

None.

setValues(...)

This method modifies the IonConcentration object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IonConcentration* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The IonConcentration object has members with the same names and descriptions as the arguments to the *IonConcentration* method.

Corresponding analysis keywords

- **INSTANCE*

KinematicHardening object

The KinematicHardening object stores the data for initial equivalent plastic strains and, if relevant, the initial backstress tensor.

The KinematicHardening object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

KinematicHardening(...)

This method creates a KinematicHardening object.

Path

```
mdb.models[name].KinematicHardening
```

Required arguments

name A String specifying the repository key.

region A *Region* object specifying the region to which the predefined field is applied.

Optional arguments

numBackStress An Int specifying the number of backstresses. The default value is 1.

equivPlasticStrain A sequence of Floats specifying the initial equivalent plastic strain.

backStress A sequence of sequences of Floats specifying the initial backstress tensor for kinematic hardening models. The default value is an empty sequence.

sectPtNum A sequence of Ints specifying section point numbers. This argument is valid only when *definition*=SECTION_PTS.

definition A SymbolicConstant specifying different types of kinematic hardening. Possible values are KINEMATIC_HARDENING, CRUSHABLE_FOAM, REBAR, SECTION_PTS, and USER_DEFINED. The default value is KINEMATIC_HARDENING.

rebarLayerNames A sequence of Strings specifying rebar layer names. This argument is valid only when *definition*=REBAR.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are MAGNITUDE and ANALYTICAL_FIELD. The default value is MAGNITUDE.

Return value

A KinematicHardening object.

Exceptions

None.

setValues(...)

This method modifies the KinematicHardening object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *KinematicHardening* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The KinematicHardening object has members with the same names and descriptions as the arguments to the *KinematicHardening* method.

In addition, the KinematicHardening object can have the following member:

field
A String specifying the name of the *AnalyticalField* object associated with this predefined field. The *field* argument applies only when *distributionType*=ANALYTICAL_FIELD. The default value is an empty string.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=HARDENING

MaterialAssignment object

The MaterialAssignment object stores the data for an initial material assignment predefined field, for use with an Eulerian analysis.

The MaterialAssignment object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

MaterialAssignment(...)

This method creates a MaterialAssignment predefined field object.

Path

```
mdb.models[name].MaterialAssignment
```

Required arguments

name
A String specifying the repository key.

instanceList
A *PartInstanceArray* object specifying the part instances to which the predefined field is applied. All instances must be assigned the same Eulerian section.

Optional arguments

useFields
A Boolean specifying whether the volume fraction data will be uniform or defined by discrete fields. The default value is OFF.

assignmentList
A sequence of tuples specifying the uniform volume fractions to be assigned. This argument is valid only when *useFields*=FALSE. Each tuple contains two entries:

- A *Region* object.
- A tuple of Floats specifying the uniform volume fraction values. The length of the tuple must match the number of material instance names specified in the Eulerain section assigned to part instances specified by *instanceList*.

fieldList
A sequence of tuples specifying the discrete volume fractions to be assigned. This argument is valid only when *useFields*=TRUE. Each tuple contains two entries:

- A *Region* object.
- A tuple of Strings specifying Discrete Field names. The length of the tuple must match the number of material instance names specified in the Eulerain section assigned to part instances specified by *instanceList*.

colorList
A sequence of three Ints specifying colors used to display the material instance assignments. This is a sequence of R,G,B colors, where the values are represented by integers between 0 and 255. The default value is an empty sequence.

Return value

A MaterialAssignment object.

Exceptions

None.

`setValues(...)`

This method modifies the MaterialAssignment object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MaterialAssignment* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The MaterialAssignment object has members with the same names and descriptions as the arguments to the *MaterialAssignment* method.

In addition, the MaterialAssignment object can have the following member:

region
A *Region* object specifying the region to which the predefined field is applied.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=EULERIAN MATERIAL

PredefinedFieldState object

The PredefinedFieldState object is the base object for the objects in the predefinedFieldState repository of the *Step* object. The members of the PredefinedFieldState object are common to all objects derived from PredefinedFieldState.

The PredefinedFieldState object has no constructor or methods.

Access

```
import load
mdb.models[name].steps[name].predefinedFieldStates[name]
```

Members

The PredefinedFieldState object has the following member:

status A SymbolicConstant specifying the propagation state of the PredefinedFieldState object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED
- DEACTIVATED
- DEACTIVATED_TO_INITIAL
- NO_LONGER_ACTIVE
- RESET_TO_INITIAL
- TO_BE_COMPUTED
- PROPAGATED_FROM_COMPUTED
- BUILT_INTO_BASE_STATE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

This member exists in all PredefinedFieldState objects, but different predefined fields use different subsets of the entire list of possible values depending on propagation rules.

SolidElectricPotential object

The SolidElectricPotential object stores the data for the solid electric potential predefined field.

The SolidElectricPotential object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

SolidElectricPotential(...)

This method creates an SolidElectricPotential predefined field object.

Path

```
mdb.models[name].SolidElectricPotential
```

Required arguments

name	A String specifying the repository key.
region	A *Region* object specifying the region to which the predefined field is applied.

Optional arguments

distributionType	A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, FIELD, and DISCRETE_FIELD. The default value is UNIFORM.
field	A String specifying the name of the AnalyticalField or DiscreteField object associated with this predefined field. The field argument applies only when *distributionType* is FIELD or *distributionType* is DISCRETE_FIELD. The default value is an empty string.
value	A Float specifying the initial value of solid electric potential.

Return value

The SolidElectricPotential object.

Exceptions

None.

setValues(...)

This method modifies the SolidElectricPotential object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SolidElectricPotential* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The SolidElectricPotential object has members with the same names and descriptions as the arguments to the *SolidElectricPotential* method.

Corresponding analysis keywords

- **INSTANCE*

Stress object

The Stress object stores the data for an initial stress predefined field.

The Stress object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

Stress(...)

This method creates a Stress predefined field object.

Path

```
mdb.models[name].Stress
```

Required arguments

name	A String specifying the repository key.
region	A *Region* object specifying the region to which the predefined field is applied. Region is ignored if the predefined field has *distributionType*=FROM_FILE.

Optional arguments

distributionType	A SymbolicConstant specifying whether the load is uniform. Possible values are UNIFORM and FROM_FILE. The default value is UNIFORM.
sigma11	A Float specifying the first principal component of the stress.
sigma22	A Float specifying the second principal component of the stress.
sigma33	A Float specifying the third principal component of the stress.
sigma12	A Float specifying the first shear component of the stress.
sigma13	A Float specifying the second shear component of the stress.
sigma23	A Float specifying the third shear component of the stress.

Return value

A Stress object.

Exceptions

None.

setValues(...)

This method modifies the Stress object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Stress* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Stress object has members with the same names and descriptions as the arguments to the *Stress* method.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=STRESS

Temperature object

The Temperature object stores the data for temperature predefined fields.

The Temperature object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

Temperature(...)

This method creates a Temperature object.

Path

```
mdb.models[name].Temperature
```

Required arguments

name

A String specifying the repository key.

createStepName

A String specifying the name of the step in which the predefined field is created.

region

A *Region* object specifying the region to which the predefined field is applied. *Region* is ignored if the predefined field has a *distributionType* member available, and *distributionType*=FROM_FILE .

Optional arguments

distributionType

A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, USER_DEFINED, FROM_FILE, FIELD, FROM_FILE_AND_USER_DEFINED, and DISCRETE_FIELD. The default value is UNIFORM.

crossSectionDistribution

A SymbolicConstant specifying how the predefined field is distributed over the cross section of the region. Possible values are

- CONSTANT_THROUGH_THICKNESS
- GRADIENTS_THROUGH_SHELL_CS
- GRADIENTS_THROUGH_BEAM_CS
- POINTS_THROUGH_SECTION

The default value is CONSTANT_THROUGH_THICKNESS.

field

A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this predefined field. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

amplitude

A String or the SymbolicConstant UNSET specifying the name of the amplitude reference. UNSET should be used if the predefined field has no amplitude reference. The default value is UNSET.



Note:

amplitude should be given only if it is valid for the specified step.

fileName

A String specifying the name of the file from which the temperature values are to be read when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

beginStep

An Int specifying the first step from which temperature values are to be read or the SymbolicConstant FIRST_STEP or LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

beginIncrement

An Int specifying the first increment of the step set in *beginStep* or the SymbolicConstants STEP_START or STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endStep

An Int specifying the last step from which temperature values are to be read or the SymbolicConstants FIRST_STEP and LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endIncrement

An Int specifying the last increment of the step set in *endStep* or the SymbolicConstants STEP_START and STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

interpolate

A SymbolicConstant specifying whether to interpolate a field read from an output database or results file. Possible values are OFF, ON or MIDSIDE_ONLY. The default value is OFF.

magnitudes

A Sequence of Doubles specifying the temperature values when *distributionType*=UNIFORM or FIELD. The value of the *magnitudes* argument is a function of the *crossSectionDistribution* argument, as shown in the following list:

- If *crossSectionDistribution*=CONSTANT_THROUGH_THICKNESS then *magnitudes* is a Double specifying the temperature.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_SHELL_CS then *magnitudes* is a sequence of Doubles specifying the mean value and the gradient in the thickness direction.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_BEAM_CS then *magnitudes* is a sequence of Doubles specifying the mean value, the gradient in the N1 direction, and the gradient in the N2 direction.
- If *crossSectionDistribution*=POINTS_THROUGH_SECTION then *magnitudes* is a sequence of Doubles specifying the temperature at each point.

absoluteExteriorTolerance A Float specifying the absolute value by which a driven node of the field can lie outside the region of the elements of the global model. The default value is 0.0. This argument cannot be used with *midside*.

exteriorTolerance A Float specifying the fraction of the average element size in the global model by which a driven node of the field can lie outside the region of the elements of the global model. The default value is 0.0. This argument cannot be used with *midside*.

Return value

A Temperature object.

Exceptions

None.

`move(...)`

This method moves the *TemperatureState* object from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the *PredefinedFieldState* is moved.

toStepName A String specifying the name of the step to which the *PredefinedFieldState* is moved.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

`setValues(...)`

This method modifies the data for an existing Temperature object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Temperature* method, except for the *name* and *createStepName* arguments.

Return value
`None.`

Exceptions
None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing Temperature object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the predefined field is modified.

Optional arguments
The optional arguments to `setValuesInStep` are the same as the optional arguments to the *Temperature* method, except for the *distributionType* and *crossSectionDistribution* arguments.

Return value
`None.`

Exceptions
None.

Members
The Temperature object can have the following members:

name A String specifying the repository key.

distributionType A SymbolicConstant specifying how the predefined field varies spatially. Possible values are UNIFORM, USER_DEFINED, FROM_FILE, FIELD, FROM_FILE_AND_USER_DEFINED, and DISCRETE_FIELD. The default value is UNIFORM.

field A String specifying the name of the *AnalyticalField* or *DiscreteField* object associated with this predefined field. The *field* argument applies only when *distributionType*=FIELD or *distributionType*=DISCRETE_FIELD. The default value is an empty string.

region A *Region* object specifying the region to which the predefined field is applied. *Region* is ignored if the predefined field has an *instances* member available. *Region* is also ignored if the predefined field has a *distributionType* member available, and *distributionType*=FROM_FILE or FROM_FILE_AND_USER_DEFINED.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=TEMPERATURE
- **TEMPERATURE*

TemperatureState object

The TemperatureState object stores the propagating data of a temperature in a step. One instance of this object is created internally by the *Temperature* object for each step.

The TemperatureState object has no constructor or methods.

The TemperatureState object is derived from the *PredefinedFieldState* object.

Access

```
import load
mdb.models[name].steps[name].predefinedFieldStates[name]
```

Members

The TemperatureState object can have the following members:

fileNameState

A SymbolicConstant specifying the propagation state of the *fileName* member. Possible values are UNSET, SET, and UNCHANGED.

beginStep

A SymbolicConstant or an Int specifying the first step from which temperature values are to be read. This argument is valid only when *distribution*=FROM_FILE or *distribution*=FROM_FILE_AND_USER_DEFINED. Possible values are FIRST_STEP, LAST_STEP, and NONE. The default value is NONE.

beginStepState

A SymbolicConstant specifying the propagation state of the *beginStep* member. Possible values are UNSET, SET, and UNCHANGED.

beginIncrement

`None` or an Int specifying the first increment of the step set in *beginStep* or the SymbolicConstants STEP_START or STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

beginIncrementState

A SymbolicConstant specifying the propagation state of the *beginIncrement* member. Possible values are UNSET, SET, and UNCHANGED.

endStep

`None` or an Int specifying the last step from which temperature values are to be read or the SymbolicConstants FIRST_STEP and LAST_STEP. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endStepState

A SymbolicConstant specifying the propagation state of the *endStep* member. Possible values are UNSET, SET, and UNCHANGED.

endIncrement

`None` or an Int specifying the last increment of the step set in *endStep* or the SymbolicConstants STEP_START and STEP_END. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED. The default value is `None`.

endIncrementState A SymbolicConstant specifying the propagation state of the *endIncrement* member. Possible values are UNSET, SET, and UNCHANGED.

midside A Boolean specifying that temperatures in second-order elements are to be interpolated from corner node temperatures. This argument is valid only when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

midsideState A SymbolicConstant specifying the propagation state of the *midside* member. Possible values are UNSET, SET, and UNCHANGED.

amplitudeState A SymbolicConstant specifying the propagation state of the *amplitudeState* member. Possible values are UNSET, SET, and UNCHANGED.

fileName A String specifying the name of the file from which the temperature values are to be read when *distributionType*=FROM_FILE or *distributionType*=FROM_FILE_AND_USER_DEFINED.

amplitude The SymbolicConstant UNSET or a String specifying the name of the amplitude reference. The SymbolicConstant UNSET should be used if the predefined field has no amplitude reference. The default value is UNSET.



Note:

amplitude should be given only if it is valid for the specified step.

magnitudesState A tuple of SymbolicConstants specifying the propagation state of each item of the *magnitudes* member. Possible values are UNSET, SET, and UNCHANGED.

magnitudes A tuple of Floats specifying the temperature values when *distributionType*=UNIFORM or *distributionType*=FIELD. The value of the *magnitudes* argument is a function of the *crossSectionDistribution* argument, as shown in the following list:

- If *crossSectionDistribution*=CONSTANT_THROUGH_THICKNESS then *magnitudes* is a Double specifying the temperature.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_SHELL_CS then *magnitudes* is a sequence of Doubles specifying the mean value and the gradient in the thickness direction.
- If *crossSectionDistribution*=GRADIENTS_THROUGH_BEAM_CS then *magnitudes* is a sequence of Doubles specifying the mean value, the gradient in the N1 direction, and the gradient in the N2 direction.
- If *crossSectionDistribution*=POINTS_THROUGH_SECTION then *magnitudes* is a sequence of Doubles specifying the temperature at each point.

status A SymbolicConstant specifying the propagation state of the *PredefinedFieldState* object. Possible values are:

- NOT_YET_ACTIVE
- CREATED
- PROPAGATED
- MODIFIED

- DEACTIVATED
- DEACTIVATED_TO_INITIAL
- NO_LONGER_ACTIVE
- RESET_TO_INITIAL
- TO_BE_COMPUTED
- PROPAGATED_FROM_COMPUTED
- BUILT_INTO_BASE_STATE
- TYPE_NOT_APPLICABLE
- INSTANCE_NOT_APPLICABLE

This member exists in all *PredefinedFieldState* objects, but different predefined fields use different subsets of the entire list of possible values depending on propagation rules.

Velocity object

The Velocity object stores the data for an initial velocity predefined field.

The Velocity object is derived from the *PredefinedField* object.

Access

```
import load
mdb.models[name].predefinedFields[name]
```

Velocity(...)

This method creates a Velocity predefined field object.

Path

```
mdb.models[name].Velocity
```

Required arguments

name A String specifying the repository key.

region A *Region* object specifying the region to which the predefined field is applied.

velocity1 A Float specifying the first component of the velocity.

velocity2 A Float specifying the second component of the velocity.

velocity3 A Float specifying the third component of the velocity.

omega A Float specifying the angular velocity.

axisBegin A sequence of Floats specifying the *X*-, *Y*-, and *Z*- coordinates of the starting point of the axis about which *omega* is defined.

axisEnd A sequence of Floats specifying the *X*-, *Y*-, and *Z*- coordinates of the end point of the axis about which *omega* is defined.

Optional arguments

field A String specifying the name of the *AnalyticalField* object associated with this predefined field. The *field* argument applies only when *distributionType*=FIELD_ANALYTICAL. The default value is an empty string.

distributionType A SymbolicConstant specifying whether the load is uniform. Possible values are MAGNITUDE and FIELD_ANALYTICAL. The default value is MAGNITUDE.

Return value

A Velocity object.

Exceptions

None.

`setValues(...)`

This method modifies the Velocity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Velocity* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The Velocity object has members with the same names and descriptions as the arguments to the *Velocity* method.

Corresponding analysis keywords

- **INITIAL CONDITIONS*, TYPE=VELOCITY
- **INITIAL CONDITIONS*, TYPE=ROTATING VELOCITY

Print commands

The print commands are used to print selected canvas objects to a file or to a Windows or PostScript printer and to control the format of the printed output.

In this section:

- *PrintOptions object*
- *EpsOptions object*
- *PageSetupOptions object*
- *PngOptions object*
- *PsOptions object*
- *SvgOptions object*
- *TiffOptions object*

PrintOptions object

The PrintOptions object stores the common settings that Abaqus uses for all print methods. The PrintOptions object has no constructor. Abaqus creates the *printOptions* member when a session is started.

Access

```
session.printOptions
```

`setValues(...)`

This method modifies the PrintOptions object.

Required arguments

None.

Optional arguments

rendition
A SymbolicConstant specifying how the image is rendered. Possible values are BLACK_AND_WHITE, GREYSCALE, and COLOR. The default value is COLOR.

vpDecorations
A Boolean specifying whether the output includes the viewport border and title. The default value is ON.

vpBackground
A Boolean specifying whether the output includes viewport backgrounds. The default value is OFF.

compass
A Boolean specifying whether the output includes the view compass The default value is OFF.

printCommand
A String specifying the default print command that will be used by the `printToPrinter` method if no *printCommand* argument is provided. The default value is "lpr".

deleteTemporaryFiles
A Boolean specifying whether to delete the temporary files created when an image is printed. The default value is ON.

You should set the *deleteTemporaryFiles* argument to False if your printer does not support print spooling.

reduceColors
A Boolean specifying whether the raster image printed is reduced from true color to 256 colors to reduce file size. The quality of the image will be compromised. The default value is OFF.

Return value

`None.`

Exceptions

None.

Members

The PrintOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

EpsOptions object

The EpsOptions object stores the settings that Abaqus uses when printing using Encapsulated PostScript format. The EpsOptions object has no constructor. Abaqus creates the *epsOptions* member when a session is started.

Access

```
session.epsOptions
```

`setValues(...)`

This method modifies the EpsOptions object.

Required arguments

None.

Optional arguments

imageSize

The SymbolicConstant SIZE_ON_SCREEN or a pair of Floats specifying the width and height of the image in the units specified by *units*. Possible numeric values are *imageSize* $\geq$ (*minWidth*, *minHeight*). The default value is SIZE_ON_SCREEN.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is 10 mm (approximately 0.4 inches).

units

A SymbolicConstant specifying the units of the *imageSize* argument. This argument is ignored if *imageSize* is SIZE_ON_SCREEN. Possible values are INCHES and MM. The default value is INCHES.

resolution

A SymbolicConstant specifying the resolution of the image in dots per inch (dpi). The *resolution* can be DPI_1200 only if *imageFormat*=VECTOR. Possible values are DPI_75, DPI_150, DPI_300, DPI_450, DPI_600, and DPI_1200. The default value is DPI_150.

fontType

A SymbolicConstant specifying the PostScript font substitution rules to be applied. Possible values are PS_ALWAYS, PS_IF_AVAILABLE, and AS_DISPLAYED. The default value is PS_IF_AVAILABLE.

imageFormat

A SymbolicConstant specifying how the viewport display will be represented. Possible values are VECTOR and RASTER. The default value is VECTOR.

shadingQuality

A SymbolicConstant specifying how fine the shading of curved surfaces will be for vector images. Possible values are EXTRA COARSE, COARSE, MEDIUM, FINE, and EXTRA FINE. The default value is MEDIUM.

Return value

```
None.
```

Exceptions

RangeError.

If either element of *imageSize* is out of range:	RangeError: imageSize must be SIZE_ON_SCREEN or a sequence of 2 Floats >= (*minWidth*, *minHeight*)

Members

The EpsOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

PageSetupOptions object

The PageSetupOptions object stores the settings that Abaqus uses when printing using a Windows printer. The PageSetupOptions object has no constructor. Abaqus creates the *pageSetupOptions* member when a session is started.

Access

```
session.pageSetupOptions
```

`setValues(...)`

This method modifies the PageSetupOptions object.

Required arguments

None.

Optional arguments

imageSize

A SymbolicConstant or a pair of Floats specifying the size of the printed image in the currently selected units (inches or millimeters). Possible values are FIT_TO_PAGE and SIZE_ON_SCREEN. The default value is FIT_TO_PAGE.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is 10 mm (approximately 0.4 inches).

units

A SymbolicConstant specifying the units to use for the margins and image size. Possible values are INCHES and MM. The default value is INCHES.

quality

A SymbolicConstant specifying the quality of the image. Possible values are COARSE, MEDIUM, and FINE. The default value is MEDIUM.

topMargin

A Float specifying the top margin of the paper in the currently selected units (inches or millimeters). Possible values are *topMargin* ≥ 0. The default value is 0.5.

bottomMargin

A Float specifying the bottom margin of the paper in the currently selected units (inches or millimeters). Possible values are *bottomMargin* ≥ 0. The default value is 0.5.

leftMargin

A Float specifying the left margin of the paper in the currently selected units (inches or millimeters). Possible values are *leftMargin* ≥ 0. The default value is 0.5.

rightMargin

A Float specifying the right margin of the paper in the currently selected units (inches or millimeters). Possible values are *rightMargin* ≥ 0. The default value is 0.5.

orientation

A SymbolicConstant specifying the orientation of the image. Possible values are PORTRAIT and LANDSCAPE. The default value is PORTRAIT.

logo A Boolean specifying whether the output includes the Abaqus logo. The default value is ON.

date A Boolean specifying whether the output includes the date. The default value is ON.

Return value

`None.`

Exceptions

RangeError.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is 10 mm (approximately 0.4 inches).

If *leftMargin* + *rightMargin* is out of range: RangeError: leftMargin and rightMargin must produce image width >= minWidth

If *topMargin* + *bottomMargin* is out of range: RangeError: topMargin and bottomMargin must produce image height >= minHeight

Members

The PageSetupOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

PngOptions object

The PngOptions object stores the settings that Abaqus uses when printing in PNG format. The PngOptions object has no constructor. Abaqus creates the *pngOptions* member when a session is started.

Access

```
session.pngOptions
```

`setValues(...)`

This method modifies the PngOptions object.

Required arguments

None.

Optional arguments

imageSize The SymbolicConstant SIZE_ON_SCREEN or a pair of Ints specifying the width and height of the image in pixels. Possible values are (*minWidth*, *minHeight*) $\leq$ *imageSize* $\leq$ (*maxWidth* and *maxHeight*). The default value is SIZE_ON_SCREEN.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is the number of pixels that occupy 10 mm at the current screen resolution. The value is typically around 50 pixels and may be different for width and height.

The maximum value of width and height (*maxWidth* and *maxHeight*) is the largest number of pixels supported by the system graphics and will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

RangeError.

If either the width or height arguments of *imageSize* are out of range (where *minWidth* and *minHeight* are the number of pixels corresponding to approximately 10 mm for a given display and *maxWidth* and *maxHeight* are the largest allowable number of pixels supported by the system graphics):	RangeError: imageSize must be SIZE_ON_SCREEN or a sequence of 2 Ints in the range (minWidth, minHeight) <= (width, height) <= (maxWidth, maxHeight).

Members

The PngOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

PsOptions object

The PsOptions object stores the settings that Abaqus uses when printing using PostScript format. The PsOptions object has no constructor. Abaqus creates the *psOptions* member when a session is started.

Access

```
session.psOptions
```

`setValues(...)`

This method modifies the PsOptions object.

Required arguments

None.

Optional arguments

paperSize

A SymbolicConstant specifying the paper size. Possible values are:

- LETTER
- LEGAL
- LEDGER
- A0
- A1
- A2
- A3
- A4
- A5

The default value is LETTER.

topMargin

A Float specifying the top margin of the paper in inches. Possible values are *topMargin* $\geq$ 0. The default value is 0.5.

bottomMargin

A Float specifying the bottom margin of the paper in inches. Possible values are *bottomMargin* $\geq$ 0. The default value is 0.5.

leftMargin

A Float specifying the left margin of the paper in inches. Possible values are *leftMargin* $\geq$ 0. The default value is 0.5.

rightMargin

A Float specifying the right margin of the paper in inches. Possible values are *rightMargin* $\geq$ 0. The default value is 0.5.

orientation

A SymbolicConstant specifying the orientation of the image. Possible values are PORTRAIT and LANDSCAPE. The default value is PORTRAIT.

logo
A Boolean specifying whether the output includes the Abaqus logo. The default value is ON.

date
A Boolean specifying whether the output includes the date. The default value is ON.

resolution
A SymbolicConstant specifying the resolution of the image in dots per inch (dpi). The *resolution* can be DPI_1200 only if *imageFormat* = VECTOR. Possible values are DPI_75, DPI_150, DPI_300, DPI_450, DPI_600, and DPI_1200. The default value is DPI_150.

fontType
A SymbolicConstant specifying the PostScript font substitution rules to be applied. Possible values are PS_ALWAYS, PS_IF_AVAILABLE, and AS_DISPLAYED. The default value is PS_IF_AVAILABLE.

imageFormat
A SymbolicConstant specifying how the viewport display will be represented. Possible values are VECTOR and RASTER. The default value is VECTOR.

shadingQuality
A SymbolicConstant specifying how fine the shading of curved surfaces will be for vector images. Possible values are EXTRA COARSE, COARSE, MEDIUM, FINE, and EXTRA FINE. The default value is MEDIUM.

Return value
`None.`

Exceptions
RangeError.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is 10 mm (approximately 0.4 inches).

If *leftMargin* + *rightMargin* is out of range: RangeError: leftMargin and rightMargin must produce image width >= minWidth

If *topMargin* + *bottomMargin* is out of range: RangeError: topMargin and bottomMargin must produce image height >= minHeight

Members
The PsOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

SvgOptions object

The SvgOptions object stores the settings that Abaqus uses when printing in SVG format. The SvgOptions object has no constructor. Abaqus creates the *svgOptions* member when a session is started.

Access

```
session.svgOptions
```

setValues(...)

This method modifies the SvgOptions object.

Required arguments

None.

Optional arguments

imageSize The SymbolicConstant SIZE_ON_SCREEN or a pair of Ints specifying the width and height of the image in pixels. Possible values are (*minWidth*, *minHeight*) $\leq$ *imageSize* $\leq$ (*maxWidth* and *maxHeight*). The default value is SIZE_ON_SCREEN.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is the number of pixels that occupy 10 mm at the current screen resolution. The value is typically around 50 pixels and may be different for width and height.

The maximum value of width and height (*maxWidth* and *maxHeight*) is the largest number of pixels supported by the system graphics and will be at least as large as the screen dimensions.

Return value

None.

Exceptions

RangeError.

If either the width or height arguments of *imageSize* are out of range (where *minWidth* and *minHeight* are the number of pixels corresponding to approximately 10 mm for a given display and *maxWidth* and *maxHeight* are the largest allowable number of pixels supported by the system graphics):	RangeError: imageSize must be SIZE_ON_SCREEN or a sequence of 2 Ints in the range (minWidth, minHeight) <= (width, height) <= (maxWidth, maxHeight).

Members

The SvgOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

TiffOptions object

The TiffOptions object stores the settings that Abaqus uses when printing in TIFF format. The TiffOptions object has no constructor. Abaqus creates the *tiffOptions* member when a session is started.

Access

```
session.tiffOptions
```

`setValues(...)`

This method modifies the TiffOptions object.

Required arguments

None.

Optional arguments

imageSize The SymbolicConstant SIZE_ON_SCREEN or a pair of Ints specifying the width and height of the image in pixels. Possible values are (*minWidth*, *minHeight*) $\leq$ *imageSize* $\leq$ (*maxWidth* and *maxHeight*). The default value is SIZE_ON_SCREEN.



Note:

The minimum value of width and height (*minWidth* and *minHeight*) is the number of pixels that occupy 10 mm at the current screen resolution. The value is typically around 50 pixels and may be different for width and height.

The maximum value of width and height (*maxWidth* and *maxHeight*) is the largest number of pixels supported by the system graphics and will be at least as large as the screen dimensions.

Return value

`None.`

Exceptions

RangeError.

If either the width or height arguments of *imageSize* are out of range (where *minWidth* and *minHeight* are the number of pixels corresponding to approximately 10 mm for a given display and *maxWidth* and *maxHeight* are the largest allowable number of pixels supported by the system graphics):	RangeError: imageSize must be SIZE_ON_SCREEN or a sequence of 2 Ints in the range (minWidth, minHeight) <= (width, height) <= (maxWidth, maxHeight).

Members

The TiffOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

Property commands

The Property commands are used to create and manage reinforcements and to assign properties to a part.

(See also *Material commands* and *Section commands*.) The Property commands are methods of a Part object.

In this section:

- *SectionAssignment object*
- *CompositeLayup object*
- *CompositePly object*
- *MaterialOrientation object*
- *PlyStackPlot object*
- *Property assignment commands*

SectionAssignment object

The SectionAssignment object is used to specify a section assignment on an assembly or part. Section assignments on the assembly are limited to connector elements only.

Access

```
import section
mdb.models[name].parts[name].sectionAssignments[i]
import assembly
mdb.models[name].rootAssembly.sectionAssignments[i]
import odbAccess
session.odbs[name].parts[name].sectionAssignments[i]
session.odbs[name].rootAssembly.instances[name].sectionAssignments[i]
session.odbs[name].rootAssembly.sectionAssignments[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.sectionAssignments[i]
```

SectionAssignment(...)

This method creates a SectionAssignment object.

Path

mdb.models[*name*].parts[*name*].SectionAssignment

mdb.models[*name*].rootAssembly.SectionAssignment

Required arguments

region	A *Set* object specifying the region to which the section is assigned.
sectionName	A String specifying the name of the section.

Optional arguments

thicknessAssignment	A SymbolicConstant specifying section thickness assignment method. Possible values are FROM_SECTION and FROM_GEOMETRY. The default value is FROM_SECTION.
offset	A Float specifying the offset of the shell section. The default value is 0.0.
offsetType	A SymbolicConstant specifying the method used to define the shell offset. If *offsetType* is set to OFFSET_FIELD the *offsetField* must have a value. Possible values are SINGLE_VALUE, MIDDLE_SURFACE, TOP_SURFACE, BOTTOM_SURFACE, FROM_GEOMETRY, and OFFSET_FIELD. The default value is SINGLE_VALUE.
offsetField	A String specifying the name of the field specifying the offset. The default value is "".

Return value

A SectionAssignment object.

Exceptions

None.

`resume()`

This method resumes the section assignment that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

`suppress()`

This method suppresses the section assignment.

Arguments

None.

Return value

`None.`

Exceptions

None.

`getVertices(...)`

This method is only valid for connector section assignments. This method returns a sequence consisting of tuples of coordinates of the connector's endpoints.

Arguments

None.

Return value

A sequence of tuples of floats.

Exceptions

An exception is thrown if getVertices() is called on any section assignment except connector section assignment.	This method is valid only for connector section assignments.

setValues(...)

This method modifies the SectionAssignment object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SectionAssignment* method.

Return value

`None.`

Exceptions

None.

Members

The SectionAssignment object has members with the same names and descriptions as the arguments to the *SectionAssignment* method.

In addition, the SectionAssignment object has the following member:

suppressed
A Boolean specifying whether the section assignment is suppressed or not. The default value is OFF.

CompositeLayup object

The CompositeLayup object is used to specify a composite layup on a part.

Access

```
import part
mdb.models[name].parts[name].compositeLayups[i]
```

CompositeLayup(...)

This method creates a CompositeLayup object.

Path

mdb.models[*name*].parts[*name*].CompositeLayup

Required arguments

name A String specifying the repository key.

Optional arguments

description A String specifying a description of the composite layup.

offsetType A SymbolicConstant specifying the method used to define the shell offset. If *offsetType*=OFFSET_FIELD the *offsetField* argument is required. This member is valid only if *elementType*=SHELL. Possible values are SINGLE_VALUE, MIDDLE_SURFACE, TOP_SURFACE, BOTTOM_SURFACE, OFFSET_FIELD, and GLOBAL. The default value is GLOBAL.

offsetField A String specifying The name of the field specifying the offset. This member is valid only if *elementType*=SHELL. The default value is an empty string.

offsetValues A Float specifying The offset of the shell section. This member is valid only if *elementType*=SHELL. The default value is 0.0.

elementType A SymbolicConstant specifying the type of element in the composite layup. Possible values are SHELL, CONTINUUM_SHELL, and SOLID. The default value is SHELL.

symmetric A Boolean specifying whether or not the layup should be made symmetric by the analysis. The default value is OFF.

Return value

A CompositeLayup object.

Exceptions

AbaqusException.

suppress()

This method suppresses a composite layup.

Arguments

None.

Return value

`None.`

Exceptions

None.

resume()

This method resumes a composite layup that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

None.

deletePlies()

This method deletes all of the plies from a composite layup.

Arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the CompositeLayup object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CompositeLayup* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The CompositeLayup object has members with the same names and descriptions as the arguments to the *CompositeLayup* method.

In addition, the CompositeLayup object has the following members:

section A *GeometryShellSection* object.

orientation A *MaterialOrientation* object.

plies A *CompositePlyArray* object specifying the plies that make up this composite layup.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*
- **SOLID SECTION*

CompositePly object

The CompositePly object defines the material layers in a composite layup.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].plies[i]
```

CompositePly(...)

This method creates a CompositePly object.

Path

mdb.models[*name*].parts[*name*].compositeLayups[*name*].CompositePly

Required arguments

thickness
A Float specifying the thickness of the section layer.

region
A *Region* object specifying the region to which the composite ply applies.

material
A String specifying the name of the material for the ply.

plyName
A String specifying the ply identifier for this section layer. The default value is an empty string.

orientationType
A SymbolicConstant specifying the method used to define the relative orientation. If *orientationType*=SPECIFY_ORIENT the *orientationValue* argument is required. If *orientationType*=CSYS the *orientation* argument is required. Possible values are CSYS, SPECIFY_ORIENT, ANGLE_0, ANGLE_45, ANGLE_90, and ANGLE_NEG45. The default value is ANGLE_0.

thicknessType
A SymbolicConstant specifying the method used to define the thickness. If *thicknessType*=SPECIFY_THICKNESS, the *thickness* argument is required. Possible values are SPECIFY_THICKNESS, FIELD_THICKNESS, and ANALYTICAL_FIELD_THICKNESS. The default value is SPECIFY_THICKNESS.

Optional arguments

orientationValue
A Float specifying the relative orientation of the section layer. The default value is 0.0.

thicknessField
A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements and composite ply. The *thicknessField* argument applies when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD for shell elements and *thicknessType*=FIELD_THICKNESS or *thicknessType*=ANALYTICAL_FIELD_THICKNESS for composite ply. The default value is an empty string.

numIntPts
An Int specifying the number of integration points to be used through the section layer. This argument is valid only if *preIntegrate*=OFF. The default value is 3.

axis
A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. The *axis* argument applies only if a valid reference is provided for the *orientation*. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle
A Float specifying the angle of the additional rotation. The *angle* argument applies only if a valid reference is provided for the *orientation*. The default value is 0.0.

additionalRotationType
A SymbolicConstant specifying the method used to describe the additional rotation when a valid orientation is specified. Use *orientationType*=ANGLE_0 and *additionalRotationType*=ROTATION_FIELD to specify a discrete field of rotations for this CompositePly. Possible values are ROTATION_NONE, ROTATION_ANGLE, and ROTATION_FIELD. The default value is ROTATION_NONE.

orientation
The SymbolicConstant None or a *DatumCsys* object specifying a coordinate system reference for the relative orientation of this layer. The default value is None.

additionalRotationField
A String specifying the name of the field specifying the additional rotation. The default value is an empty string.

Return value

A CompositePly object.

Exceptions

AbaqusException.

Members

The CompositePly object has members with the same names and descriptions as the arguments to the *CompositePly* method.

MaterialOrientation object

The MaterialOrientation object represents the orientation of the material properties and composite layups.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].orientation
mdb.models[name].parts[name].materialOrientations[i]
import odbAccess
session.odbs[name].parts[name].materialOrientations[i]
session.odbs[name].rootAssembly.instances[name]\
.materialOrientations[i]
session.odbs[name].steps[name].frames[i].fieldOutputs[name].values[i]\
.instance.materialOrientations[i]
```

MaterialOrientation(...)

This method creates a MaterialOrientation object.

Path

mdb.models[*name*].parts[*name*].MaterialOrientation

Required arguments

None.

Optional arguments

region

A *Set* object specifying a region for which the material orientation is defined.

localCsys

A *DatumCsys* object specifying the local coordinate system or `None`, describing the material orientation for the given region. In the ODB, this member was previously accessible using "csys," but support has now been added for localCsys and the csys member will be deprecated.

axis

A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle

A Float specifying the angle of the additional rotation (if accessed from the ODB instead of the MDB, it will be a string instead of a float). The default value is 0.0.

stackDirection

A SymbolicConstant specifying the stack or thickness direction. Possible values are STACK_1, STACK_2, STACK_3, and STACK_ORIENTATION. The default value is STACK_3.

fieldName

A String specifying the name of the *DiscreteField* object specifying the orientation. The default value is an empty string.

orientationType
A SymbolicConstant specifying the method used to define the material orientation. If *orientationType*=SYSTEM, the *region* and *localCsys* arguments are required. If *orientationType*=FIELD, the *fieldName* argument is required. Possible values are GLOBAL, SYSTEM, FIELD, DISCRETE, and USER. The default value is GLOBAL.

normalAxisDirection
A SymbolicConstant specifying the axis that is defined by the normal axis direction for a discrete orientation. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_3.

normalAxisDefinition
A SymbolicConstant specifying the method used to define the normal axis direction for a discrete orientation. Possible values are SURFACE, NORMAL_DATUM, and NORMAL_VECTOR. The default value is NORMAL_VECTOR.

normalAxisRegion
A *Surface* object specifying a region whose geometric normals define the normal axis for the discrete orientation.

normalAxisDatum
A *DatumAxis* object specifying the Datum Axis or `None`, describing the normal axis direction for the discrete orientation.

flipNormalDirection
A Boolean specifying the flag to reverse the direction of the defined normal axis direction. The default value is OFF.

normalAxisVector
A sequence of Floats specifying the vector that defines the direction of the normal axis of the discrete orientation.

primaryAxisDirection
A SymbolicConstant specifying the axis that is defined by the primary axis direction for a discrete orientation. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

primaryAxisDefinition
A SymbolicConstant specifying the method used to define the primary axis direction for a discrete orientation. Possible values are SURFACE, PRIMARY_DATUM, and PRIMARY_VECTOR. The default value is PRIMARY_VECTOR.

primaryAxisRegion
A *Set* object specifying a region whose geometric tangents define the primary axis for the discrete orientation.

primaryAxisDatum
A *DatumAxis* object specifying the Datum Axis or `None`, describing the primary axis direction for the discrete orientation.

flipPrimaryDirection
A Boolean specifying the flag to reverse the direction of the defined primary axis direction. The default value is OFF.

primaryAxisVector
A sequence of Floats specifying the vector that defines the direction of the primary axis of the discrete orientation.

Return value

A MaterialOrientation object.

Exceptions

None.

ReferenceOrientation(...)

This method creates a MaterialOrientation object.

Path

mdb.models[*name*].parts[*name*].compositeLayups[*name*].ReferenceOrientation

Required arguments

None.

Optional arguments

localCsys

A *DatumCsys* object specifying the local coordinate system or `None`, describing the material orientation for the given region. In the ODB, this member was previously accessible using "csys," but support has now been added for localCsys and the csys member will be deprecated.

axis

A SymbolicConstant specifying the axis of a datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle

A Float specifying the angle of the additional rotation (if accessed from the ODB instead of the MDB, it will be a string instead of a float). The default value is 0.0.

stackDirection

A SymbolicConstant specifying the stack or thickness direction. Possible values are STACK_1, STACK_2, STACK_3, and STACK_ORIENTATION. The default value is STACK_3.

fieldName

A String specifying the name of the *DiscreteField* object specifying the orientation. The default value is an empty string.

orientationType

A SymbolicConstant specifying the method used to define the material orientation. If *orientationType*=SYSTEM, the *region* and *localCsys* arguments are required. If *orientationType*=FIELD, the *fieldName* argument is required. Possible values are GLOBAL, SYSTEM, FIELD, DISCRETE, and USER. The default value is GLOBAL.

additionalRotationField

A String specifying the name of the *DiscreteField* object specifying the additional rotation. The default value is an empty string.

additionalRotationType

A SymbolicConstant specifying the method used to describe the additional rotation when a valid orientation is specified. Possible values are ROTATION_NONE, ROTATION_ANGLE, and ROTATION_FIELD. The default value is ROTATION_NONE.

normalAxisDirection A SymbolicConstant specifying the axis that is defined by the normal axis direction for a discrete orientation. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_3.

normalAxisDefinition A SymbolicConstant specifying the method used to define the normal axis direction for a discrete orientation. Possible values are SURFACE, DATUM, and VECTOR. The default value is VECTOR.

normalAxisRegion A *Surface* object specifying a region whose geometric normals define the normal axis for the discrete orientation.

normalAxisDatum A *DatumAxis* object specifying the Datum Axis or `None`, describing the normal axis direction for the discrete orientation.

flipNormalDirection A Boolean specifying the flag to reverse the direction of the defined normal axis direction. The default value is OFF.

normalAxisVector A sequence of Floats specifying the vector that defines the direction of the normal axis of the discrete orientation.

primaryAxisDirection A SymbolicConstant specifying the axis that is defined by the primary axis direction for a discrete orientation. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

primaryAxisDefinition A SymbolicConstant specifying the method used to define the primary axis direction for a discrete orientation. Possible values are EDGE, DATUM, and VECTOR. The default value is VECTOR.

primaryAxisRegion A *Set* object specifying a region whose geometric tangents define the primary axis for the discrete orientation.

primaryAxisDatum A *DatumAxis* object specifying the Datum Axis or `None`, describing the primary axis direction for the discrete orientation.

flipPrimaryDirection A Boolean specifying the flag to reverse the direction of the defined primary axis direction. The default value is OFF.

primaryAxisVector A sequence of Floats specifying the vector that defines the direction of the primary axis of the discrete orientation.

Return value

A MaterialOrientation object.

Exceptions

None.

`setValues(...)`

This method modifies the MaterialOrientation object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MaterialOrientation* method.

Return value

`None.`

Exceptions

None.

Members

The MaterialOrientation object has members with the same names and descriptions as the arguments to the *MaterialOrientation* method.

In addition, the MaterialOrientation object can have the following members:

additionalRotationType
A SymbolicConstant specifying the method used to describe the additional rotation when a valid orientation is specified. Possible values are ROTATION_NONE, ROTATION_ANGLE, and ROTATION_FIELD. The default value is ROTATION_NONE.

additionalRotationField
A String specifying the name of the *DiscreteField* object specifying the additional rotation. The default value is an empty string.

PlyStackPlot object

The PlyStackPlot object is used to plot the stacking of plies in a composite layup or in a composite shell section.

Access

```
import section
import visualization
```

MdbPlyStackPlot(...)

This method creates a PlyStackPlot object from a region of a part that contains a composite shell layup.

Path

section.MdbPlyStackPlot

Required arguments

part	A *Part* object.
region	A *Region* object which contains a composite shell layup.

Optional arguments

None.

Return value

A PlyStackPlot object.

Exceptions

None.

OdbPlyStackPlot(...)

This method creates a PlyStackPlot object from a composite shell section of an Odb object.

Path

visualization.OdbPlyStackPlot

Required arguments

odb	An *Odb* object.
sectionName	A String specifying the section name that contains a composite shell section.

Optional arguments

offset A Float specifying the shell offset. The default value is 0.0.

Return value

A PlyStackPlot object.

Exceptions

None.

Members

The PlyStackPlot object has no members.

Property assignment commands

The Property assignment commands are used to assign and unassign properties to parts. The `part` and `section` modules must be imported to give access to the Property assignment commands.

Access

```
import part
import section
```

assignBeamSectionOrientation

This method assigns a beam section orientation to a region of a part.

Path

mdb.models[*name*].parts[*name*].assignBeamSectionOrientation

Required arguments

region A sequence of geomSequences of Edge objects or a sequence of sequences of one-dimensional elements.

method A SymbolicConstant specifying the assignment method. Only a value of N1_COSINES is currently supported.

n1 A sequence of three Floats specifying the approximate local $\mathbf{n_1}$-direction of the beam cross-section.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignMaterialOrientation

This method assigns a material orientation to a region.

Path

mdb.models[*name*].parts[*name*].assignMaterialOrientation

Required arguments

region A sequence of geomSequences of Vertex, Edge, Face, and Cell objects or a sequence of sequences of elements.

localCsys A Datum object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

`None.`

Exceptions

None.

`assignRebarOrientation`

This method assigns a rebar reference orientation to a region.

Path

mdb.models[*name*].parts[*name*].assignRebarOrientation

Required arguments

region A sequence of geomSequences of Vertex, Edge, Face, and Cell objects or a sequence of sequences of elements.

localCsys A Datum object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

`None.`

Exceptions

None.

flipNormal

This method flips the normals of shell or membrane elements of an orphan mesh or of two-dimensional geometric regions.

Path

mdb.models[*name*].parts[*name*].flipNormal

Required arguments

regions A Region object specifying the region on which normals are flipped. For 3D parts, the region contains Face objects or two-dimensional triangle or quadrilateral Element objects. For axisymmetric parts, the region contains Edge objects or line Elements objects.

Optional arguments

referenceRegion A two-dimensional element object whose normal is to be matched. If unspecified, all the normals associated with the given regions will be flipped. The *referenceRegion* argument is applicable only if the argument regions contain a sequence of quadrilateral or triangular elements.

Return value

`None.`

Exceptions

None.

flipTangent

This method flips the tangents of beam or truss elements of an orphan mesh or of one-dimensional geometric regions.

Path

mdb.models[*name*].parts[*name*].flipTangent

Required arguments

regions A Region object specifying the region on which normals are flipped. The region contains Edge objects or one-dimensional Element objects.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

unassignBeamSectionOrientation

This method deletes a beam section orientation assignment.

Path

mdb.models[*name*].parts[*name*].unassignBeamSectionOrientation

Required arguments

index An Int specifying the number of the beam section orientation assignment to be deleted.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

unassignMaterialOrientation

This method deletes a material orientation assignment.

Path

mdb.models[*name*].parts[*name*].unassignMaterialOrientation

Required arguments

index An Int specifying the number of the material assignment to be deleted.

Optional arguments

None.

Return value
`None.`

Exceptions

None.

unassignRebarOrientation

This method deletes a rebar orientation assignment.

Path

mdb.models[*name*].parts[*name*].unassignRebarOrientation

Required arguments

index An Int specifying the number of the rebar reference orientation assignment to be deleted.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Region commands

Region commands are used to create part and assembly sets or surfaces from elements, nodes, and geometry. For more information, see *Specifying a region*. Part and assembly objects have the following member, a repository of Set objects:

- *sets*

In turn, a Set object can contain any one of the following types:

- elements
- nodes
- geometry

A Set object can contain a number of entities of a single type (nodes, elements, or geometry) or a combination of node and element types. However, except for nodes and elements, a Set object cannot contain a combination of types.

The following are members of the Set object:

- nodes
- elements
- cells
- faces
- edges
- vertices
- referencePoints

Region commands are also used to create surfaces on the assembly. Surfaces are sets with additional "sidedness" information.

Part sets contain regions of a part. You can assign section definitions to a set created by selecting a region of a part. The part sets can be accessed from the instance; however, the section definition you assigned to a region is copied automatically to all instances of that part in the assembly.

Assembly sets contain regions of an assembly and are used by many commands that operate on the assembly. For example, you can apply a load or boundary condition to a set created by selecting a region of the assembly. Sets can include regions from multiple part instances.

In this section:

- *Assembly object*
- *Part object*
- *Region object*
- *Set object*
- *Surface object*
- *Skin object*
- *Stringer object*

Assembly object

The following commands operate on Assembly objects. For more information about the Assembly object, see *Assembly object*.

Access

```
import regionToolset
```

clashSets(...)

This command prints a message describing the relationship between the contents of two sets. Possible outcomes are:

- Both sets are the same.
- Set 2 is a subset of set 1.
- Set 2 is a superset of set 1.
- Set 2 intersects set 1.
- Set 2 touches set 1 (their boundaries intersect).
- Set 2 and set 1 are disjoint.

This command accepts only positional arguments and has no keywords.

Required arguments

arg1	A Set or Surface object specifying set 1.
arg2	A Set or Surface object specifying set 2.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteSets(...)

This command deletes the given sets from the assembly.

Required arguments

setNames	A sequence of Strings specifying the set names that will be deleted from the assembly.

Optional arguments

None.

Return value

None.

Exceptions

None.

markSetInternal(...)

This command marks the given Set as internal or external.

Required arguments

setName	A string specifying the Set name.
internalSet	A Boolean specifying whether the Set should be marked as internal.

Optional arguments

None.

Return value

None.

Exceptions

None.

markSurfaceInternal(...)

This command marks the given Surface as internal or external.

Required arguments

setName	A string specifying the Surface name.
internalSurface	A Boolean specifying whether the Surface should be marked as internal.

Optional arguments

None.

Return value

None.

Exceptions

None.

isSetInternal(...)

This command returns a flag indicating whether the Set is Internal.

Required arguments

setName A string specifying the Set name.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

isSurfaceInternal(...)

This command returns a flag indicating whether the Surface is Internal.

Required arguments

surfaceName A string specifying the Surface name.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteSurfaces(...)

This command deletes the given surfaces from the assembly.

Required arguments

surfaceNames A sequence of Strings specifying the surface names that will be deleted from the assembly.

Optional arguments

None.

Return value

None.

Exceptions

None.

Part object

The following commands operate on Part objects. For more information about the Part object, see *Part object*.

Access

```
import regionToolset
```

`clashSets(...)`

This command prints a message describing the relationship between the contents of two sets. Possible outcomes are:

- Both sets are the same.
- Set 2 is a subset of set 1.
- Set 2 is a superset of set 1.
- Set 2 intersects set 1.
- Set 2 touches set 1 (their boundaries intersect).
- Set 2 and set 1 are disjoint.

This command accepts only positional arguments and has no keywords.

Required arguments

arg1	A Set or Surface object specifying set 1.
arg2	A Set or Surface object specifying set 2.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`markSetInternal(...)`

This command marks the given Set as internal or external.

Required arguments

setName	A string specifying the Set name.
internalSet	A Boolean specifying whether the Set should be marked as internal.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

markSurfaceInternal(...)

This command marks the given Surface as internal or external.

Required arguments

setName	A string specifying the Surface name.
internalSurface	A Boolean specifying whether the Surface should be marked as internal.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

isSetInternal(...)

This command returns a flag indicating whether the Set is Internal.

Required arguments

setName	A string specifying the Set name.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

isSurfaceInternal(...)

This command returns a flag indicating whether the Surface is Internal.

Required arguments

surfaceName A string specifying the Surface name.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteSets(...)

This command deletes the given sets from the part.

Required arguments

setNames A sequence of Strings specifying the set names that will be deleted from the part.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

deleteSurfaces(...)

This command deletes the given surfaces from the part.

Required arguments

surfaceNames A sequence of Strings specifying the surface names that will be deleted from the part.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Region object

The purpose of the Region object is to provide a link between an attribute and the geometric or discrete entities to which the attribute is applied. An attribute (Load, BC, IC, etc.) is applied to one or more Region objects; each Region object in turn is associated with a picked Set or Surface or with a named Set or Surface. The Region object provides a unified interface (or "façade") to data and functionality located at the Set or Surface.

A script that applies an attribute to a picked Set or Surface requires the explicit creation of a Region object and will implicitly create a matching internal Set or Surface. Conversely, a script that applies an attribute to a named Set or Surface requires the explicit creation of that Set or Surface (see 39.4) and will implicitly create a region object.

The reference between Region and Set/Surface is by name (user-provided or internal.) If the Set/Surface is suppressed, deleted, or renamed, the Region object will not be able to find the associated Set/Surface, and will communicate that to the attribute, but will not be affected otherwise. If a Set/Surface with the same name becomes available (only possible with user-provided names) the Region object will re-establish the connection. Another way of correcting this problem is to re-apply the attribute.

Wherever a particular Load, BC, IC, etc. command accepts a named set or a named surface, that command will also accept a Region object. For example,

```
myRegion = regionToolset.Region(edges=edges1)
mdb.models['Model-1'].DisplacementBC(name='BC-1',
    createStepName='Initial', region=myRegion, u1=SET,
    u2=SET)
myRegion = regionToolset.Region(elements=e[1:100])
p = mdb.models['mirror'].parts['COLLAR_MIRROR-1']
p.SectionAssignment(region=myRegion, sectionName='Section-1')
```

Abaqus does not provide a regions repository; as an alternative, you should assign a variable to a Region object and refer to the variable. The lifecycle of a Region object is similar to the lifecycle of a *Leaf* object used by display groups; as a result, you should use a Region object immediately after you create it. The contents of a Region object are not intended to survive regeneration. If you use an out-of-date Region object, the script is unlikely to function correctly.

Querying an attribute's Region will return a Region tuple. The contents of the tuple are the set name followed by the owners of the set and three flags expressed as integers. The flags indicate the region space, the type of region and the whether the region is an internal set.

Access

```
import regionToolset
```

Region(...)

This command creates a set-like region. For example,

```
myRegion = regionToolset.Region(vertices=v[2:4],
    edges=e[4:5]+e[6:9])
```

The arguments are the same as the arguments to the Set method, except for the *name* argument.

In most cases, the constructor will be called with only one argument of sequences. The arguments *xVertices*, *xEdges*, and *xFaces* are used to exclude lower-dimension entities and to provide finer control on the content of the region. For example, the following statement defines a region enclosing a square face but without two of its edges:

```
region = regionToolset.Region(faces=f[3:4], xEdges=e[1:3])
```

Path

```
Region
```

Required arguments

None.

Optional arguments

elements A sequence of Element objects. The default value is `None`.

nodes A sequence of Node objects. The default value is `None`.

vertices A sequence of Vertex objects. The default value is `None`.

edges A sequence of Edge objects. The default value is `None`.

faces A sequence of Face objects. The default value is `None`.

cells A sequence of Cell objects. The default value is `None`.

referencePoints A sequence of ReferencePoint objects. The default value is an empty sequence.

xVertices A sequence of Vertex objects that excludes specific vertices from the region. The default value is `None`.

xEdges A sequence of Vertex objects that excludes specific edges from the region. The default value is `None`.

xFaces A sequence of Vertex objects that excludes specific faces from the region. The default value is `None`.

skinFaces A tuple of tuples specifying a skin name and the sequence of faces associated with this skin. Valid only for geometric regions on 3D and 2D parts.

skinEdges A tuple of tuples specifying a skin name and the sequence of edges associated with this skin. Valid only for geometric regions on Axisymmetric parts.

stringerEdges A tuple of tuples specifying a stringer name and the sequence of edges associated with this stringer. Valid only for geometric regions on 3D and 2D parts.

Return value

A Region object.

Exceptions

None.

Region(...)

This command creates a surface-like region. For example,

```
myRegion = regionToolset.Region(side1Faces=f[12:14])
```

The arguments are the same as the arguments to the Surface method, except for the *name* argument.

Path

```
Region
```

Required arguments

None.

Optional arguments

In most cases, you will provide only one argument of sequences.

side1Faces
A sequence of Face objects (surface applies to SIDE1 of face). The default value is `None`.

side2Faces
A sequence of Face objects (surface applies to SIDE2 of face). The default value is `None`.

side12Faces
A sequence of Face objects (surface applies to both SIDE1 and SIDE2 of face). The default value is `None`.

side1Edges
A sequence of Edge objects (surface applies to SIDE1 of edge). The default value is `None`.

side2Edges
A sequence of Edge objects (surface applies to SIDE2 of edge). The default value is `None`.

end1Edges
A sequence of Edge objects (surface applies to END1 of edge). The default value is `None`.

end2Edges
A sequence of Edge objects (surface applies to END2 of edge). The default value is `None`.

circumEdges
A sequence of Edge objects (surface applies circumferentially to edge). The default value is `None`.

face1Elements
A sequence of Element objects (surface applies to FACE1 of element). The default value is `None`.

face2Elements
A sequence of Element objects (surface applies to FACE2 of element). The default value is `None`.

face3Elements
A sequence of Element objects (surface applies to FACE3 of element). The default value is `None`.

face4Elements
A sequence of Element objects (surface applies to FACE4 of element). The default value is `None`.

face5Elements
A sequence of Element objects (surface applies to FACE5 of element). The default value is `None`.

face6Elements
A sequence of Element objects (surface applies to FACE6 of element). The default value is `None`.

side1Elements
A sequence of Element objects (surface applies to SIDE1 of element). The default value is `None`.

side2Elements
A sequence of Element objects (surface applies to SIDE2 of element). The default value is `None`.

side12Elements
A sequence of Element objects (surface applies to both SIDE1 and SIDE2 of element). The default value is `None`.

end1Elements
A sequence of Element objects (surface applies to END1 of element). The default value is `None`.

end2Elements
A sequence of Element objects (surface applies to END2 of element). The default value is `None`.

circumElements
A sequence of Element objects (surface applies to circumference of element). The default value is `None`.

Return value

A Region object.

Exceptions

None.

Members

The Region object has no members.

Set object

If a set spans more than one part instance, the members *vertices*, *edges*, *faces*, *cells*, *elements*, or *nodes* return a sequence of all the queried vertices/edges/faces/celss/elements/nodes respectively for each part instance contained in the set. For example:

```
assembly=mdb.models['Transmission'].rootAssembly
clutchElements=assembly.instances['Clutch'].elements
clutchSet=clutchElements[16:18]+clutchElements[78:80]
housingElements=assembly.instances['Housing'].elements
housingSet=housingElements[45:48]
transmissionSet=assembly.Set(name='TransmissionSet', \
                             elements=(clutchSet, housingSet))

len(transmissionSet.elements)=7
transmissionSet.elements[0]=mdb.models['Transmission'].rootAssembly.instances['Clutch-1'].elements[16]

transmissionSet.elements[6]=mdb.models['Transmission'].rootAssembly.instances['housing-1'].elements[47]
```

Access

```
import part
mdb.models[name].parts[name].allInternalSets[name]
mdb.models[name].parts[name].allSets[name]
mdb.models[name].parts[name].sets[name]
import assembly
mdb.models[name].rootAssembly.allInstances[name].sets[name]
mdb.models[name].rootAssembly.allInternalSets[name]
mdb.models[name].rootAssembly.allSets[name]
mdb.models[name].rootAssembly.instances[name].sets[name]
mdb.models[name].rootAssembly.modelInstances[i].sets[name]
mdb.models[name].rootAssembly.sets[name]
```

Set(...)

This method creates a set from a sequence of objects in a model database.

Path

mdb.models[*name*].parts[*name*].Set

mdb.models[*name*].rootAssembly.Set

Required arguments

name A String specifying the repository key.

Optional arguments

At least one sequence argument must be provided—*elements*, *nodes*, *vertices*, *edges*, *faces*, *cells*, or *referencePoints*. The arguments *xVertices*, *xEdges*, and *xFaces* are used to exclude lower-dimension entities and to provide finer control

on the content of the set. For example, the following statement defines a region enclosing a square face but without two of its edges:

```
set = mdb.models['Model-1'].rootAssembly.Set(name='mySet', \
        faces=f[3:4], \
                                                xEdges=e[1:3])
```

nodes
A sequence of *MeshNode* objects. The default value is `None`.

elements
A sequence of *MeshElement* objects. The default value is `None`.

region
A Region object specifying other objects to be included in the set. The default value is `None`.

vertices
A sequence of Vertex objects. The default value is `None`.

edges
A sequence of Edge objects. The default value is `None`.

faces
A sequence of Face objects. The default value is `None`.

cells
A sequence of Cell objects. The default value is `None`.

xVertices
A sequence of Vertex objects that excludes specific vertices from the set. The default value is `None`.

xEdges
A sequence of Edge objects that excludes specific edges from the set. The default value is `None`.

xFaces
A sequence of Face objects that excludes specific faces from the set. The default value is `None`.

referencePoints
A sequence of ReferencePoint objects. The default value is an empty sequence.

skinFaces
A tuple of tuples specifying a skin name and the sequence of faces associated with this skin. Valid only for geometric regions on 3D and 2D parts.

skinEdges
A tuple of tuples specifying a skin name and the sequence of edges associated with this skin. Valid only for geometric regions on Axisymmetric parts.

stringerEdges
A tuple of tuples specifying a stringer name and the sequence of edges associated with this stringer. Valid only for geometric regions on 3D and 2D parts.

Return value

A Set object.

Exceptions

InvalidNameError.

Set(...)

This method copies a set from an existing set.

Path

mdb.models[*name*].parts[*name*].Set

mdb.models[*name*].rootAssembly.Set

Required arguments

name	A String specifying the name of the set.
objectToCopy	A Set object to be copied.

Optional arguments

None.

Return value

A Set object.

Exceptions

InvalidNameError.

SetByBoolean(...)

This method creates a set by performing a boolean operation on two or more input sets.

Path

mdb.models[*name*].parts[*name*].SetByBoolean

mdb.models[*name*].rootAssembly.SetByBoolean

Required arguments

name	A String specifying the repository key.
sets	A sequence of Set objects.

Optional arguments

operation	A SymbolicConstant specifying the boolean operation to perform. Possible values are UNION, INTERSECTION, and DIFFERENCE. The default value is UNION. Note that if DIFFERENCE is specified, the order of the given input sets is important; All sets specified after the first one are subtracted from the first one.

Return value

A Set object.

Exceptions

InvalidNameError.

SetFromColor(...)

This method creates a set containing faces of the part marked with a specified color attribute. Third-party applications can assign color attributes to faces, and the color attribute can be imported into Abaqus from an ACIS file. You can use this method to create sets only on parts; however, you can access the sets from instances of the parts in the assembly.

Path

mdb.models[*name*].parts[*name*].SetFromColor

Required arguments

name
A String specifying the repository key.

color
A sequence of three Ints specifying the RGB color. Values can range from 0 to 255. The first integer is for red, the second for green, and the third for blue. For example, a face colored in yellow is identified by:

```
color=(255,255,0)
```

Optional arguments

None.

Return value

A Set object.

Exceptions

InvalidNameError.

SetFromElementLabels(...)

This method creates a set from a sequence of element labels in a model database.

Path

mdb.models[*name*].parts[*name*].SetFromElementLabels

mdb.models[*name*].rootAssembly.SetFromElementLabels

Required arguments

name
A String specifying the repository key.

elementLabels

A sequence of element labels. An element label is a sequence of Int element identifiers. For example, for a part:

```
elementLabels=(2,3,5,7)
```

For an assembly:

```
elementLabels=(('Instance-1', (2,3,5,7)),
      ('Instance-2', (1,2,3)))
```

Optional arguments

None.

Return value

A Set object.

Exceptions

InvalidNameError.

SetFromNodeLabels(...)

This method creates a set from a sequence of node labels in a model database.

Path

mdb.models[*name*].parts[*name*].SetFromNodeLabels

mdb.models[*name*].rootAssembly.SetFromNodeLabels

Required arguments

name

A String specifying the repository key.

nodeLabels

A sequence of node labels. A node label is a sequence of Int node identifiers. For example, for a part:

```
nodeLabels=(2,3,5,7)
```

For an assembly:

```
nodeLabels=(('Instance-1', (2,3,5,7)), ('Instance-2', (1,2,3)))
```

Optional arguments

unsorted

A Boolean specifying whether the created set is unsorted. The default value is False.

Return value

A Set object.

Exceptions

InvalidNameError.

MapSetsFromOdb(...)

This method creates sets based on mapping sets from element centroid locations in an Odb.

Path

mdb.models[*name*].parts[*name*].MapSetsFromOdb

Required arguments

odbPath	A String specifying the path to the ODB containing the source sets.
odbSets	A list of names of sets on the ODB to map.

Optional arguments

partSets	A list of names of sets to construct or use corresponding to the ODB sets.
method	An enum to specify OVERWRITE, APPEND, INTERSECT, or REMOVE. The default is OVERWRITE.

Return value

A Set object or a tuple of Set objects.

Exceptions

None.

Members

The Set object has the following members:

elements	A *MeshElementArray* object.
nodes	A *MeshNodeArray* object.
vertices	A *VertexArray* object.
edges	An *EdgeArray* object.
faces	A *FaceArray* object.
cells	A *CellArray* object.
xVertices	A *VertexArray* object.
xEdges	An *EdgeArray* object.

xFaces

A *FaceArray* object.

referencePoints

A *ReferencePointArray* object.

Surface object

The Surface object stores surfaces selected from the assembly. A surface is comprised of geometric or discrete entities but not both. An instance of a Surface object is available from the *surface* member of the Assembly object.

Access

```
import part
mdb.models[name].parts[name].allInternalSurfaces[name]
mdb.models[name].parts[name].allSurfaces[name]
mdb.models[name].parts[name].surfaces[name]
import assembly
mdb.models[name].rootAssembly.allInstances[name].surfaces[name]
mdb.models[name].rootAssembly.allInternalSurfaces[name]
mdb.models[name].rootAssembly.allSurfaces[name]
mdb.models[name].rootAssembly.instances[name].surfaces[name]
mdb.models[name].rootAssembly.modelInstances[i].surfaces[name]
mdb.models[name].rootAssembly.surfaces[name]
```

Surface(...)

This method creates a surface from a sequence of objects in a model database. The surface will apply to the sides specified by the arguments.For example,

```
surface=mdb.models['Model-1'].parts['Part-1'].Surface(side1Faces=side1Faces,
name='Surf-1')
```

Path

mdb.models[*name*].parts[*name*].Surface

mdb.models[*name*].rootAssembly.Surface

Required arguments

On three-dimensional solid faces, you can use the following arguments:

- *side1Faces*
- *side2Faces*

On three-dimensional shell faces, you can use the following arguments:

- *side1Faces*
- *side2Faces*
- *side12Faces*

On three-dimensional wire edges, you can use the following arguments:

- *end1Edges*
- *end2Edges*
- *circumEdges*

On three-dimensional or two-dimensional or axisymmetric edges, you can use the following arguments:

- *side1Edges*

- *side2Edges*

On two-dimensional or axisymmetric shell elements, you can use the following arguments:

- *face1Elements*
- *face2Elements*
- *face3Elements*
- *face4Elements*

On solid elements, you can use the following arguments:

- *face1Elements*
- *face2Elements*
- *face3Elements*
- *face4Elements*
- *face5Elements*
- *face6Elements*

On three-dimensional shell elements, you can use the following arguments:

- *side1Elements*
- *side2Elements*
- *side12Elements*

On three-dimensional wire elements, you can use the following arguments:

- *end1Elements*
- *end2Elements*
- *circumElements*

On two-dimensional or axisymmetric wire elements, you can use the following arguments:

- *side1Elements*
- *side2Elements*

name A String specifying the repository key. The default value is an empty string.

Optional arguments

side1Elements A sequence of *MeshElement* objects (surface applies to SIDE1 of element). The default value is `None`.

side2Elements A sequence of *MeshElement* objects (surface applies to SIDE2 of element). The default value is `None`.

side12Elements A sequence of MeshElement objects (surface applies to both SIDE1 and SIDE2 of element). The default value is `None`.

end1Elements
A sequence of MeshElement objects (surface applies to END1 of element). The default value is `None`.

end2Elements
A sequence of MeshElement objects (surface applies to END2 of element). The default value is `None`.

circumElements
A sequence of MeshElement objects (surface applies to circumference of element). The default value is `None`.

face1Elements
A sequence of MeshElement objects (surface applies to FACE1 of element) or MeshFace objects. The default value is `None`.

face2Elements
A sequence of MeshElement objects (surface applies to FACE2 of element) or MeshFace objects. The default value is `None`.

face3Elements
A sequence of MeshElement objects (surface applies to FACE3 of element) or MeshFace objects. The default value is `None`.

face4Elements
A sequence of MeshElement objects (surface applies to FACE4 of element) or MeshFace objects. The default value is `None`.

face5Elements
A sequence of MeshElement objects (surface applies to FACE5 of element) or MeshFace objects. The default value is `None`.

face6Elements
A sequence of MeshElement objects (surface applies to FACE6 of element) or MeshFace objects. The default value is `None`.

side1Faces
A sequence of Face objects (surface applies to SIDE1 of face). The default value is `None`.

side2Faces
A sequence of Face objects (surface applies to SIDE2 of face). The default value is `None`.

side12Faces
A sequence of Face objects (surface applies to both SIDE1 and SIDE2 of face). The default value is `None`.

side1Edges
A sequence of Edge objects (surface applies to SIDE1 of edge). The default value is `None`.

side2Edges
A sequence of Edge objects (surface applies to SIDE2 of edge). The default value is `None`.

end1Edges
A sequence of Edge objects (surface applies to END1 of edge). The default value is `None`.

end2Edges
A sequence of Edge objects (surface applies to END2 of edge). The default value is `None`.

circumEdges A sequence of Edge objects (surface applies circumferentially to edge). The default value is `None`.

Return value

A Surface object.

Exceptions

InvalidNameError.

Surface(...)

This method copies a surface from an existing surface.

Path

mdb.models[*name*].parts[*name*].Surface

mdb.models[*name*].rootAssembly.Surface

Required arguments

name A String specifying the name of the surface.

objectToCopy A Surface object to be copied.

Optional arguments

None.

Return value

A Surface object.

Exceptions

InvalidNameError.

SurfaceByBoolean(...)

This method creates a surface by performing a boolean operation on two or more input surfaces.

Path

mdb.models[*name*].parts[*name*].SurfaceByBoolean

mdb.models[*name*].rootAssembly.SurfaceByBoolean

Required arguments

name A String specifying the repository key.

surfaces A sequence of Surface objects.

Optional arguments

operation A SymbolicConstant specifying the boolean operation to perform. Possible values are UNION, INTERSECTION, andDIFFERENCE. The default value is UNION. Note that if DIFFERENCE is specified, the order of the given input surfaces is important; All surfaces specified after the first one are subtracted from the first one.

Return value

A Surface object.

Exceptions

InvalidNameError.

SurfaceFromElsets(...)

This method creates a surface from a sequence of element sets in a model database.

Path

mdb.models[*name*].rootAssembly.SurfaceFromElsets

Required arguments

name A String specifying the repository key.

elementSetSeq A sequence of element sets. For example,

```
elementSetSeq=((elset1, S1),(elset2, S2))
```

where `elset1=mdb.models[name].rootAssembly.sets['Clutch']` and `S1` and `S2` indicate the side of the element set.

Optional arguments

None.

Return value

A Surface object.

Exceptions

InvalidNameError.

Members

The Surface object can have the following members:

edges

An *EdgeArray* object.

faces

A *FaceArray* object.

elements

A *MeshElementArray* object.

nodes

A *MeshNodeArray* object.

sides

A tuple of SymbolicConstants specifying the sides; for example, (SIDE1, SIDE2).

instances

A tuple of Ints specifying the instances. This member is not applicable for a Surface object on an output database.

Skin object

The Skin object stores information on skin reinforcements created on entities.

Access

```
import part
mdb.models[name].parts[name].skins[name]
import assembly
mdb.models[name].rootAssembly.allInstances[name].skins[name]
mdb.models[name].rootAssembly.instances[name].skins[name]
mdb.models[name].rootAssembly.skins[name]
```

Skin(...)

This method creates a skin from a sequence of objects in a model database. At least one of the optional arguments needs to be specified.

Path

mdb.models[*name*].parts[*name*].Skin

Required arguments

name A String specifying the repository key. The default value is an empty string.

Optional arguments

faces A sequence of Face objects specifying the faces on which skins should be created. Applicable to three and two dimensional parts.

edges A sequence of Edge objects specifying the edges on which skins should be created. Applicable to axisymmetric parts.

elementFaces A sequence of MeshFace objects specifying the mesh faces on which skins should be created. Applicable to three and two dimensional parts.

elementEdges A sequence of MeshEdge objects specifying the mesh edges on which skins should be created. Applicable to axisymmetric parts.

Return value

A Skin object.

Exceptions

InvalidNameError.

EditSkin(...)

This method modifies underlying entities of the selected skin. At least one of the optional arguments needs to be specified.

Path

mdb.models[*name*].parts[*name*].EditSkin

Required arguments

name A String specifying the repository key. The default value is an empty string.

Optional arguments

faces A sequence of Face objects specifying the faces on which skins should be created. Applicable to three and two dimensional parts.

edges A sequence of Edge objects specifying the edges on which skins should be created. Applicable to axisymmetric parts.

elementFaces A sequence of MeshFace objects specifying the mesh faces on which skins should be created. Applicable to three and two dimensional parts.

elementEdges A sequence of MeshEdge objects specifying the mesh edges on which skins should be created. Applicable to axisymmetric parts.

Return value

A Skin object.

Exceptions

InvalidNameError.

Members

The Skin object has the following members:

elements A *MeshElementArray* object.

edges An *EdgeArray* object.

faces A *FaceArray* object.

Stringer object

The Stringer object stores information on stringer reinforcements created on entities.

Access

```
import part
mdb.models[name].parts[name].stringers[name]
import assembly
mdb.models[name].rootAssembly.allInstances[name].stringers[name]
mdb.models[name].rootAssembly.instances[name].stringers[name]
mdb.models[name].rootAssembly.stringers[name]
```

Stringer(...)

This method creates a stringer from a sequence of objects in a model database. At least one of the optional arguments needs to be specified.

Path

mdb.models[*name*].parts[*name*].Stringer

Required arguments

name A String specifying the repository key. The default value is an empty string.

Optional arguments

edges A sequence of Edge objects specifying the edges on which stringers should be created. Applicable to three and two dimensional parts.

elementEdges A sequence of MeshEdge objects specifying the mesh edges on which stringers should be created. Applicable to three and two dimensional parts.

Return value

A Stringer object.

Exceptions

InvalidNameError.

EditStringer(...)

This method modifies underlying entities of the selected stringer. At least one of the optional arguments needs to be specified.

Path

mdb.models[*name*].parts[*name*].EditStringer

Required arguments

name A String specifying the repository key. The default value is an empty string.

Optional arguments

edges A sequence of Edge objects specifying the edges on which stringers should be created. Applicable to three and two dimensional parts.

elementEdges A sequence of MeshEdge objects specifying the mesh edges on which stringers should be created. Applicable to three and two dimensional parts.

Return value

A Stringer object.

Exceptions

InvalidNameError.

Members

The Stringer object has the following members:

elements A *MeshElementArray* object.

edges An *EdgeArray* object.

Section commands

The Section commands are used to create sections and profiles with their associated properties and behavior. The various section objects are all derived from the Section object. The various profile objects are all derived from the Profile object. See *Property commands* for the property assignment commands.

In this section:

- *Section object*
- *AcousticInfiniteSection object*
- *AcousticInterfaceSection object*
- *BeamSection object*
- *CohesiveSection object*
- *CompositeShellSection object*
- *CompositeSolidSection object*
- *ConnectorSection object*
- *EulerianSection object*
- *GasketSection object*
- *GeneralStiffnessSection object*
- *GeometryShellSection object*
- *HomogeneousShellSection object*
- *HomogeneousSolidSection object*
- *LayerProperties object*
- *MembraneSection object*
- *MPCSection object*
- *PEGSection object*
- *RebarLayers object*
- *SectionLayer object*
- *ShellSection object*
- *SolidSection object*
- *SurfaceSection object*
- *TransverseShearBeam object*
- *TransverseShearShell object*
- *TrussSection object*

Section object

The Section object defines the properties of a section. The Section object is the abstract base type for other Section objects. The Section object has no explicit constructor. The methods and members of the Section object are common to all objects derived from the Section.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

sectionsFromOdb(...)

This method creates Section objects by reading an output database. The new sections are placed in the sections repository.

Path

mdb.models[*name*].sectionsFromOdb

Required arguments

fileName
A String specifying the name of the output database file (including the `.odb` extension) to be read. This String can also be the full path to the output database file if it is located in another directory.

Optional arguments

None.

Return value

A list of Section objects.

Exceptions

None.

Members

The Section object has the following member:

name
A String specifying the repository key.

AcousticInfiniteSection object

The AcousticInfiniteSection object defines the properties of an acoustic section.

The AcousticInfiniteSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

AcousticInfiniteSection(...)

This method creates an AcousticInfiniteSection object.

Path

```
mdb.models[name].AcousticInfiniteSection
session.odbs[name].AcousticInfiniteSection
```

Required arguments

name A String specifying the repository key.

material A String specifying the name of the material.

Optional arguments

thickness A Float specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

order An Int specifying the number of ninth-order polynomials that will be used to resolve the variation of the acoustic field in the infinite direction. Possible values are 0 <*order*≤ 10. The default value is 10.

Return value

An AcousticInfiniteSection object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the AcousticInfiniteSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticInfiniteSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The AcousticInfiniteSection object has members with the same names and descriptions as the arguments to the *AcousticInfiniteSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

AcousticInterfaceSection object

The AcousticInterfaceSection object defines the properties of an acoustic section.

The AcousticInterfaceSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

AcousticInterfaceSection(...)

This method creates an AcousticInterfaceSection object.

Path

```
mdb.models[name].AcousticInterfaceSection
session.odbs[name].AcousticInterfaceSection
```

Required arguments

name A String specifying the repository key.

Optional arguments

thickness A Float specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

Return value

An AcousticInterfaceSection object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the AcousticInterfaceSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AcousticInterfaceSection* method, except for the *name* argument.

Return value
`None.`

Exceptions
RangeError.

Members
The AcousticInterfaceSection object has members with the same names and descriptions as the arguments to the *AcousticInterfaceSection* method.

Corresponding analysis keywords
- **INTERFACE*

BeamSection object

The BeamSection object defines the properties of a beam section.

The BeamSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

BeamSection(...)

This method creates a BeamSection object.

Path

```
mdb.models[name].BeamSection
session.odbs[name].BeamSection
```

Required arguments

name	A String specifying the repository key.
integration	A SymbolicConstant specifying the integration method for the section. Possible values are BEFORE_ANALYSIS and DURING_ANALYSIS.
profile	A String specifying the name of the profile. This argument represents the start profile in case of *beamShape*=TAPERED.

Optional arguments

poissonRatio	A Float specifying the Poisson's ratio of the section. The default value is 0.0.
thermalExpansion	A Boolean specifying whether to use thermal expansion data. The default value is OFF.
temperatureDependency	A Boolean specifying whether the data depend on temperature. The default value is OFF.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.
density	`None` or a Float specifying the density of the section. The default value is `None`.
referenceTemperature	`None` or a Float specifying the reference temperature of the section. The default value is `None`.
temperatureVar	A SymbolicConstant specifying the temperature variation for the section. Possible values are LINEAR and INTERPOLATED. The default value is LINEAR.

alphaDamping

A Float specifying the α_R factor to create mass proportional damping in direct-integration dynamics. The default value is 0.0.

betaDamping

A Float specifying the β_R factor to create stiffness proportional damping in direct-integration dynamics. The default value is 0.0.

compositeDamping

A Float specifying the fraction of critical damping to be used in calculating composite damping factors for the modes (for use in modal dynamics). The default value is 0.0.

useFluidInertia

A Boolean specifying whether added mass effects will be simulated. The default value is OFF.

submerged

A SymbolicConstant specifying whether the section is either full submerged or half submerged. This argument applies only when *useFluidInertia* = True. Possible values are FULLY and HALF. The default value is FULLY.

fluidMassDensity

`None` or a Float specifying the mass density of the fluid. This argument applies only when *useFluidInertia* = True and must be specified in that case. The default value is `None`.

crossSectionRadius

`None` or a Float specifying the radius of the cylindrical cross-section. This argument applies only when *useFluidInertia* = True and must be specified in that case. The default value is `None`.

lateralMassCoef

A Float specifying the added mass coefficient, C_A, for lateral motions of the beam. This argument applies only when*useFluidInertia* = True. The default value is 1.0.

axialMassCoef

A Float specifying the added mass coefficient, $C_{(A-E)}$, for motions along the axis of the beam. This argument affects only the term added to the free end(s) of the beam, and applies only when *useFluidInertia* = True. The default value is 0.0.

massOffsetX

A Float specifying the local 1-coordinate of the center of the cylindrical cross-section with respect to the beam cross-section. This argument applies only when *useFluidInertia* = True. The default value is 0.0.

massOffsetY

A Float specifying the local 2-coordinate of the center of the cylindrical cross-section with respect to the beam cross-section. This argument applies only when *useFluidInertia* = True. The default value is 0.0.

beamShape

A SymbolicConstant specifying the change in cross-section of the beam along length. Possible values are CONSTANT and TAPERED. The default value is CONSTANT. This parameter is available for manipulating the model database but not for the ODB API.

material

A String specifying the name of the material. The default value is an empty string. The material is required when *integration* is "DURING_ANALYSIS".

table
A sequence of sequences of Floats specifying the items described below. The default value is an empty sequence.

outputPts
A sequence of pairs of Floats specifying the positions at which output is requested. The default value is an empty sequence.

centroid
A pair of Floats specifying the *X*–*Y* coordinates of the centroid. The default value is (0.0, 0.0).

shearCenter
A pair of Floats specifying the *X*–*Y* coordinates of the shear center. The default value is (0.0, 0.0).

profileEnd
A String specifying the name of the end profile. The type of the end profile must be same as that of the start profile. This argument is valid only when *beamShape*=TAPERED. The default value is an empty string. This parameter is available for manipulating the model database but not for the ODB API.

beamSectionOffset
A pair of Floats specifying the *X*- and *Y*-coordinates to define an offset for the cross-section origin from the beam axis with respect to the default (that is, without an offset) local (1, 2) axis system. The default beamSectionOffset is (0, 0).

Table data

The table data specify the following:

- E, the Young's modulus of the section.
- G, the torsional shear modulus of the section.
- Thermal expansion coefficient, if using thermal expansion.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BeamSection object.

Exceptions

None.

`setValues(...)`

This method modifies the BeamSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BeamSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The BeamSection object has members with the same names and descriptions as the arguments to the *BeamSection* method.

In addition, the BeamSection object can have the following member:

beamTransverseShear A *TransverseShearBeam* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **BEAM GENERAL SECTION*
- **BEAM SECTION*
- **BEAM FLUID INERTIA*
- **CENTROID*
- **DAMPING*
- **SHEAR CENTER*
- **SECTION POINTS*

CohesiveSection object

The CohesiveSection object defines the properties of a cohesive section.

The CohesiveSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

CohesiveSection(...)

This method creates a CohesiveSection object.

Path

```
mdb.models[name].CohesiveSection
session.odbs[name].CohesiveSection
```

Required arguments

name
A String specifying the repository key.

response
A SymbolicConstant specifying the geometric assumption that defines the constitutive behavior of the cohesive elements. Possible values are TRACTION_SEPARATION, CONTINUUM, and GASKET.

material
A String specifying the name of the material.

Optional arguments

initialThicknessType
A SymbolicConstant specifying the method used to compute the initial thickness. Possible values are:

- SOLVER_DEFAULT, specifying that Abaqus will use the analysis product default
- GEOMETRY, specifying that Abaqus will compute the thickness from the nodal coordinates of the elements.
- SPECIFY, specifying that Abaqus will use the value given for *initialThickness*

The default value is SOLVER_DEFAULT.

initialThickness
A Float specifying the initial thickness for the section. The *initialThickness* argument applies only when *initialThicknessType*=SPECIFY. The default value is 1.0.

outOfPlaneThickness
`None` or a Float specifying the out-of-plane thickness for the section. The default value is `None`.

Return value

A CohesiveSection object.

Exceptions

RangeError and InvalidNameError.

`setValues(...)`

This method modifies the CohesiveSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CohesiveSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The CohesiveSection object has members with the same names and descriptions as the arguments to the *CohesiveSection* method.

Corresponding analysis keywords

- **COHESIVE SECTION*

CompositeShellSection object

The CompositeShellSection object defines the properties of a composite shell section.

The CompositeShellSection object is derived from the *GeometryShellSection* object.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

CompositeShellSection(...)

This method creates a CompositeShellSection object.

Path

```
mdb.models[name].parts[name].compositeLayups[i].CompositeShellSection
mdb.models[name].CompositeShellSection
session.odbs[name].CompositeShellSection
```

Required arguments

name A String specifying the repository key.

layup A *SectionLayerArray* object specifying the shell cross-section.

Optional arguments

symmetric A Boolean specifying whether or not the layup should be made symmetric by the analysis. The default value is OFF.

thicknessType A SymbolicConstant specifying the distribution used for defining the thickness of the elements. Possible values are UNIFORM, ANALYTICAL_FIELD, DISCRETE_FIELD, NODAL_ANALYTICAL_FIELD, and NODAL_DISCRETE_FIELD. The default value is UNIFORM.

preIntegrate A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (ON) or integrated during the analysis (OFF). The default value is OFF.

poissonDefinition A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson

A Float specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

integrationRule

A SymbolicConstant specifying the shell section integration rule. Possible values are SIMPSON and GAUSS. The default value is SIMPSON.

temperature

A SymbolicConstant specifying the mode used for temperature and field variable input across the section thickness. Possible values are GRADIENT and POINTWISE. The default value is GRADIENT.

idealization

A SymbolicConstant specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to ON. Possible values are NO_IDEALIZATION, SMEAR_ALL_LAYERS, MEMBRANE, and BENDING. The default value is NO_IDEALIZATION.

nTemp

`None` or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*=POINTWISE. The default value is `None`.

thicknessModulus

`None` or a Float specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is `None`.

useDensity

A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density

A Float specifying the value of density to apply to this section. The default value is 0.0.

layupName

A String specifying the layup name for this section. The default value is an empty string.

thicknessField

A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD. The default value is an empty string.

nodalThicknessField

A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*=NODAL_ANALYTICAL_FIELD or *thicknessType*=NODAL_DISCRETE_FIELD. The default value is an empty string.

Return value

A CompositeShellSection object.

Exceptions

None.

`setValues(...)`

This method modifies the CompositeShellSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CompositeShellSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The CompositeShellSection object has members with the same names and descriptions as the arguments to the *CompositeShellSection* method.

In addition, the CompositeShellSection object can have the following members:

rebarLayers	A *RebarLayers* object specifying reinforcement properties.
transverseShear	A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

CompositeSolidSection object

The CompositeSolidSection object defines the properties of a composite solid section.

The CompositeSolidSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

CompositeSolidSection(...)

This method creates a CompositeSolidSection object.

Path

```
mdb.models[name].CompositeSolidSection
session.odbs[name].CompositeSolidSection
```

Required arguments

name	A String specifying the repository key.
layup	A *SectionLayerArray* object specifying the solid cross-section.

Optional arguments

symmetric	A Boolean specifying whether or not the layup should be made symmetric by the analysis. The default value is OFF.
layupName	A String specifying the layup name for this section. The default value is an empty string.

Return value

A CompositeSolidSection object.

Exceptions

None.

setValues(...)

This method modifies the CompositeSolidSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CompositeSolidSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The CompositeSolidSection object has members with the same names and descriptions as the arguments to the *CompositeSolidSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

ConnectorSection object

A ConnectorSection object describes the connection type and the behavior of a connector.

The ConnectorSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

ConnectorSection(...)

This method creates a ConnectorSection object.

Path

```
mdb.models[name].ConnectorSection
session.odbs[name].ConnectorSection
```

Required arguments

name

A String specifying the repository key.

Optional arguments

assembledType

A SymbolicConstant specifying the assembled connection type. Possible values are:

- NONE
- BEAM
- BUSHING
- CVJOINT
- CYLINDRICAL
- HINGE
- PLANAR
- RETRACTOR
- SLIPRING
- TRANSLATOR
- UJOINT
- WELD

The default value is NONE.

You cannot include the *assembledType* argument if *translationalType* or *rotationalType* are given a value other than NONE. At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given a value other than NONE.

rotationalType

A SymbolicConstant specifying the basic rotational connection type. Possible values are:

- NONE
- ALIGN
- CARDAN
- CONSTANT_VELOCITY
- EULER
- FLEXION_TORSION
- FLOW_CONVERTER
- PROJECTION_FLEXION_TORSION
- REVOLUTE
- ROTATION
- ROTATION_ACCELEROMETER
- UNIVERSAL

The default value is NONE.

You cannot include the *rotationalType* argument if *assembledType* is given a value other than NONE. At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given an value other than NONE.

translationalType

A SymbolicConstant specifying the basic translational connection type. Possible values are:

- NONE
- ACCELEROMETER
- AXIAL
- CARTESIAN
- JOIN
- LINK
- PROJECTION_CARTESIAN
- RADIAL_THRUST
- SLIDE_PLANE
- SLOT

The default value is NONE.

You cannot include the *translationalType* argument if *assembledType* is given a value other than NONE. At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given an value other than NONE.

integration

A SymbolicConstant specifying the time integration scheme to use for analysis. This argument is applicable only to an Abaqus/Explicit analysis. Possible values are UNSPECIFIED, IMPLICIT, and EXPLICIT. The default value is UNSPECIFIED.

u1ReferenceLength

`None` or a Float specifying the reference length associated with constitutive response for the first component of relative motion. The default value is `None`.

u2ReferenceLength

`None` or a Float specifying the reference length associated with constitutive response for the second component of relative motion. The default value is `None`.

u3ReferenceLength

`None` or a Float specifying the reference length associated with constitutive response for the third component of relative motion. The default value is `None`.

ur1ReferenceAngle

`None` or a Float specifying the reference angle in degrees associated with constitutive response for the fourth component of relative motion. The default value is `None`.

ur2ReferenceAngle

`None` or a Float specifying the reference angle in degrees associated with constitutive response for the fifth component of relative motion. The default value is `None`.

ur3ReferenceAngle

`None` or a Float specifying the reference angle in degrees associated with constitutive response for the sixth component of relative motion. The default value is `None`.

massPerLength

`None` or a Float specifying the mass per unit reference length of belt material. This argument is applicable only when *assembledType*=SLIPRING, and must be specified in that case. The default value is `None`.

contactAngle

`None` or a Float specifying the contact angle made by the belt wrapping around node b. This argument is applicable only to an Abaqus/Explicit analysis, and only when *assembledType*=SLIPRING. The default value is `None`.

materialFlowFactor

A Float specifying the scaling factor for material flow at node b. This argument is applicable only when *assembledType*=RETRACTOR or *rotationalType*=FLOW_CONVERTER. The default value is 1.0.

regularize

A Boolean specifying whether or not all tabular data associated with the *behaviorOptions* will be regularized. This argument is applicable only for an Abaqus/Explicit analysis. The default value is ON.

defaultTolerance

A Boolean specifying whether or not the default regularization tolerance will be used for all tabular data associated with the *behaviorOptions*. This argument is applicable only for an Abaqus/Explicit analysis and only if *regularize*=ON. The default value is ON.

regularization

A Float specifying the regularization increment to be used for all tabular data associated with the *behaviorOptions*. This argument is applicable only for an Abaqus/Explicit analysis and only if *regularize*=ON and *defaultTolerance*=OFF. The default value is 0.03.

extrapolation

A SymbolicConstant specifying the extrapolation technique to be used for all tabular data associated with the *behaviorOptions*. Possible values are CONSTANT and LINEAR. The default value is CONSTANT.

behaviorOptions

A *ConnectorBehaviorOptionArray* object.

Return value

A ConnectorSection object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the ConnectorSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ConnectorSection object has members with the same names and descriptions as the arguments to the *ConnectorSection* method.

Corresponding analysis keywords

- **CONNECTOR SECTION*
- **CONNECTOR BEHAVIOR*
- **CONNECTOR CONSTITUTIVE REFERENCE*

EulerianSection object

The EulerianSection object defines the properties of an Eulerian section.

The EulerianSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

EulerianSection(...)

This method creates an EulerianSection object.

Path

```
mdb.models[name].EulerianSection
session.odbs[name].EulerianSection
```

Required arguments

name A String specifying the repository key.

data A String-to-String Dictionary specifying a dictionary mapping Material instance names to Material names. Internally the specified mapping gets sorted on Material instance name.

Optional arguments

None.

Return value

An EulerianSection object.

Exceptions

None.

setValues(...)

This method modifies the EulerianSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EulerianSection* method, except for the *name* argument.

Return value
`None.`

Exceptions
None.

Members
The EulerianSection object has members with the same names and descriptions as the arguments to the *EulerianSection* method.

Corresponding analysis keywords
- **EULERIAN SECTION*

GasketSection object

The GasketSection object defines the properties of a gasket section.

The GasketSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

GasketSection(...)

This method creates a GasketSection object.

Path

```
mdb.models[name].GasketSection
session.odbs[name].GasketSection
```

Required arguments

name
A String specifying the repository key.

material
A String specifying the name of the material of which the gasket is made or material that defines gasket behavior.

Optional arguments

crossSection
A Float specifying the cross-sectional area, width, or out-of-plane thickness, if applicable, depending on the gasket element type. The default value is 1.0.

initialGap
A Float specifying the initial gap. The default value is 0.0.

initialThickness
The SymbolicConstant DEFAULT or a Float specifying the initial gasket thickness. If DEFAULT is specified, the initial thickness is determined using nodal coordinates. The default value is DEFAULT.

initialVoid
A Float specifying the initial void. The default value is 0.0.

stabilizationStiffness
The SymbolicConstant DEFAULT or a Float specifying the default stabilization stiffness used in all but link elements to stabilize gasket elements that are not supported at all nodes, such as those that extend outside neighboring components. If DEFAULT is specified, a value is used equal to 10^{-9} times the initial compressive stiffness in the thickness direction. The default value is DEFAULT.

Return value

A GasketSection object.

Exceptions

InvalidNameError and ValueError.

setValues(...)

This method modifies the GasketSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GasketSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

ValueError.

Members

The GasketSection object has members with the same names and descriptions as the arguments to the *GasketSection* method.

Corresponding analysis keywords

- **GASKET SECTION*

GeneralStiffnessSection object

The GeneralStiffnessSection object defines the properties of a shell section via the stiffness matrix.

The GeneralStiffnessSection object is derived from the *ShellSection* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

GeneralStiffnessSection(...)

This method creates a GeneralStiffnessSection object.

Path

```
mdb.models[name].GeneralStiffnessSection
session.odbs[name].GeneralStiffnessSection
```

Required arguments

name
A String specifying the repository key.

stiffnessMatrix
A sequence of Floats specifying the stiffness matrix for the section in the order D11, D12, D22, D13, D23, D33, ..., D66. Twenty-one entries must be given.

Optional arguments

referenceTemperature
`None` or a Float specifying the reference temperature for thermal expansion. The default value is `None`.

applyThermalStress
A Boolean specifying whether or not the section stiffness varies with thermal stresses. The default value is OFF.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is OFF.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

poissonDefinition
A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson
A Float specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

useDensity
A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density
A Float specifying the value of density to apply to this section. The default value is 0.0.

thermalStresses
A sequence of Floats specifying the generalized stress values caused by a unit temperature rise. Six entries must be given if the value of *applyThermalStress* is set to True. The default value is ("").

scalingData
A sequence of sequences of Floats specifying the scaling factors for given temperatures and/or field data. Each row should contain (Y, alpha, T, F1,...,Fn). The default value is an empty sequence.

Return value

A GeneralStiffnessSection object.

Exceptions

None.

`setValues(...)`

This method modifies the GeneralStiffnessSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GeneralStiffnessSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The GeneralStiffnessSection object has members with the same names and descriptions as the arguments to the *GeneralStiffnessSection* method.

In addition, the GeneralStiffnessSection object can have the following members:

rebarLayers
A *RebarLayers* object specifying reinforcement properties.

transverseShear A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL GENERAL SECTION*

GeometryShellSection object

The GeometryShellSection object defines the properties of a geometry shell section. The GeometryShellSection object has no explicit constructor and no methods. The GeometryShellSection object is an abstract base type.

The GeometryShellSection object is derived from the *ShellSection* object.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

Section(...)

This method creates a GeometryShellSection object.

Path

mdb.models[*name*].parts[*name*].compositeLayups[*name*].Section

Required arguments

None.

Optional arguments

nodalThicknessField A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*=NODAL_ANALYTICAL_FIELD or *thicknessType*=NODAL_DISCRETE_FIELD. The default value is an empty string.

thicknessField A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD. The default value is an empty string.

thicknessType A SymbolicConstant specifying the distribution used for defining the thickness of the elements. Possible values are UNIFORM, ANALYTICAL_FIELD, DISCRETE_FIELD, NODAL_ANALYTICAL_FIELD, and NODAL_DISCRETE_FIELD. The default value is UNIFORM.

preIntegrate A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (ON) or integrated during the analysis (OFF). The default value is OFF.

poissonDefinition A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson
A Float specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

integrationRule
A SymbolicConstant specifying the shell section integration rule. Possible values are SIMPSON and GAUSS. The default value is SIMPSON.

temperature
A SymbolicConstant specifying the mode used for temperature and field variable input across the section thickness. Possible values are GRADIENT and POINTWISE. The default value is GRADIENT.

nTemp
`None` or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*=POINTWISE. The default value is `None`.

thicknessModulus
`None` or a Float specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is `None`.

useDensity
A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density
A Float specifying the value of density to apply to this section. The default value is 0.0.

Return value

A GeometryShellSection object.

Exceptions

None.

Members

The GeometryShellSection object can have the following members:

name
A String specifying the repository key.

thicknessType
A SymbolicConstant specifying the distribution used for defining the thickness of the elements. Possible values are UNIFORM, ANALYTICAL_FIELD, DISCRETE_FIELD, NODAL_ANALYTICAL_FIELD, and NODAL_DISCRETE_FIELD. The default value is UNIFORM.

preIntegrate
A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (ON) or integrated during the analysis (OFF). The default value is OFF.

poissonDefinition

A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson

A Float specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

integrationRule

A SymbolicConstant specifying the shell section integration rule. Possible values are SIMPSON and GAUSS. The default value is SIMPSON.

temperature

A SymbolicConstant specifying the mode used for temperature and field variable input across the section thickness. Possible values are GRADIENT and POINTWISE. The default value is GRADIENT.

idealization

A SymbolicConstant specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to ON. Possible values are NO_IDEALIZATION, SMEAR_ALL_LAYERS, MEMBRANE, and BENDING. The default value is NO_IDEALIZATION.

nTemp

`None` or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*=POINTWISE. The default value is `None`.

thicknessModulus

`None` or a Float specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is `None`.

useDensity

A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density

A Float specifying the value of density to apply to this section. The default value is 0.0.

thicknessField

A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD. The default value is an empty string.

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

nodalThicknessField

A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*=NODAL_ANALYTICAL_FIELD or *thicknessType*=NODAL_DISCRETE_FIELD. The default value is an empty string.

transverseShear A *TransverseShearShell* object specifying the transverse shear stiffness properties.

HomogeneousShellSection object

The HomogeneousShellSection object defines the properties of a shell section.

The HomogeneousShellSection object is derived from the *GeometryShellSection* object.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

HomogeneousShellSection(...)

This method creates a HomogeneousShellSection object.

Path

```
mdb.models[name].parts[name].compositeLayups[i]\
.HomogeneousShellSection
mdb.models[name].HomogeneousShellSection
session.odbs[name].HomogeneousShellSection
```

Required arguments

name A String specifying the repository key.

material A String specifying the name of the section material.

Optional arguments

thickness A Float specifying the thickness of the section. The *thickness* argument applies only when *thicknessType*=UNIFORM. The default value is 0.0.

numIntPts An Int specifying the number of integration points to be used through the section. Possible values are *numIntPts*> 0. The default value is 5.

To use the default settings of the analysis products, set *numIntPts* to 5 if *integrationRule*=SIMPSON or set *numIntPts* to 7 if *integrationRule*=GAUSS.

thicknessType A SymbolicConstant specifying the distribution used for defining the thickness of the elements. Possible values are UNIFORM, ANALYTICAL_FIELD, DISCRETE_FIELD, NODAL_ANALYTICAL_FIELD, and NODAL_DISCRETE_FIELD. The default value is UNIFORM.

preIntegrate A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (ON) or integrated during the analysis (OFF). The default value is OFF.

poissonDefinition A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson A Float specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

integrationRule A SymbolicConstant specifying the shell section integration rule. Possible values are SIMPSON and GAUSS. The default value is SIMPSON.

temperature A SymbolicConstant specifying the mode used for temperature and field variable input across the section thickness. Possible values are GRADIENT and POINTWISE. The default value is GRADIENT.

idealization A SymbolicConstant specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to ON. Possible values are NO_IDEALIZATION, SMEAR_ALL_LAYERS, MEMBRANE, and BENDING. The default value is NO_IDEALIZATION.

nTemp `None` or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*=POINTWISE. The default value is `None`.

thicknessModulus `None` or a Float specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is `None`.

useDensity A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density A Float specifying the value of density to apply to this section. The default value is 0.0.

thicknessField A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD. The default value is an empty string.

nodalThicknessField A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*=NODAL_ANALYTICAL_FIELD or *thicknessType*=NODAL_DISCRETE_FIELD. The default value is an empty string.

Return value

A HomogeneousShellSection object.

Exceptions

None.

setValues(...)

This method modifies the HomogeneousShellSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HomogeneousShellSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The HomogeneousShellSection object has members with the same names and descriptions as the arguments to the *HomogeneousShellSection* method.

In addition, the HomogeneousShellSection object can have the following members:

rebarLayers A *RebarLayers* object specifying reinforcement properties.

transverseShear A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

HomogeneousSolidSection object

The HomogeneousSolidSection object defines the properties of a solid section.

The HomogeneousSolidSection object is derived from the *SolidSection* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

HomogeneousSolidSection(...)

This method creates a HomogeneousSolidSection object.

Path

```
mdb.models[name].HomogeneousSolidSection
session.odbs[name].HomogeneousSolidSection
```

Required arguments

name A String specifying the repository key.

material A String specifying the name of the material.

Optional arguments

thickness A Float specifying the thickness of the section. Possible values are `None` or greater than zero. The default value is 1.0.

Return value

A HomogeneousSolidSection object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the HomogeneousSolidSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HomogeneousSolidSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The HomogeneousSolidSection object has members with the same names and descriptions as the arguments to the *HomogeneousSolidSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

LayerProperties object

The LayerProperties object defines the properties of a layer of reinforcement for membrane, shell, and surface sections.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section.rebarLayers\
.layerTable[i]
mdb.models[name].sections[name].rebarLayers.layerTable[i]
import odbSection
session.odbs[name].sections[name].rebarLayers.layerTable[i]
```

LayerProperties(...)

This method creates a LayerProperties object.

Path

section.LayerProperties

odbSection.LayerProperties

Required arguments

barArea	A Float specifying the area per bar.
orientationAngle	A Float or a String specifying the orientation of the rebar. A Float specifies the angular orientation; a String specifies an orientation name.
layerName	A String specifying the name of the rebar layer.
material	A String specifying the name of the rebar material.

Optional arguments

barSpacing	A Float specifying the spacing of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to CONSTANT. The default value is 0.0.
layerPosition	A Float specifying the position of the rebar from the middle surface of the shell. *layerPosition* applies only for homogeneous shell sections and composite shell sections. The default value is 0.0.
spacingAngle	A Float specifying the spacing angle of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to ANGULAR. The default value is 0.0.
extensionRatio	A Float specifying the extension ratio for the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to LIFT_EQUATION. The default value is 0.0.

radius

A Float specifying the radius of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to LIFT_EQUATION. The default value is 0.0.

Return value

A LayerProperties object.

Exceptions

None.

Members

The LayerProperties object has members with the same names and descriptions as the arguments to the *LayerProperties* method.

Corresponding analysis keywords

- **REBAR LAYER*

MembraneSection object

The MembraneSection object defines the properties of a membrane section.

The MembraneSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

MembraneSection(...)

This method creates a MembraneSection object.

Path

```
mdb.models[name].MembraneSection
session.odbs[name].MembraneSection
```

Required arguments

name
A String specifying the repository key.

material
A String specifying the name of the material.

Optional arguments

thickness
A Float specifying the thickness for the section. Possible values are *thickness*> 0.0. The default value is 1.0.

thicknessType
A SymbolicConstant specifying the distribution used for defining the thickness of the elements. Possible values are UNIFORM, ANALYTICAL_FIELD, and DISCRETE_FIELD. The default value is UNIFORM.

poissonDefinition
A SymbolicConstant specifying whether to use the default value for the Poisson's ratio. Possible values are:

- DEFAULT, specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- VALUE, specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is DEFAULT.

poisson
A Float specifying the section Poisson's ratio. Possible values are −1.0 ≤*poisson*≤ 0.5. This argument is valid only when *poissonDefinition*=VALUE. The default value is 0.5.

thicknessField A String specifying the name of the *AnalyticalField* or *DiscreteField* object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*=ANALYTICAL_FIELD or *thicknessType*=DISCRETE_FIELD. The default value is an empty string.

Return value

A MembraneSection object.

Exceptions

RangeError and InvalidNameError.

`setValues(...)`

This method modifies the MembraneSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MembraneSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The MembraneSection object has members with the same names and descriptions as the arguments to the *MembraneSection* method, except for the *thicknessType* and *thicknessField* arguments.

In addition, the MembraneSection object can have the following member:

rebarLayers A *RebarLayers* object specifying reinforcement properties.

Corresponding analysis keywords

- **MEMBRANE SECTION*

MPCSection object

The MPCSection object defines the properties of a multi-point constraint section.

The MPCSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

MPCSection(...)

This method creates a MPCSection object.

Path

```
mdb.models[name].MPCSection
session.odbs[name].MPCSection
```

Required arguments

name
A String specifying the repository key.

mpcType
A SymbolicConstant specifying the MPC type of the section. Possible values are BEAM_MPC, ELBOW_MPC, PIN_MPC, LINK_MPC, TIE_MPC, and USER_DEFINED.

Optional arguments

userMode
A SymbolicConstant specifying the mode of the MPC when it is user-defined. Possible values are DOF_MODE and NODE_MODE. The default value is DOF_MODE.

The *userMode* argument applies only when *mpcType*=USER_DEFINED.

userType
An Int specifying to differentiate between different constraint types in a user-defined MPCSection. The default value is 0.

The *userType* argument applies only when *mpcType*=USER_DEFINED.

Return value

A MPCSection object.

Exceptions

RangeError and InvalidNameError.

Members

The MPCSection object has members with the same names and descriptions as the arguments to the *MPCSection* method.

Corresponding analysis keywords

- **MPC*

PEGSection object

The PEGSection object defines the properties of a solid section.

The PEGSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

PEGSection(...)

This method creates a PEGSection object.

Path

```
mdb.models[name].PEGSection
session.odbs[name].PEGSection
```

Required arguments

name A String specifying the repository key.

material A String specifying the name of the material.

Optional arguments

thickness A Float specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

wedgeAngle1 A Float specifying the value of the x component of the angle between the bounding planes, $\Delta\phi_x$. The default value is 0.0.

wedgeAngle2 A Float specifying the value of the y component of the angle between the bounding planes, $\Delta\phi_y$. The default value is 0.0.

Return value

A PEGSection object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the PEGSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PEGSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PEGSection object has members with the same names and descriptions as the arguments to the *PEGSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

RebarLayers object

The RebarLayers object defines the rebar properties of a section.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section.rebarLayers
mdb.models[name].sections[name].rebarLayers
import odbSection
session.odbs[name].sections[name].rebarLayers
```

RebarLayers(...)

This method creates a RebarLayers object.

Path

```
mdb.models[name].parts[name].compositeLayups[i].section.RebarLayers
mdb.models[name].sections[name].RebarLayers
session.odbs[name].sections[name].RebarLayers
```

Required arguments

rebarSpacing A SymbolicConstant specifying the type of rebar geometry. Possible values are CONSTANT, ANGULAR, and LIFT_EQUATION.

layerTable A *LayerPropertiesArray* object specifying the layers of reinforcement.

Optional arguments

None.

Return value

A RebarLayers object.

Exceptions

None.

setValues(...)

This method modifies the RebarLayers object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RebarLayers* method.

Return value

`None.`

Exceptions

None.

Members

The RebarLayers object has members with the same names and descriptions as the arguments to the *RebarLayers* method.

Corresponding analysis keywords

- **REBAR LAYER*

SectionLayer object

The SectionLayer object defines the material layer in a composite shell.

Access

```
import section
mdb.models[name].parts[name].compositeLayups[i].section.layup[i]
mdb.models[name].sections[name].layup[i]
import odbSection
session.odbs[name].sections[name].layup[i]
```

SectionLayer(...)

This method creates a SectionLayer object.

Path

```
section.SectionLayer
odbSection.SectionLayer
```

Required arguments

thickness A Float specifying the thickness of the section layer.

material A String specifying the name of the section layer material.

Optional arguments

orientAngle A Float or a String specifying the relative orientation of the section layer. A Float specifies the angular orientation; a String specifies a user-subroutine orientation name. If a String is specified, a user-subroutine orientation is used, otherwise the Float value is used as an angular orientation. The default value is 0.0.

numIntPts An Int specifying the number of integration points to be used through the section. This argument is valid only if the *preIntegrate* argument on the parent *CompositeShellSection* object is set to ON. The default value is 3.

axis A SymbolicConstant specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. This only applies if a valid reference is provided for the orientation. Possible values are AXIS_1, AXIS_2, and AXIS_3. The default value is AXIS_3.

angle A Float specifying the angle of the additional rotation. This only applies if a valid reference is provided for the orientation. The default value is 0.0.

additionalRotationType A SymbolicConstant specifying the method used to describe the additional rotation when a valid orientation is specified. Possible values are ROTATION_NONE, ROTATION_ANGLE, and ROTATION_FIELD. The default value is ROTATION_NONE.

plyName A String specifying the ply identifier for this section layer. The default value is "".

orientation The SymbolicConstant None or a *DatumCsys* object specifying a coordinate system reference for the relative orientation of this layer. If this reference is valid it is used as the relative orientation of the layer, otherwise the *orientAngle* is used as described. The default value is None.

additionalRotationField A String specifying the name of the field specifying the additional rotation. The default value is "".

Return value

A SectionLayer object.

Exceptions

None.

Members

The SectionLayer object has members with the same names and descriptions as the arguments to the *SectionLayer* method.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

ShellSection object

The ShellSection object defines the properties of a shell section. The ShellSection object is derived from the *Section* object. The ShellSection object has no explicit constructor and no methods or members.

The ShellSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

Members

The ShellSection object can have the following members:

name
A String specifying the repository key.

transverseShear
A *TransverseShearShell* object specifying the transverse shear stiffness properties.

SolidSection object

The SolidSection object defines the properties of a solid section. The SolidSection object has no explicit constructor, no members, and no methods. The SolidSection object is an abstract base type.

The SolidSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

Members

The SolidSection object has the following member:

name
A String specifying the repository key.

SurfaceSection object

The SurfaceSection object defines the properties of a surface section.

The SurfaceSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

SurfaceSection(...)

This method creates a SurfaceSection object.

Path

```
mdb.models[name].SurfaceSection
session.odbs[name].SurfaceSection
```

Required arguments

name A String specifying the repository key.

Optional arguments

useDensity A Boolean specifying whether or not to use the value of *density*. The default value is OFF.

density A Float specifying the value of density to apply to this section. The default value is 0.0.

Return value

A SurfaceSection object.

Exceptions

RangeError and InvalidNameError.

Members

The SurfaceSection object has members with the same names and descriptions as the arguments to the *SurfaceSection* method.

In addition, the SurfaceSection object can have the following member:

rebarLayers A *RebarLayers* object specifying reinforcement properties.

Corresponding analysis keywords

- **SURFACE SECTION*

TransverseShearBeam object

The TransverseShearBeam object defines the transverse shear stiffness properties of a beam section.

Access

```
import section
mdb.models[name].sections[name].beamTransverseShear
import odbSection
session.odbs[name].sections[name].beamTransverseShear
```

TransverseShearBeam(...)

This method creates a TransverseShearBeam object.

Path

```
mdb.models[name].sections[name].TransverseShearBeam
session.odbs[name].sections[name].TransverseShearBeam
```

Required arguments

scfDefinition A SymbolicConstant specifying how slenderness compensation factor of the section is given. Possible values are ANALYSIS_DEFAULT, COMPUTED, and VALUE.

Optional arguments

k23 `None` or a Float specifying the k23 shear stiffness of the section. The default value is `None`.

k13 `None` or a Float specifying the k13 shear stiffness of the section. The default value is `None`.

slendernessCompensation The SymbolicConstant COMPUTED or a Float specifying the slenderness compensation factor of the section. The default value is 0.25.

Return value

A TransverseShearBeam object.

Exceptions

None.

setValues(...)

This method modifies the TransverseShearBeam object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TransverseShearBeam* method.

Return value

`None.`

Exceptions

None.

Members

The TransverseShearBeam object has members with the same names and descriptions as the arguments to the *TransverseShearBeam* method.

Corresponding analysis keywords

- **TRANSVERSE SHEAR STIFFNESS*

TransverseShearShell object

The TransverseShearShell object defines the transverse shear stiffness properties of a shell section.

Access

```
import section
mdb.models[name].sections[name].transverseShear
import odbSection
session.odbs[name].sections[name].transverseShear
```

TransverseShearShell(...)

This method creates a TransverseShearShell object.

Path

```
mdb.models[name].sections[name].TransverseShearShell
session.odbs[name].sections[name].TransverseShearShell
```

Required arguments

k11 A Float specifying the shear stiffness of the section in the first direction.

k22 A Float specifying the shear stiffness of the section in the second direction.

k12 A Float specifying the coupling term in the shear stiffness of the section.

Optional arguments

None.

Return value

A TransverseShearShell object.

Exceptions

None.

setValues(...)

This method modifies the TransverseShearShell object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TransverseShearShell* method.

Return value

`None.`

Exceptions

None.

Members

The TransverseShearShell object has members with the same names and descriptions as the arguments to the *TransverseShearShell* method.

Corresponding analysis keywords

- **TRANSVERSE SHEAR STIFFNESS*

TrussSection object

The TrussSection object defines the properties of a truss section.

The TrussSection object is derived from the *Section* object.

Access

```
import section
mdb.models[name].sections[name]
import odbSection
session.odbs[name].sections[name]
```

TrussSection(...)

This method creates a TrussSection object.

Path

```
mdb.models[name].TrussSection
session.odbs[name].TrussSection
```

Required arguments

name A String specifying the repository key.

material A String specifying the name of the material.

Optional arguments

area A Float specifying the cross-sectional area for the section. Possible values are *area*> 0. The default value is 1.0.

Return value

A TrussSection object.

Exceptions

RangeError and InvalidNameError.

setValues(...)

This method modifies the TrussSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TrussSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TrussSection object has members with the same names and descriptions as the arguments to the *TrussSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

Session commands

Session commands are used to create objects that are not stored with the model; for example, viewports and display groups.
Abaqus/CAE retains Session objects only for the duration of the session; they are not saved when the model database is saved.

In this section:

- *Session object*
- *AutoColors object*
- *Color object*
- *Drawing object*
- *Image object*
- *JournalOptions object*
- *MemoryReductionOptions object*
- *NetworkDatabaseConnector object*
- *NumberFormat object*

Session object

The Session object has no constructor. Abaqus creates the *session* member when a session is started.

Access

```
session
```

setValues(...)

This method modifies the Session object.

Required arguments

None.

Optional arguments

kernelMemoryLimit A Float specifying the memory limit value for the Abaqus/CAE kernel process in megabytes. If the limit is exceeded, Abaqus/CAE displays an error message.

The default memory limit value for Windows 32-bit systems if not set is 1800 MB. Increasing the memory limit is not recommended unless you are using a Windows 32-bit system with the boot option `/3GB /userva=SizeInMBytes` to extend the amount of memory available for Abaqus/CAE. In this case the limit can be changed to 2800 MB.

If the kernel memory size reaches the **abq_ker_memory** value or the virtual memory limit of the machine, the following message will be displayed:

```
Operation did not complete due to a memory allocation failure.
```

For optimal performance, the memory limit should be set to a value less than the physical amount of memory on the machine.

The minimum setting allowed is 256 MB.

Return value

`None.`

Exceptions

None.

enableCADConnection(...)

This method enables the Abaqus/CAE listening port for the specified *CAD* system.

Required arguments

CADName A String specifying the CAD system. Available options are Pro/ENGINEER, CATIA V5, CATIA V6, NX, and SolidWorks.

Optional arguments

portNum An Integer specifying the port number to be used by the CAD system to communicate with Abaqus/CAE. If unspecified, attempts will be made to identify an open port. The default ports used are:

Pro/E: 49178

CATIA V5: 49179

SolidWorks: 49180

NX: 49181

CATIA V6: 49182

Return value

The connection port number used for the CAD connection.

Exceptions

None.

`isCADConnectionEnabled()`

This method checks the status of CAD Connection.

Arguments

None.

Return value

A Boolean value of True if the CAD connection enabled and False if the CAD connection disabled.

Exceptions

None.

`disableCADConnection(...)`

This method disables an associative import CAD connection that was enabled.

Required arguments

CADName A String specifying the CAD system for which associative import will be disabled. Available options are Pro/ENGINEER, CATIA V5, and CATIA V6, NX, and SolidWorks.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`enableParameterUpdate(...)`

This method enables parameter updates for ProE and NX by establishing a connection with the listening port previously set up by the CAD application.

Required arguments

CADName
A String specifying the CAD system for which parameter update will be enabled. Available options are Pro/ENGINEER and NX.

CADVersion
A String specifying the CAD system version. Allowable options take the form of the specific CAD system plus a version string. Examples for Pro/ENGINEER are "Wildfire5" and "Creo4." An NX example is "NX11."

Optional arguments

CADPort
An Integer specifying the port number to be used by Abaqus/CAE to communicate with the CAD system. If unspecified, attempts will be made to identify an open port. This port number is not the same as the *portNum* used by the associative import interface. The default CAD listening ports are:

ProE: 3344

NX: 3344

Return value

`None.`

Exceptions

None.

`setCADPortNumber(...)`

This method enables parameter updates for CATIA V5 and CATIA V6 by establishing a connection with the listening port previously set up by the CAD application. This port number is used to send the parameter information to the CAD system.

Required arguments

CADName
A String specifying the CAD system for which the port number will be saved. The available options are 'CATIA V5' and ' CATIA V6'.

Port
An integer specifying the port number to be used by Abaqus/CAE to send the modified parameters to the CAD system.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`updateCADParameters(...)`

This method updates the parameters for the specified model using the specified parameter file.

Required arguments

modelName
A String specifying the model name to update.

CADName
A String specifying the CAD system for which Abaqus updates the parameters. The available options are 'Pro/ENGINEER', 'CATIA V5', and 'CATIA V6.'

parameterFile
A parameter file containing the parameters that were exposed in the CAD system using the 'ABQ_' prefix.

CADPartFile
A file name specifying the CAD part file for which parameter update is triggered.

For *CADName*='CATIA V5' or 'CATIA V6', you can specify either products or parts using this argument. If you specify a product, Abaqus updates all the parts associated with that product.

For *CADName*='Pro/ENGINEER', this argument is optional, and you can specify update for parts only. However, a single file can be associated with multiple parts in the case of family tables. In this case, Abaqus updates all listed parts.

Optional arguments

CADPartName
An String specifying the part name to update. This part name should match the part name in the originating CAD system.

If you specify neither *CADPartFile* nor *CADPartName* during an update in which you specified *CADName*='Pro/ENGINEER,', Abaqus updates all the parts in the specified file.

Return value

`None.`

Exceptions

None.

disableParameterUpdate(...)

This method disables an associative CAD connection using parameters.

Required arguments

CADName A String specifying the CAD system for which the parameter update will be disabled. Available option is Pro/ENGINEER.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

printToFile(...)

This method prints canvas objects to a file using the attributes stored in the *PrintOptions* object and the appropriate format options object.

Required arguments

fileName A String specifying the file to which the image is to be written. If no file extension is supplied, an extension is added based on the selected image format (`.ps`, `.eps`, `.png`, `.tif`, `.svg`, or `.svgz`).

Optional arguments

format A SymbolicConstant specifying the image format. Possible values are PNG, SVG, TIFF, PS, and EPS. The default value is PNG.

canvasObjects A sequence of canvas objects (viewports, text strings, or arrows) to print. The default is to print all canvas objects.

compression A Boolean specifying the format for an SVG file. It is valid to use this argument only when *format* is SVG. Possible values are False (Uncompressed) and True (Compressed).

Return value

`None.`

Exceptions

None.

printToPrinter(...)

This method prints canvas objects to a Windows printer or to a PostScript printer. The attributes used for printing to a Windows printer are stored in the *PrintOptions* object and the *PageSetupOptions* object; the attributes used for printing to a PostScript printer are stored in the *PrintOptions* object and the *PsOptions* object.

Required arguments

None.

Optional arguments

printCommand A String specifying the operating system command or printer name to issue for printing to the printer. The default value is "lpr" or the value specified by the `printOptions` method. If you create a script to print directly to a Windows printer, the *printCommand* must take the following form:

```
session.printToPrinter.setValues(printCommand='PRINTER[
                                  number of characters in name
                                  ]:
                                  printername
                                  PROPERTIES[
                                  number of characters in properties
                                  ]:
                                  document properties
                                  ')
```

The `PROPERTIES` is a list of characters that represents the printing preferences for the selected Windows printer. The properties are not required in a script; the printed output will use the current settings for the selected printer. You can use `'PRINTER[7]: DEFAULT'` to specify the default Windows printer.

numCopies An Int specifying the number of copies to print. Possible values are $1 \leq$ *numCopies* ≤ 100. The default value is 1.

canvasObjects A sequence of canvas objects (viewports, text strings, or arrows) to print. The default is to print all canvas objects.

Return value

`None.`

Exceptions

If *printCommand* is invalid:	SystemError: invalid print command
If the print command fails:	SystemError: print command failed
If *numCopies* is out of range:	RangeError: numCopies must be in the range 1 <= value <= 100

If *compression* is specified when *format* is not SVG: TypeError: keyword error on compression

`saveOptions(...)`

This method saves your customized display settings.

Required arguments

directory
A SymbolicConstant specifying the directory in which Abaqus saves the file that will be used to restore your customized settings (`abaqus_2025.gpr`). Possible values are HOME and CURRENT.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`writeVrmlFile(...)`

This method exports the current viewport objects to a file.

Required arguments

fileName
A String specifying the file to which the graphics data is to be written. If no file extension is supplied, an extension is added based on the selected format (`.wrl`, `.wrz`).

Optional arguments

compression
A Boolean specifying the format of the VRML file. Possible values are OFF (Uncompressed) and ON (Compressed).

canvasObjects
A sequence of canvas objects (viewports, text strings, or arrows) to export.

Return value

`None.`

Exceptions

None.

`write3DXMLFile(...)`

This method exports the current viewport objects to a file.

Required arguments

fileName A String specifying the file to which the graphics data is to be written. If no file extension is supplied, (`.3dxml`) will be added.

Optional arguments

format A Boolean specifying the format. Possible values are False (Uncompressed) and True (Compressed).

canvasObjects A sequence of canvas objects to export.

Return value

`None.`

Exceptions

None.

`writeOBJFile(...)`

This method exports the current viewport objects to a file.

Required arguments

fileName A String specifying the file to which the graphics data is to be written. If no file extension is supplied, (`.obj`) will be added.

Optional arguments

canvasObjects A sequence of canvas objects to export.

Return value

`None.`

Exceptions

None.

Members

The Session object can have the following members:

attachedToGui A Boolean specifying whether an Abaqus interactive session is running.

replayInProgress A Boolean specifying whether Abaqus is executing a replay file.

kernelMemoryFootprint
A Float specifying the memory usage value for the Abaqus/CAE kernel process in megabytes.

kernelMemoryMaxFootprint
A Float specifying the maximum value for the memory usage for the Abaqus/CAE kernel process in megabytes.

kernelMemoryLimit
A Float specifying the limit for the memory use for the Abaqus/CAE kernel process in megabytes.

colors
A repository of *Color* objects.

journalOptions
A *JournalOptions* object specifying how to record selection of geometry in the journal and replay files.

memoryReductionOptions
A *MemoryReductionOptions* object specifying options for running in reduced memory mode.

nodeQuery
A *NodeQuery* object specifying nodes and their coordinates in a path.

sketcherOptions
A *ConstrainedSketcherOptions* object specifying common options for all sketches.

viewerOptions
A *ViewerOptions* object.

animationOptions
An *AnimationOptions* object.

aviOptions
An *AVIOptions* object.

gifOptions
A *GIFOptions* object.

imageAnimationOptions
An *ImageAnimationOptions* object.

imageAnimation
An *ImageAnimation* object.

mp4Options
An *MP4Options* object.

quickTimeOptions
A *QuickTimeOptions* object.

viewports
A repository of *Viewport* objects.

customData
A *RepositorySupport* object.

defaultFieldReportOptions
A *FieldReportOptions* object.

defaultFreeBodyReportOptions
A *FreeBodyReportOptions* object.

fieldReportOptions
A *FieldReportOptions* object.

freeBodyReportOptions A *FreeBodyReportOptions* object.

odbs A repository of *Odb* objects.

scratchOdbs A repository of *ScratchOdb* objects.

defaultOdbDisplay A *DefaultOdbDisplay* object.

defaultPlot A *DefaultPlot* object.

defaultChartOptions A *DefaultChartOptions* object.

odbData A repository of *OdbData* objects.

mdbData A repository of *MdbData* objects.

paths A repository of *Path* objects.

freeBodies A repository of *FreeBody* objects.

streams A repository of *Stream* objects.

spectrums A repository of *Spectrum* objects.

currentProbeValues A *CurrentProbeValues* object.

defaultProbeOptions A *ProbeOptions* object.

probeOptions A *ProbeOptions* object.

probeReport A *ProbeReport* object.

defaultProbeReport A *ProbeReport* object.

selectedProbeValues A *SelectedProbeValues* object.

printOptions A *PrintOptions* object.

epsOptions An *EpsOptions* object.

pageSetupOptions A *PageSetupOptions* object.

pngOptions A *PngOptions* object.

psOptions A *PsOptions* object.

svgOptions A *SvgOptions* object.

tiffOptions
A *TiffOptions* object.

autoColors
An *AutoColors* object specifying the color palette to be used for color coding.

xyColors
An *AutoColors* object specifying the color palette to be used for*XYCurve* objects.

xyDataObjects
A repository of *XYData* objects.

curves
A repository of *XYCurve* objects.

xyPlots
A repository of *XYPlot* objects.

charts
A repository of *Chart* objects.

defaultXYReportOptions
An *XYReportOptions* object.

xyReportOptions
An *XYReportOptions* object.

views
A repository of *View* objects.

networkDatabaseConnectors
A repository of *NetworkDatabaseConnector* objects.

displayGroups
A repository of *DisplayGroup* objects.

graphicsInfo
A *GraphicsInfo* object.

defaultGraphicsOptions
A *GraphicsOptions* object.

graphicsOptions
A *GraphicsOptions* object.

defaultViewportAnnotationOptions
A *ViewportAnnotationOptions* object.

queues
A repository of *Queue* objects.

currentViewportName
A String specifying the name of the current viewport.

sessionState
A Dictionary object specifying the viewports and their associated models. The Dictionary key specifies the viewport name. The Dictionary value is a Dictionary specifying the model name.

images
A repository of *Image* objects.

movies
A repository of *Movie* objects.

defaultLightOptions
A *LightOptions* object.

drawingArea

A *DrawingArea* object.

defaultMesherOptions

A *MesherOptions* object specifying how to control default settings in the Mesh module.

drawings

A repository of *Drawing* objects.

AutoColors object

The AutoColors object defines a color palette.

Access

```
session.autoColors
```

`setValues(...)`

This method changes the color palette.

Required arguments

colors

A sequence of Strings specifying the colors of the palette. Strings must be named colors or represent red, green, blue components in hexadecimal form. For instance, '#FF0000' for red, '#00FF00' for green and '#0000FF' for blue

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The AutoColors object has members with the same names and descriptions as the arguments to the *`setValues`* method.

Color object

The Color object contains the RGB definition of a system color.

Access

```
session.colors[name]
```

setByRGB(...)

This method changes the RGB value of a user-defined color. However, users cannot define colors, and this method does not modify system-defined colors.

Required arguments

rgb
A sequence of three Floats specifying the RGB value of the color. The Float values must be between 0.0 and 1.0.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The Color object has the following members:

name
A String specifying the name of the color.

rgb
A tuple of three Floats specifying the RGB value of the color. The Float values must be between 0.0 and 1.0.

Drawing object

A drawing is the container for a geometric object. The Drawing object stores the vertex data and various settings that determine how the drawing will be displayed.

Access

```
session.drawings[name]
```

Drawing(...)

This method creates an empty Drawing object.

Path

```
session.Drawing
```

Required arguments

name	A String specifying the repository key.

Optional arguments

None.

Return value

A Drawing object.

Exceptions

ValueError.

If the user attempts to create a new drawing with the name of an existing drawing:	ValueError: There is already a drawing with this name

`setVertices(...)`

This method accepts the vertex data that defines the Drawing object. It defines in an array of vertices with a length equal to the length of the *vertexData* sequence divided by *vertexDimension*.

Required arguments

vertexDimension	An Integer in the range of 2 to 4 specifying how many Float values are needed to compose a single vertex.
vertexData	A sequence of Float values that will be used to compose the vertices. There must be *vertexDimension* values in the sequence for each vertex.

Optional arguments

None.

Return value

The number of vertices described.

Exceptions

RangeError.

ValueError.

If an invalid *vertexDimension* is specified:	RangeError: *vertexDimension* must be in the range 2 <= value <= 4
If *vertexData* is an empty sequence:	ValueError: *vertexData* cannot be empty.
If `setVertices` has already been called and this call is sending fewer vertices:	ValueError: *vertexData* cannot be reduced

`setNormals(...)`

This method accepts the normal data for each vertex. It defines in an array of normal vectors with a length equal to the length of the *normalData* sequence divided by 3.

Required arguments

normalData

A sequence of Float values that will be used to compose the normals. There must be 3 values in the sequence for each normal.

If only one normal is specified, all vertices will use the normal value.

Optional arguments

None.

Return value

The number of normals described.

Exceptions

RangeError.

ValueError.

If *normalData* is sequence with less than 3 values:	ValueError: *normalData* must have at least three values.
If `setNormals` has already been called and this call is sending fewer values:	ValueError: *normalData* cannot be reduced.

`setColors(...)`

This method accepts the color data for each vertex. It defines in an array of colors with a length equal to the length of the *colorData* sequence divided by *colorDimension*.

Required arguments

colorDimension An Integer in the range of 3 to 4 specifying how many Float values are needed to compose a single color.

colorData A sequence of Float values in the range of 0.0 to 1.0 that will be used to compose the colors. There must be *colorDimension* values in the sequence for each color. The first color will be associated with the fist vertex and so on.

The first Float will be the red value for the first color. The second Float will be the green value and the third will be the blue value. When *colorDimension* is 4 the 4th Float will be the alpha value for the first color but is ignored.

If only one color is specified, all vertices will use the color value.

Optional arguments

None.

Return value

The number of colors described.

Exceptions

RangeError.

ValueError.

If an invalid *colorDimension* is specified:	RangeError: *colorDimension* must be in the range 3 <= value <= 4
If *colorData* is an empty sequence:	ValueError: *colorData* cannot be empty.
If `setColors` has already been called and this call is sending fewer colors:	ValueError: *colorData* cannot be reduced

`setEdgeColor(...)`

This method allows a separate, single color to be used when rendering the edges of the drawing. Once called, edges will be rendered using the specified color but facets will continue to use the colors specified in the `setColors` method. An empty sequence can be specified to resume using the colors arrays for edges.

Required arguments

edgeColor A sequence of 0 or 3 Float values in the range of 0.0 to 1.0 that will be used to compose the edge color.

If the initial Float value is -1, the viewport background color will be used for the edge color.

Optional arguments

None.

Return value

`None.`

Exceptions

ValueError.

If *edgeColor* is not a sequence of 0 or 3 Floats: ValueError: *edgeColor* must be a tuple with 3 values

`setPointColor(...)`

This method allows a separate, single color to be used when rendering the points of the drawing. Once called, points will be rendered using the specified color but facets will continue to use the colors specified in the `setColors` method. An empty sequence can be specified to resume using the colors arrays for points.

Required arguments

pointColor A sequence of 0 or 3 Float values in the range of 0.0 to 1.0 that will be used to compose the point color.

If the initial Float value is -1, the viewport background color will be used for the edge color.

Optional arguments

None.

Return value

`None.`

Exceptions

ValueError.

If *pointColor* is a not sequence of 0 or 3 Floats: ValueError: *pointColor* must be a tuple with 3 values

`addArrayDraw(...)`

This method adds a rendering command to the drawing and can be called multiple times to add additional rendering commands. When the drawing is referenced by a *Viewport*, the drawing commands are used the render the Drawing.

The rendering command constructs the specified type of geometric primitive using *numVertices* array elements starting at element index *startIndex*.

Required arguments

type	A SymbolicConstant specifying the type of graphics primitive rendered by this command. Possible values are POINTS, LINES, LINE_LOOP, LINE_STRIP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUADS, and QUAD_STRIP.
startIndex	An Integer specifying the index of the first vertex to render.
numVertices	An Integer specifying the total number of vertices to render.

Optional arguments

polygonMode	A SymbolicConstant specifying how polygonal graphics primitives will be rendered by this command. Possible values are FILL, EDGES, and POINTS. The default value is FILL.

Return value

The total number of rendering commands that have been specified.

Exceptions

ValueError.

If (*startIndex* + *numVertices* - 1) is larger than the length of the vertex array:	ValueError: Drawing request extends past array size of vertices.
If (*startIndex* + *numVertices* - 1) is larger than the length of the normal array and normals are required for the graphics primitive:	Drawing request extends past array size of normals.
If (*startIndex* + *numVertices* - 1) is larger than the length of the color array and vertex colors are required for the graphics primitive:	Drawing request extends past array size of colors.

`addIndexDraw(...)`

This method adds a rendering command to the drawing and can be called multiple times to add additional rendering commands. When the drawing is referenced by a *Viewport*, the drawing commands are used the render the Drawing.

The rendering command constructs the specified type of geometric primitive using *numVertices* array elements starting at element index *startIndex*.

Required arguments

type	A SymbolicConstant specifying the type of graphics primitive rendered by this command. Possible values are POINTS, LINES, LINE_LOOP, LINE_STRIP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUADS, and QUAD_STRIP.
indices	A sequence of Integer values specifying index of each vertex to render.

Optional arguments

polygonMode A SymbolicConstant specifying how polygonal graphics primitives will be rendered by this command. Possible values are FILL, EDGES, and POINTS. The default value is FILL.

Return value

The total number of rendering commands that have been specified.

Exceptions

ValueError.

If any value in the *indices* sequence negative:	ValueError: Index values must be positive.
If any value in the *indices* sequence is larger than the length of the vertex array:	ValueError: Drawing request extends past array size of vertices.
If any value in the *indices* sequence is larger than the length of the normal array and normals are required for the graphics primitive:	Drawing request extends past array size of normals.
If any value in the *indices* sequence is larger than the length of the color array and vertex colors are required for the graphics primitive:	Drawing request extends past array size of colors.

`setValues(...)`

This method modifies the rendering of the Drawing object.

Required arguments

None.

Optional arguments

show A Boolean specifying whether the drawing object will be rendered when referenced. The default value is OFF.

cullBackfaces A Boolean specifying whether polygonal graphics primitives facing away from the viewer should be culled (not rendered). The default value is OFF.

The winding order, and not the normal, of the graphics primitive is used to determine its facing.

frontFaceOrder A SymbolicConstant specifying the winding order for polygonal graphics primitives that face the viewer. Possible values are:

- CCW, specifying front face winding order is counter-clockwise.
- CW, specifying front face winding order is clockwise.

The default value is CCW.

smoothShade

A Boolean specifying whether the lighting of polygonal graphics primitives is consistent for each facet or calculated for each displayed pixel. The default value is ON.

When False, only the last normal for each facet will be used in the lighting calculation.

edgesInShaded

A Boolean specifying whether edge and point drawing commands will be issued in a FILLED or SHADED display. The default value is ON.

If no edge or point drawing commands have been defined, the polygonal drawing commands will be issued in WIREFRAME and HIDDEN_LINE displays with the *polygonMode* set to EDGES. If only edge and point drawing commands have been defined, the Drawing will not be rendered in FILLED or SHADED displays.

translucency

A Float specifying the opacity for polygonal graphics primitives. Possible values are $0.0 \leq$ *translucency* ≤ 1.0 with 0.0 being completely transparent (invisible) and 1.0 being opaque. The default value is 1.0.

A value greater than 0.3 will cause the translucent facets to be sorted by depth before being rendered and has the side effect of disabling two-sided lighting for those facets.

lineSize

A Float specifying the width of the line, in millimeters, used to render edges. Possible values are $0.0 \leq$ *lineSize* ≤ 5.0 with 0.0 being interpreted as the thinnest possible line. The default value is 0.0.

A value of 0.0 will be one pixel on the output device. One pixel on the screen is generally visible but one pixel on a 1200 DPI printer may not be clear.

pointSize

A Float specifying the width of points, in millimeters, used to render points. Possible values are $0.0 \leq$ *lineSize* ≤ 5.0 with 0.0 being interpreted as the smallest possible point. The default value is 0.0.

A value of 0.0 will be one pixel on the output device. One pixel on the screen is generally visible but one pixel on a 1200 DPI printer may not be clear.

depthTest

A Boolean specifying whether an object's depth will be considered during rendering. The default value is ON.

When OFF, the object will not be hidden by other objects in front of it.

Return value

`None.`

Exceptions

RangeError.

If an invalid *translucency* value is specified:	RangeError: *translucency* must be in the range 0.0 <= value <= 1.0
If an invalid *lineSize* value is specified:	RangeError: *lineSize* must be in the range 0.0 <= value <= 5.0
If an invalid *pointSize* value is specified:	RangeError: *pointSize* must be in the range 0.0 <= value <= 5.0

Members

The Drawing object has members with the same names and descriptions as the arguments to the *Drawing* method.

In addition, the Drawing object has the following members:

vertexCount

An Int specifying the number of vertices in the vertex array after a call to the `setVertices` method. The default value is 0.

normalCount

An Int specifying the number of normal vectors in the normal array after a call to the `setNormals` method. The default value is 0.

colorCount

An Int specifying the number of colors in the color array after a call to the `setColors` method. The default value is 0.

show

A Boolean specifying whether the drawing object will be rendered when referenced. The default value is OFF.

cullBackfaces

A Boolean specifying whether polygonal graphics primitives facing away from the viewer should be culled (not rendered). The default value is OFF.

The winding order, and not the normal, of the graphics primitive is used to determine its facing.

frontFaceOrder

A SymbolicConstant specifying the winding order for polygonal graphics primitives that face the viewer. Possible values are:

- CCW, specifying front face winding order is counter-clockwise.
- CW, specifying front face winding order is clockwise.

The default value is CCW.

translucency

A Float specifying the opacity for polygonal graphics primitives. Possible values are $0.0 \le$ *translucency* ≤ 1.0 with 0.0 being completely transparent (invisible) and 1.0 being opaque. The default value is 1.0.

A value greater than 0.3 will cause the translucent facets to be sorted by depth before being rendered and has the side effect of disabling two-sided lighting for those facets.

lineSize

A Float specifying the width of the line, in millimeters, used to render edges. Possible values are $0.0 \le$ *lineSize* ≤ 5.0 with 0.0 being interpreted as the thinnest possible line. The default value is 0.0.

A value of 0.0 will be one pixel on the output device. One pixel on the screen is generally visible but one pixel on a 1200 DPI printer may not be clear.

pointSize

A Float specifying the width of points, in millimeters, used to render points. Possible values are $0.0 \le$ *lineSize* ≤ 5.0 with 0.0 being interpreted as the smallest possible point. The default value is 0.0.

A value of 0.0 will be one pixel on the output device. One pixel on the screen is generally visible but one pixel on a 1200 DPI printer may not be clear.

smoothShade

A Boolean specifying whether the lighting of polygonal graphics primitives is consistent for each facet or calculated for each displayed pixel. The default value is ON.

When False, only the last normal for each facet will be used in the lighting calculation.

edgesInShaded

A Boolean specifying whether edge and point drawing commands will be issued in a FILLED or SHADED display. The default value is ON.

If no edge or point drawing commands have been defined, the polygonal drawing commands will be issued in WIREFRAME and HIDDEN_LINE displays with the *polygonMode* set to EDGES. If only edge and point drawing commands have been defined, the Drawing will not be rendered in FILLED or SHADED displays.

edgeColor

A tuple of three Floats specifying the *Red*, *Green*, and *Blue* component values for the edge color. Possible values for each component are between 0.0 and 1.0.

pointColor

A tuple of three Floats specifying the *Red*, *Green*, and *Blue* component values for the point color. Possible values for each component are between 0.0 and 1.0.

depthTest

A Boolean specifying whether an object's depth will be considered during rendering. The default value is ON.

When OFF, the object will not be hidden by other objects in front of it.

Image object

The Image object is used to store color values and attributes associated with a raster file. Upon creation, the Image object is added to the session.images repository.

Access

```
session.images[name]
```

Image(...)

This method creates an Image object from the contents of the specified file.

Path

```
session.Image
```

Required arguments

name A String specifying the repository name for the image.

fileName A String specifying the file from which the image is to be read. The file extension must be specified and indicate the image format (`.bmp`, `.gif`, `.jpg`, `.jpeg`, `.ico`, `.pcx`, `.png`, `.rgb`, `.tga`, `.tif`, or `.xpm`).

Optional arguments

None.

Return value

An Image object.

Exceptions

ValueError.

If *fileName* does not exist or can not be read: ValueError: Unable to open image file

If *fileName* references an unsupported image file format: ValueError: Unsupported image format

If the contents of *fileName* are corrupt or can not be decoded: ValueError: Unable to decode image file

`ImageFromMovie(...)`

This method creates an Image object from a given frame of an existing *Movie* object.

Required arguments

name A String specifying the repository name for the image.

movieName	A String specifying the name of the movie from which the image is to be extracted. The movie must exist in the session.movies repository.
frame	An Int specifying the movie frame number defining the image to be extracted.
time	A Float specifying the time of the movie defining the image to be extracted.

Optional arguments

None.

Return value

An Image object.

Exceptions

ValueError.

TypeError.

If *movieName* does not exist:	ValueError: There is no movie object with this name: 'movieName'
If *frame* references an non existing frame:	ValueError: Could not load frame n from movie: 'movieName'
If *time* references an non existing frame:	ValueError: Could not load frame at time 't' from movie: 'movieName'
If *time* and *frame* are given in the same command:	TypeError: keyword error on time

Members

The Image object has members with the same names and descriptions as the arguments to the *Image* method.

JournalOptions object

A JournalOptions object specifies how to record selection of geometry in the journal and replay files. *journalOptions* can also be used to set the numeric formatting options for field report output, geometry commands output, and a default format for other numeric output. The JournalOptions object has no constructor. Abaqus creates the *journalOptions* member when a session is started.

Access

```
session.journalOptions
```

`setValues(...)`

This method modifies the JournalOptions object.

Required arguments

None.

Optional arguments

replayGeometry
A SymbolicConstant specifying the format of the geometry in the replay file. Possible values are COORDINATE, INDEX, and COMPRESSEDINDEX. The default value is COMPRESSEDINDEX.

recoverGeometry
A SymbolicConstant specifying the format of the geometry in the recovery file. Possible values are COORDINATE, INDEX, and COMPRESSEDINDEX. The default value is COMPRESSEDINDEX.

defaultFormat
A *NumberFormat* object specifying the default format for numeric output. The default values are the same as the default values for the *NumberFormat* object.

fieldReportFormat
A *NumberFormat* object specifying the default format for numbers in a field report output. The default values are the same as the default values for the *NumberFormat* object.

geometryFormat
A *NumberFormat* object specifying the default format for numbers in geometry commands output. The default values are the same as the default values for the *NumberFormat* object.

Return value

```
None.
```

Exceptions

None.

Members

The JournalOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

MemoryReductionOptions object

The MemoryReductionOptions object controls the default settings that Abaqus/CAE uses for running in reduced memory mode. The MemoryReductionOptions object has no constructor. Abaqus creates the *MemoryReductionOptions* member when a session is started.

Access

```
session.memoryReductionOptions
```

setValues(...)

This method modifies the MemoryReductionOptions object.

Required arguments

None.

Optional arguments

reducedMemoryMode
A Boolean specifying whether Abaqus/CAE should run in reduced memory mode. The default value is ON.

percentThreshold
A Float specifying the percent of *kernelMemoryLimit* at which the reduced memory mode starts. The default value is 75.0.

Return value

```
None.
```

Exceptions

None.

Members

The MemoryReductionOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

NetworkDatabaseConnector object

The NetworkDatabaseConnector object allows you to access an output database on a remote system.

Access

```
session.networkDatabaseConnectors[name]
```

NetworkDatabaseConnector(...)

This method creates a NetworkDatabaseConnector object that you can use to access a remote output database. You can create a network database connector from any platform: Windows or Linux. However, the network database connector server must reside on a Linux platform; you cannot access an output database that resides on a remote Windows system. You can access only a remote output database; you cannot access a remote model database.

Path

```
session.NetworkDatabaseConnector
```

Required arguments

name A String specifying the repository key.

hostName A String specifying the name of the remote computer.

directory A String specifying the directory on the remote computer.

Optional arguments

remoteAbaqusDriverName A String specifying the name of command to execute Abaqus/CAE on the remote computer.

remoteLoginMechanism A SymbolicConstant specifying the remote shell command on the local system. Possible values are RSH and SSH. The default value is SSH.

sshPath A String specifying the path to the`ssh` command on the local system. The default value is an empty string.

serverPort An Int specifying the server port on the remote computer. If *serverPort* =0, the host and remote systems are allowed to establish their own port numbers. The default value is 0.

connectionPort An Int specifying the connection port on the remote computer. The default value is 0.

serverTimeout An Int specifying the timeout in seconds for the remote server. For example: 86400 corresponds to one day. The server exits if it does not receive any communication from the client during the time specified. The default value is 86400.

allowAutomaticStartup A Boolean specifying whether to start the remote network database connector server. The default value is ON.

Return value

A NetworkDatabaseConnector object.

Exceptions

None.

start(...)

This method starts the remote network database connector server on the remote host.

Required arguments

None.

Optional arguments

serverPort An Int specifying the server port on the remote computer. If *serverPort* =0, the host and remote systems are allowed to establish their own port numbers. The default value is 0.

serverTimeout An Int specifying the timeout in seconds for the remote server. For example: 86400 corresponds to one day. The server exits if it does not receive any communication from the client during the time specified. The default value is 86400.

Return value

`None.`

Exceptions

None.

stop()

This method stops the remote network database connector server on the remote host.

Arguments

None.

Return value

`None.`

Exceptions

None.

`setValues(...)`

This method modifies the NetworkDatabaseConnector object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *NetworkDatabaseConnector* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The NetworkDatabaseConnector object has members with the same names and descriptions as the arguments to the *NetworkDatabaseConnector* method.

In addition, the NetworkDatabaseConnector object has the following member:

connected A Boolean specifying if the connection between the client and the server is established.

NumberFormat object

The NumberFormat object is a formatting template used to define formatting options for certain numeric output.

Access

```
import visualization
session.defaultFieldReportOptions.numberFormat
session.fieldReportOptions.numberFormat
session.journalOptions.defaultFormat
session.journalOptions.fieldReportFormat
session.journalOptions.geometryFormat
```

NumberFormat(...)

This method creates a NumberFormat object.

Path

```
session.defaultFieldReportOptions.NumberFormat
session.fieldReportOptions.NumberFormat
session.journalOptions.NumberFormat
```

Required arguments

None.

Optional arguments

blankPad

A Boolean specifying whether the printed digits should be padded with blank characters to ensure equal sized fields. The *blankPad* argument is useful when your printed output includes columns. The default value is ON.

format

A SymbolicConstant specifying the formatting type. Possible values are ENGINEERING, SCIENTIFIC, and AUTOMATIC. The default value is ENGINEERING.

numDigits

An Int specifying the number of digits to be displayed in the result. *numDigits* > 0. The default value is 6.

precision

An Int specifying the number of decimal places to which the number is to be truncated for display. *precision* ≤ 0. If *precision* =0, no truncation is applied. The default value is 0.

Return value

A NumberFormat object.

Exceptions

None.

Members

The NumberFormat object has members with the same names and descriptions as the arguments to the *NumberFormat* method.

Sketcher commands

Sketcher commands are used to define the entities, such as the geometry, constraints, and dimensions, to create a sketch and to store the values and attributes associated with a particular sketch.

In this section:

- *ConstrainedSketch object*
- *ConstrainedSketchConstraint object*
- *ConstrainedSketchDimension object*
- *ConstrainedSketcherOptions object*
- *ConstrainedSketchGeometry object*
- *ConstrainedSketchGeometryArray object*
- *ConstrainedSketchImageOptions object*
- *ConstrainedSketchOptions object*
- *ConstrainedSketchParameter object*
- *ConstrainedSketchVertex object*
- *ConstrainedSketchVertexArray object*

ConstrainedSketch object

A ConstrainedSketch object contains the entities that are used to create a sketch. The objects include *ConstrainedSketchGeometry* objects contained in the Geometry Repository, such as Line, Arc, and Spline. Vertex, Dimension, Constraint, and Parameter objects are contained in their respective repositories.

Access

```
import sketch
mdb.models[name].sketches[name]
```

ConstrainedSketch(...)

This method creates a ConstrainedSketch object. If the sketch cannot be created, the method returns `None`.

Path

```
mdb.models[name].ConstrainedSketch
```

Required arguments

name
A String specifying the repository key.

sheetSize
A Float specifying the sheet size.

Optional arguments

gridSpacing
A Float specifying the spacing between gridlines. Possible values are Floats > 0. The default value is approximately 2 percent of *sheetSize*.

transform
A sequence of sequences of Floats specifying the three-dimensional orientation of the sketch. The sequence is a 3 × 4 transformation matrix specifying the axis of rotation and the translation vector. Possible values are any Floats.

The default value for the axis of rotation is the identity matrix

```
(1.0, 0.0, 0.0),  (0.0, 1.0, 0.0),  (0.0, 0.0, 1.0)
```

The default value for the translation vector is

```
(0.0, 0.0, 0.0)
```

The default values position the sketch on the *X*–*Y* plane centered at the origin.

Return value

A ConstrainedSketch object.

Exceptions

None.

ConstrainedSketch(...)

This method copies one ConstrainedSketch object to a new ConstrainedSketch object.



Note: If the name of the sketch to be copied to is `__edit__`, Abaqus creates an exact copy that contains both reference geometry and a nonidentity transform matrix. Otherwise, the Sketch copy constructor strips the reference geometry from the copied sketch and sets the transform matrix to identity, creating a stand-alone copy.

Path

```
mdb.models[name].ConstrainedSketch
```

Required arguments

name	A String specifying the repository key.
objectToCopy	A ConstrainedSketch object to be copied.

Optional arguments

None.

Return value

A ConstrainedSketch object.

Exceptions

InvalidNameError.

ConstrainedSketchFromGeometryFile(...)

This method creates a ConstrainedSketch object and places it in the sketches repository.

Path

```
mdb.models[name].ConstrainedSketchFromGeometryFile
```

Required arguments

name	A String specifying the repository key.
geometryFile	An *AcisFile* object specifying a file containing geometry. The geometry in the file is converted to two-dimensional sketch geometry in the *X*–*Y* plane.

Optional arguments

None.

Return value

A ConstrainedSketch object.

Exceptions

InvalidNameError.

print()

This method prints the following statistics about a sketch:

- The sketch Id (a positive integer).
- The number of geometry curves (the number of *ConstrainedSketchGeometry* objects).
- The number of dimensions (the number of *ConstrainedSketchDimension* objects).
- The number of vertices (the number of *ConstrainedSketchVertex* objects).

Arguments

None.

Return value

`None.`

Exceptions

None.

assignCenterline(...)

This method indicates the construction line that will be used as a centerline for revolved features.

Required arguments

line
A *ConstrainedSketchGeometry* object specifying a construction line that indicates the centerline of revolved features.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignCenterOfTwist(...)

This method indicates the isolated point that will be used as the center of twist when an extruded feature is created with twist.

Required arguments

point A *ConstrainedSketchVertex* object specifying an isolated point that indicates the center of twist for extruded features that use a twist angle.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`autoDimension(...)`

This method applies dimensions to the selected *ConstrainedSketchGeometry* objects in an effort to make the ConstrainedSketch well defined.

Required arguments

objectList A sequence specifying the *ConstrainedSketchGeometry* objects to dimension.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`autoTrimCurve(...)`

This method automatically trims a selected *ConstrainedSketchGeometry* object at the specified location. If the object does not intersect other *ConstrainedSketchGeometry* objects, the entire selected object will be deleted.

Required arguments

curve1 The *ConstrainedSketchGeometry* object to be trimmed.

point1 A pair of Floats specifying the location on *ConstrainedSketchGeometry* where the trimming should be applied. *point1* and *parameter1* are mutually exclusive.

parameter1 A Float specifying the parameter location on the *ConstrainedSketchGeometry* where the trimming should be applied. *point1* and *parameter1* are mutually exclusive.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`breakCurve(...)`

This method breaks a specified *ConstrainedSketchGeometry* object (*curve1*) using another specified *ConstrainedSketchGeometry* object (*curve2*). If the selected *ConstrainedSketchGeometry* objects intersect, then only *curve1* will be broken; *curve2* is not affected by the operation. The location for the break is determined by the specified point values.

Required arguments

curve1	A *ConstrainedSketchGeometry* object specifying the object to be broken.
point1	A pair of Floats specifying the location on *curve1* near where the break should be applied.
curve2	A *ConstrainedSketchGeometry* object specifying where *curve1* should be broken.
point2	A pair of Floats specifying the location on *curve2* near where *curve1* should be broken.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`copyMirror(...)`

This method creates copies of the given *ConstrainedSketchGeometry* objects, mirrors them about a selected line, and inserts them into the appropriate repositories of the ConstrainedSketch object.

Required arguments

mirrorLine	A *ConstrainedSketchGeometry* object specifying the line about which Abaqus will mirror the sketch.
objectList	A sequence of *ConstrainedSketchGeometry* objects specifying the sketch to be copied and mirrored.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`copyMove(...)`

This method creates copies of the given *ConstrainedSketchGeometry* objects, moves them from their original position, and inserts them into the appropriate repositories of the ConstrainedSketch object.

Required arguments

vector	A sequence of two Floats specifying the translation vector.
objectList	A sequence of *ConstrainedSketchGeometry* objects to be copied and moved.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`copyRotate(...)`

This method creates copies of the given *ConstrainedSketchGeometry* objects, rotates them, and inserts them into the appropriate repositories of the ConstrainedSketch object.

Required arguments

centerPoint	A pair of Floats specifying the center of rotation.
angle	A Float specifying the angle of rotation in degrees.
objectList	A sequence of *ConstrainedSketchGeometry* objects to be copied and moved.

Optional arguments

None.

Return value
None.

Exceptions
None.

copyScale(...)

This method creates copies of the given *ConstrainedSketchGeometry* objects, scales them by the specified value about a selected point, and inserts them into the appropriate repositories of the ConstrainedSketch object.

Required arguments

scaleValue	A Float specifying the value for scaling.
scaleCenter	A pair of Floats specifying the center of scaling.
objectList	A sequence of *ConstrainedSketchGeometry* objects to be copied and scaled.

Optional arguments
None.

Return value
None.

Exceptions
None.

delete(...)

This method deletes the given *ConstrainedSketchGeometry*, *ConstrainedSketchDimension*, or *ConstrainedSketchConstraint* objects.

Required arguments

objectList	A sequence of *ConstrainedSketchGeometry*, *ConstrainedSketchDimension*, or *ConstrainedSketchConstraint* objects to be deleted.

Optional arguments
None.

Return value
None.

Exceptions
None.

deleteParameter(...)

The command deletes a specified parameter.

Required arguments

name A String specifying the name of the parameter to delete.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

dragEntity(...)

This method drags a specified *ConstrainedSketchGeometry* or *ConstrainedSketchVertex* object to a specific location.

Required arguments

entity A *ConstrainedSketchGeometry* or *ConstrainedSketchVertex* object specifying the object to drag.

points A sequence of sequences of three Floats specifying a sequence of points along which to drag the entity. The order of points in the sequence defines a path that determines the solution.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

linearPattern(...)

This method copies *ConstrainedSketchGeometry* objects in a linear pattern along one or two directions. This method also copies any associated dimension or constraint objects that exist between the given objects.

Required arguments

number1
An Integer specifying the total number of copies, including the original objects, that appear along the first direction in the pattern. Possible values are $1 \leq$ *number1* ≤ 1000.

spacing1
A Float specifying the spacing between copies along the first direction in the pattern. Possible values are $0.0 \leq$ *spacing1* .

angle1
A Float specifying the angle in degrees of the first direction in the pattern. Possible values are $-360.0 \leq$ *angle1* ≤ 360.0.

Optional arguments

vertexList
A sequence of *ConstrainedSketchVertex* objects to copy.

geomList
A sequence of *ConstrainedSketchGeometry* objects to copy.

number2
An integer specifying the total number of copies, including the original objects, that appear along the second direction in the pattern. Possible values are $1 \leq$ *number2* ≤ 1000. The default value is 1. The value of either *number1* or *number2* must be greater than one.

spacing2
A Float specifying the spacing between copies along the first direction in the pattern. Possible values are $0.0 \leq$ *spacing2*. The default value is *spacing1*.

angle2
A Float specifying the angle in degrees of the first direction in the pattern. Possible values are $-360.0 \leq$ *angle2* ≤ 360.0. The default value is 90° beyond the value of *angle1*.

Return value
`None.`

Exceptions

AbaqusException Number must be greater than 1 for at least one direction.

`mergeVertices(...)`

This method merges the *ConstrainedSketchVertex* objects that lie within the specified distance of each other. If only one *ConstrainedSketchVertex* object is selected, it will merge all *ConstrainedSketchVertex* objects that lie within the specified distance of that vertex. If more than one vertex is selected, the search will be restricted to only the selected *ConstrainedSketchVertex* objects.

Required arguments

value
A Float specifying the search radius.

vertexList
A sequence of *ConstrainedSketchVertex* objects to be merged.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

move(...)

This method translates the given *ConstrainedSketchGeometry* objects by the given vector.

Required arguments

vector	A sequence of two Floats specifying the translation vector.
objectList	A sequence of *ConstrainedSketchGeometry* objects specifying the objects to be translated.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

offset(...)

This method creates copies of the selected *ConstrainedSketchGeometry* objects, offsets them by the specified distance in the specified direction, and inserts them into the ConstrainedSketch object's appropriate repositories. If connected objects are selected, trim or extend is carried out to complete the offset.

Required arguments

distance	A Float specifying the distance to be offset.
objectList	A sequence of *ConstrainedSketchGeometry* objects to be copied and offset.
side	A SymbolicConstant specifying which side the offset should occur. Possible values are LEFT and RIGHT.

Optional arguments

filletCorners A Boolean specifying whether the corners need to be rounded instead of being extended.

Return value

`None.`

Exceptions

None.

`radialPattern(...)`

This method copies *ConstrainedSketchGeometry* objects in a radial pattern about a specified center point.

Required arguments

number An Int specifying the total number of copies, including the original objects, that appear in the radial pattern. Possible values are $2 \leq$ *number2* ≤ 1000.

totalAngle A Float specifying the total angle in degrees between the first and last instance in the pattern. A positive angle corresponds to a counter-clockwise direction. The values 360° and -360° represent a special case where the pattern makes a full circle. In this case, because the copy would overlay the original, the copy is not placed at the last position. Possible values are $-360.0 \leq$ *totalAngle* ≤ 360.0.

centerPoint A pair of Floats specifying the center of the radial pattern.

Optional arguments

vertexList A sequence of *ConstrainedSketchVertex* objects to copy.

geomList A sequence of *ConstrainedSketchGeometry* objects to copy.

Return value

`None.`

Exceptions

None.

`resetView()`

This method resets the view to be perpendicular to the sketching plane.

Arguments

None.

Return value
`None.`

Exceptions
None.

rectangle(...)

This method creates four lines that form a rectangle with diagonal corners defined by the given points and inserts them into the geometry repository of the ConstrainedSketch object.

Required arguments

point1	A pair of Floats specifying the first corner of the rectangle.
point2	A pair of Floats specifying the second corner of the rectangle.

Optional arguments
None.

Return value
A list of *ConstrainedSketchGeometry* objects.

Exceptions
None.

removeGapsAndOverlaps(...)

This method removes gaps and overlaps between sketch geometries specified by the user. This method is particularly useful when cleaning up imported sketches

Required arguments

tolerance	A float value that specifies the largest size of the gap or overlap between entities that is to be removed. Typically this value is small and is used to close gaps and overlaps that may not exist in the originating program but exist in the sketch because of mismatched tolerances between the two programs.
geomList	A sequence of *ConstrainedSketchGeometry* objects where the gaps and overlaps are to be removed.

Optional arguments
None.

Return value
`None.`

Exceptions

None.

repairShortEdges(...)

This method deletes the short edges specified, optionally selecting only those short edges whose lengths are smaller than the specified tolerance and healing the resultant gap in the sketch. This method is particularly useful in conjunction with `removeGapsAndOverlap` when cleaning up imported sketches.

Required arguments

geomList A sequence of *ConstrainedSketchGeometry* objects where the short edges are to be removed.

Optional arguments

tolerance A float value that is used to select and delete only those edges specified in *geomList* whose lengths are smaller than the given value. The default value is –1.0. This value implies that all edges specified in *geomList* will be removed and the sketch healed to remove gaps left by their removal.

Return value

`None.`

Exceptions

None.

retrieveSketch(...)

This method copies all *ConstrainedSketchGeometry*, *ConstrainedSketchDimension*, *ConstrainedSketchConstraint*, and *ConstrainedSketchParameter* objects from the specified ConstrainedSketch object. The new objects are added to the existing objects (if any). The objects in the specified ConstrainedSketch object are not modified by the retrieve operation.

Required arguments

sketch A ConstrainedSketch object specifying the object from which to copy.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

rotate(...)

This method rotates the given *ConstrainedSketchGeometry* objects by the given angle and about the given point.

Required arguments

centerPoint	A pair of Floats specifying the center of rotation.
angle	A Float specifying the angle of rotation in degrees.
objectList	A sequence of *ConstrainedSketchGeometry* specifying the objects to be rotated.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

scale(...)

This method scales the given *ConstrainedSketchGeometry* objects by the given scale factor and about the given point.

Required arguments

scaleValue	A Float specifying the value of scale.
scaleCenter	A pair of Floats specifying the center of scale.
objectList	A sequence of *ConstrainedSketchGeometry* objects specifying the objects to be scaled.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`setPrimaryObject(...)`

This method makes the ConstrainedSketch object the primary object in the current viewport. The sketch remains the primary object in the current viewport until an `unsetPrimaryobject` command is issued.

Required arguments

option A SymbolicConstant specifying how the sketch is displayed. Possible values are:

- STANDALONE: Indicates a new stand-alone sketch. The current viewport is cleared and is replaced by the stand-alone sketch. The view direction is set to $-\boldsymbol{Z}$.
- SUPERIMPOSE: Indicates that the sketch is superimposed on the current viewport. The view direction is changed to be perpendicular to the sketch plane. The change is effected smoothly as an animated sequence of many small viewing steps.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

`trimExtendCurve(...)`

This method trims or extends a specified *ConstrainedSketchGeometry* object (*curve1*) using another specified *ConstrainedSketchGeometry* object (*curve2*). *curve2* is not affected by the operation. The location for the trim or extend is determined by the specified point values.

Required arguments

curve1 The *ConstrainedSketchGeometry* object specifying the object to be trimmed or extended.

point1 A pair of Floats specifying the location on *curve1* where trim or extend should be applied.

curve2 The *ConstrainedSketchGeometry* object specifying the object to which *curve1* is trimmed or extended. *curve2* is not trimmed or extended.

point2 A pair of Floats specifying the location on *curve2* near where *curve1* should be trimmed or extended.

Optional arguments

None.

Return value
`None.`

Exceptions
None.

`undo()`

This method undoes the effects of the last ConstrainedSketch object method.

Arguments
None.

Return value
`None.`

Exceptions
None.

`unsetPrimaryObject()`

This method removes the ConstrainedSketch object from the current viewport, reversing the effects of the `setPrimaryobject` command. If the *option* argument was set to SUPERIMPOSE, the viewport will be returned to the view orientation that was in place when the `setPrimaryobject` command was issued. If the *option* argument was set to STANDALONE, the viewport will be left empty.

Arguments
None.

Return value
`None.`

Exceptions
None.

`writeAcisFile(...)`

This method exports the geometry of the sketch to a named file in ACIS format.

Required arguments

fileName A String specifying the file name.

Optional arguments

version A Float specifying the ACIS version. For example, the Float "12.0" corresponds to ACIS Version 12.0. The default value is the current version of ACIS.

Return value
None.

Exceptions
InvalidNameError.

writeIgesFile(...)

This method exports the geometry of the sketch to a named file in IGES format.

Required arguments

filename A String specifying the file name.

Optional arguments

flavor A SymbolicConstant specifying a particular flavor of IGES to export. Possible values are STANDARD, AUTOCAD, SOLIDWORKS, JAMA, and MSBO.

Return value
None.

Exceptions
InvalidNameError.

Members

The ConstrainedSketch object can have the following members:

constraints A repository of *ConstrainedSketchConstraint* objects.

dimensions A repository of *ConstrainedSketchDimension* objects.

geometry A *ConstrainedSketchGeometryArray* object specifying the sketch geometry, such as lines, arcs, circles, and splines.

parameters A repository of *ConstrainedSketchParameter* objects specifying sketch parameters, which may be associated with dimensions.

sketchOptions A *ConstrainedSketchOptions* object specifying the sketch option settings.

vertices A *ConstrainedSketchVertexArray* object.

imageOptions A *ConstrainedSketchImageOptions* object.

ConstrainedSketchConstraint object

The ConstrainedSketchConstraint object stores the constraints associated with a sketch.

Access

```
import sketch
mdb.models[name].sketches[name].constraints[i]
```

CoincidentConstraint(...)

This method creates a coincident constraint. This constraint applies to two vertices, to a vertex and a *ConstrainedSketchGeometry* object, or to two *ConstrainedSketchGeometry* objects of the same type and constrains them to be coincident.

Path

```
mdb.models[name].sketches[name].CoincidentConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object or a *Vertex* object specifying the first object.

entity2 A *ConstrainedSketchGeometry* object or a *Vertex* object specifying the second object.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

ConcentricConstraint(...)

This method creates a concentric constraint. This constraint applies to any combination of circles, arcs, ellipses, and points and constrains them to be concentric. A concentric constraint implies that the center of *ConstrainedSketchGeometry* objects coincide.

Path

```
mdb.models[name].sketches[name].ConcentricConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object specifying the first arc, circle, ellipse, or sketch vertex.

entity2 A *ConstrainedSketchGeometry* object specifying the second arc, circle, ellipse, or sketch vertex.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

EqualLengthConstraint(...)

This method creates an equal length constraint. This constraint applies to lines and constrains them such that their lengths are equal.

Path

```
mdb.models[name].sketches[name].EqualLengthConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object specifying the first line.

entity2 A *ConstrainedSketchGeometry* object specifying the second line.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

EqualRadiusConstraint(...)

This method creates an equal radius constraint. This constraint applies to circles and arcs and constrains them such that their radii are equal.

Path

```
mdb.models[name].sketches[name].EqualRadiusConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object specifying the first arc or circle.

entity2 A *ConstrainedSketchGeometry* specifying the second arc or circle.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

FixedConstraint(...)

This method creates a fixed constraint. This constraint applies to a *ConstrainedSketchGeometry* object or a *Vertex* object and constrains them to be fixed in space. Both the location and the shape of the sketch geometry is fixed.

Path

```
mdb.models[name].sketches[name].FixedConstraint
```

Required arguments

entity A *ConstrainedSketchGeometry* object or a *Vertex* object specifying the item to fix in space.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

HorizontalConstraint(...)

This method creates a horizontal constraint. This constraint applies to a line and constrains it to be horizontal.

Path

```
mdb.models[name].sketches[name].HorizontalConstraint
```

Required arguments

entity A *ConstrainedSketchGeometry* object specifying the line to constrain.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

VerticalConstraint(...)

This method creates a vertical constraint. This constraint applies to a line and constrains it to be vertical.

Path

```
mdb.models[name].sketches[name].VerticalConstraint
```

Required arguments

entity A *ConstrainedSketchGeometry* object specifying the line to constrain.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

ParallelConstraint(...)

This method creates a parallel constraint. This constraint applies to lines and constrains them to be parallel.

Path

```
mdb.models[name].sketches[name].ParallelConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object specifying the first line.

entity2 A *ConstrainedSketchGeometry* object specifying the second line.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

PerpendicularConstraint(...)

This method creates a perpendicular constraint. This constraint applies to different types of *ConstrainedSketchGeometry* objects and constrains them to be perpendicular to each other.

Path

```
mdb.models[name].sketches[name].PerpendicularConstraint
```

Required arguments

entity1
A *ConstrainedSketchGeometry* object specifying the first object.

entity2
A *ConstrainedSketchGeometry* object specifying the second object.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

EqualDistanceConstraint(...)

This method creates an equal distance constraint. This constraint can be applied between a midpoint *Vertex* object and any other two *Vertex* objects or between a midpoint *Vertex* object and two *ConstrainedSketchGeometry* objects that are lines. The equal distance constraint forces the midpoint vertex to remain at an equal distance from the two other vertices or lines.

Path

```
mdb.models[name].sketches[name].EqualDistanceConstraint
```

Required arguments

entity1
A*ConstrainedSketchGeometry* object specifying the first line or *Vertex* object.

entity2 A *ConstrainedSketchGeometry* object specifying the second line or *Vertex* object.

midpoint A *Vertex* object specifying the vertex that will be positioned an equal distance from *entity1* and *entity2*.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

TangentConstraint(...)

This method creates a tangent constraint. This constraint applies to different types of *ConstrainedSketchGeometry* objects and constrains them to remain tangential.

Path

```
mdb.models[name].sketches[name].TangentConstraint
```

Required arguments

entity1 A *ConstrainedSketchGeometry* object specifying the first object.

entity2 A *ConstrainedSketchGeometry* object specifying the second object.

Optional arguments

None.

Return value

A ConstrainedSketchConstraint object.

Exceptions

None.

Members

The ConstrainedSketchConstraint object has no members.

ConstrainedSketchDimension object

The ConstrainedSketchDimension object stores the dimensions associated with a sketch.

Access

```
import sketch
mdb.models[name].sketches[name].dimensions[i]
```

AngularDimension(...)

This method constructs a ConstrainedSketchDimension object between two *ConstrainedSketchGeometry* objects, with the given angle between them.

Path

```
mdb.models[name].sketches[name].AngularDimension
```

Required arguments

line1 A *ConstrainedSketchGeometry* object specifying the first line.

line2 A *ConstrainedSketchGeometry* object specifying the second line.

textPoint A pair of Floats specifying the location of the dimension text.

Optional arguments

value A Float specifying the angle between the two lines.

reference A Boolean specifying whether the created dimension enforces the above value or if it simply measures the angle between two lines.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

HorizontalDimension(...)

This method constructs a ConstrainedSketchDimension object between two vertices. A horizontal dimension indicates the horizontal distance along the *X*-axis between two vertices.

Path

```
mdb.models[name].sketches[name].HorizontalDimension
```

Required arguments

vertex1 A *ConstrainedSketchVertex* object specifying the first endpoint.

vertex2 A *ConstrainedSketchVertex* object specifying the second endpoint.

textPoint A pair of Floats specifying the location of the dimension text.

Optional arguments

value A Float distance between the two vertices.

reference A Boolean specifying whether the created dimension enforces the above value or if it simply measures the distance between the two vertices.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

ObliqueDimension(...)

This method constructs a ConstrainedSketchDimension object between two vertices. An oblique dimension indicates the distance between two vertices.

Path

```
mdb.models[name].sketches[name].ObliqueDimension
```

Required arguments

vertex1 A *ConstrainedSketchVertex* object specifying the first endpoint.

vertex2 A *ConstrainedSketchVertex* object specifying the second endpoint.

textPoint A pair of Floats specifying the location of the dimension text.

Optional arguments

value A Float specifying the distance between the two *ConstrainedSketchVertex* objects.

reference A Boolean specifying whether the created dimension enforces the above value or if it simply measures the distance between the two vertices.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

RadialDimension(...)

This method constructs a ConstrainedSketchDimension object on a circular or elliptical arc. A radial dimension indicates the radius of an arc or circle or the major or minor radius of an ellipse.

Path

```
mdb.models[name].sketches[name].RadialDimension
```

Required arguments

curve A *ConstrainedSketchGeometry* object specifying the circular or elliptical arc.

textPoint A pair of Floats specifying the location of the dimension text.

Optional arguments

value A Float specifying the radius of the arc, circle or ellipse.

reference A Boolean specifying whether the created dimension enforces the above value or if it simply measures the angle between two lines.

majorRadius A Float specifying the major Radius if *curve* is an ellipse. This is mutually exclusive with *value* and *minorRadius*.

minorRadius A Float specifying the minor Radius if *curve* is an ellipse. This is mutually exclusive with *value* and *majorRadius*.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

VerticalDimension(...)

This method constructs a ConstrainedSketchDimension between two vertices. A vertical dimension controls the vertical distance along the *Y*-axis between two vertices.

Path

```
mdb.models[name].sketches[name].VerticalDimension
```

Required arguments

vertex1	A *ConstrainedSketchVertex* object specifying the first endpoint.
vertex2	A *ConstrainedSketchVertex* object specifying the second endpoint.
textPoint	A pair of Floats specifying the location of the dimension text.

Optional arguments

value	A Float specifying the angle between the two lines.
reference	A Boolean specifying whether the created dimension enforces the above value or if it simply measures the angle between two lines.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

DistanceDimension(...)

This method constructs a ConstrainedSketchDimension object between two *ConstrainedSketchGeometry*, or a*ConstrainedSketchVertex* and *ConstrainedSketchGeometry* object. A distance dimension specifies the shortest distance between two entities.

Path

```
mdb.models[name].sketches[name].DistanceDimension
```

Required arguments

entity1	A *ConstrainedSketchVertex* object or *ConstrainedSketchGeometry* object.
entity2	A *ConstrainedSketchVertex* object or *ConstrainedSketchGeometry* object.
textPoint	A pair of Floats specifying the location of the dimension text.

Optional arguments

value	A Float specifying the angle between the two lines.
reference	A Boolean specifying whether the created dimension enforces the above value or if it simply measures the angle between two lines.

Return value

A ConstrainedSketchDimension object (`None` if the dimension cannot be created).

Exceptions

None.

Members

The ConstrainedSketchDimension object has no members.

ConstrainedSketcherOptions object

The ConstrainedSketcherOptions object is used to store values and attributes which will be applied to all sketches used in the current session. The ConstrainedSketcherOptions object has no constructor.

Access

```
import sketch
session.sketcherOptions
```

`setValues(...)`

This method modifies the *ConstrainedSketchOptions* object.

Required arguments

None.

Optional arguments

constructionGeometry A Boolean specifying whether construction geometry is shown. The default value is ON.

gridSnap A Boolean specifying whether the cursor snaps to the grid. The default value is ON.

preselection A Boolean specifying whether geometry will be preselected. The default value is ON.

addImpliedConstraints A Boolean specifying if implied constraints are added during sketching. The default value is ON.

maxCoplanarEntities An Int specifying the maximum number of coplanar entities which should be automatically projected from the background, when a sketch based feature is created or edited. When this value is exceeded no entities are automatically projected and a warning issued. Possible values are *maxCoplanarEntities* > 0. The default value is 300.

autoConstrainAngularTolerance A Float specifying the angular tolerance in degrees which is used to determine parallel and tangential conditions during the auto-constrain operation. For example any two lines which have an angle smaller than the given *autoConstrainAngularTolerance* will be assumed to be parallel, and a parallel constrain may be added during the auto-constrain operation. The default value is 0.01.

autoConstrainLinearTolerance A Float specifying the linear tolerance which is used to determine when two points or geometries are coincident during the auto-constrain operation. The default value is 10^{-6}.

autoConstrainOptions
A sequence of SymbolicConstants specifying which type of constraints may be added by the auto-constraint tool. Possible values are PARALLEL, PERPENDICULAR, IDENTICAL, TANGENT, CONCENTRIC, and EQUALRADIUS. The default value is (PARALLEL,, PERPENDICULAR,, IDENTICAL,, TANGENT,, CONCENTRIC,, EQUALRADIUS).

dragMethod
A SymbolicConstant specifying the constraint solving mode used by the sketcher during drag operation. Possible values are MINIMUM_MOVE, STANDARD, WEIGHTED, and RELAXATION. The default value is MINIMUM_MOVE.

editMethod
A SymbolicConstant specifying the constraint solving mode used by the sketcher during regular sketch editing and adding new constraints and dimensions. Possible values are MINIMUM_MOVE, STANDARD, WEIGHTED, and RELAXATION. The default value is STANDARD.

Return value
`None.`

Exceptions
RangeError.

Members
The ConstrainedSketcherOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

ConstrainedSketchGeometry object

The ConstrainedSketchGeometry object stores the geometry of a sketch, such as lines, circles, arcs, and construction lines.

Access

```
import sketch
mdb.models[name].sketches[name].geometry[i]
mdb.models[name].sketches[name].geometry[i][i]
```

Arc3Points(...)

This method constructs an arc using a two endpoints and an intermediate third point on the arc.

Path

```
mdb.models[name].sketches[name].Arc3Points
```

Required arguments

point1 A pair of Floats specifying the first endpoint of the arc.

point2 A pair of Floats specifying the second endpoint of the arc.

point3 A pair of Floats specifying the third point on the arc.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the arc cannot be created).

Exceptions

None.

ArcByCenterEnds(...)

This method constructs an arc using a center point and two vertices. The Arc object is added to the geometry repository of the *ConstrainedSketch* object. The arc is created in a clockwise fashion from *point1* to *point2*.

Path

```
mdb.models[name].sketches[name].ArcByCenterEnds
```

Required arguments

center A pair of Floats specifying the center point of the arc.

point1 A pair of Floats specifying the first endpoint of the arc.

point2 A pair of Floats specifying the second endpoint of the arc.

direction A SymbolicConstant specifying the direction of the arc. Possible values are CLOCKWISE and COUNTERCLOCKWISE.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the arc cannot be created).

Exceptions

If incompatible data are given, the second endpoint is ignored.

ArcByStartEndTangent(...)

This method constructs an arc using two vertices. The Arc object is added to the geometry repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].ArcByStartEndTangent
```

Required arguments

point1 A pair of Floats specifying the first endpoint of the arc.

point2 A pair of Floats specifying the second endpoint of the arc.

vector A sequence of two Floats specifying the start direction for constructing the arc.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the arc cannot be created).

Exceptions

None.

CircleByCenterPerimeter(...)

This method constructs a circle using a center point and a point on the perimeter. The circle is added to the geometry repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].CircleByCenterPerimeter
```

Required arguments

center A pair of Floats specifying the center point of the circle.

point1 A pair of Floats specifying a point on the perimeter of the circle.

Return value

A ConstrainedSketchGeometry object (`None` if the circle cannot be created).

Exceptions

None.

ConstructionCircleByCenterPerimeter(...)

This method constructs a construction circle using a center point and a point on the perimeter. The circle is added to the geometry repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].ConstructionCircleByCenterPerimeter
```

Required arguments

center A pair of Floats specifying the center point of the construction circle.

point1 A pair of Floats specifying a point on the perimeter of the construction circle.

Return value

A ConstrainedSketchGeometry object (`None` if the circle cannot be created).

Exceptions

None.

EllipseByCenterPerimeter(...)

This method constructs an ellipse using a center point, a major axis point, and a minor axis point. The ellipse is added to the geometry repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].EllipseByCenterPerimeter
```

Required arguments

center A pair of Floats specifying the center point of the ellipse.

axisPoint1 A pair of Floats specifying the major or minor axis point of the ellipse.

axisPoint2 A pair of Floats specifying the minor or major axis point of the ellipse.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the ellipse cannot be created).

Exceptions

None.

FilletByRadius(...)

This method constructs a fillet arc of a given radius between two curves. The fillet is added to the geometry repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].FilletByRadius
```

Required arguments

radius A Float specifying the radius of the fillet arc. Possible values are Floats > 0.

curve1 A ConstrainedSketchGeometry object specifying the first curve.

nearPoint1 A pair of Floats specifying a point on the sketch near where the user wishes the fillet to intersect with *curve1*. This point does not need to be on*curve1*; it is used as a hint to draw the fillet.

curve2 A ConstrainedSketchGeometry object specifying the second curve.

nearPoint2 A pair of Floats specifying a point on the sketch near where the user wishes the fillet to intersect with *curve2*. This point does not need to be on *curve2*; it is used as a hint to draw the fillet.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the fillet cannot be created).

Exceptions

If the radius given cannot be used to create a fillet between the two curves given:	Range Error: cannot construct the Fillet specified

Line(...)

This method creates a line between two given points.

Path

```
mdb.models[name].sketches[name].Line
```

Required arguments

point1	A pair of Floats specifying the first endpoint.
point2	A pair of Floats specifying the second endpoint.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the line cannot be created).

Exceptions

None.

ConstructionLine(...)

This method creates an oblique construction line that runs between two given points.

Path

```
mdb.models[name].sketches[name].ConstructionLine
```

Required arguments

point1	A pair of Floats specifying the first endpoint.
point2	A pair of Floats specifying the second endpoint.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the line cannot be created).

Exceptions

None.

Spline(...)

This method creates a spline curve running through a sequence of points.

Path

```
mdb.models[name].sketches[name].Spline
```

Required arguments

points A sequence of pairs of Floats specifying the points through which the spline passes.

Optional arguments

constrainPoints A Boolean that determines whether the points given are to constrained to always remain on the Spline. The default is True. For a large sequence of *points*, significant performance gains may be achieved by setting the value to False.

Return value

A ConstrainedSketchGeometry object (`None` if the spline cannot be created).

Exceptions

None.

Spot(...)

This method creates a spot construction point located at the specified coordinates. The spot is added to the vertex repository of the *ConstrainedSketch* object.

Path

```
mdb.models[name].sketches[name].Spot
```

Required arguments

point A pair of Floats specifying the coordinates of the spot construction point.

Optional arguments

None.

Return value

A ConstrainedSketchGeometry object (`None` if the spot cannot be created).

Exceptions

None.

getVertices()

This method returns an list of *ConstrainedSketchVertex* objects which are a part of the given ConstrainedSketchGeometry object.

Arguments

None.

Return value

A list of *ConstrainedSketchVertex* objects.

Exceptions

None.

getSize()

This method returns the length of the given ConstrainedSketchGeometry object.

Arguments

None.

Return value

The length of the given ConstrainedSketchGeometry.

Exceptions

None.

getPointAtDistance(...)

This method returns a point offset along the given ConstrainedSketchGeometry from the given end by a specified arc length distance or a percentage of the total length of the ConstrainedSketchGeometry object.

Required arguments

point A pair of Floats specifying the point from which the distance is to be measured.

distance A float specifying the arc length distance along the ConstrainedSketchGeometry from the *point* at which the required point is situated.

Optional arguments

percentage A Boolean that specifies if the *distance* is an absolute distance or is a fraction relative to the length of the ConstrainedSketchGeometry object.

Return value

A pair of floats representing the point along the edge.

Exceptions

None.

Members

The ConstrainedSketchGeometry object has the following members:

id
An Int specifying the index of the sketch entity in the ConstrainedSketchGeometryArray.

curveType
A SymbolicConstant specifying the geometry of the sketch entity. Possible values are ARC, CIRCLE, ELLIPSE, LINE, and SPLINE.

type
A SymbolicConstant specifying the type of sketch entity. Possible values are REGULAR, REFERENCE, and CONSTRUCTION.

pointOn
A tuple of Floats specifying the *X*- and*Y*-coordinates of a point located on the geometry.

ConstrainedSketchGeometryArray object

The ConstrainedSketchGeometryArray is a sequence of *ConstrainedSketchGeometry* objects.

Access

```
import sketch
mdb.models[name].sketches[name].geometry[i]
```

`findAt(...)`

This method returns the *ConstrainedSketchGeometry* object located at the given coordinates.

Required arguments

coordinates A sequence of Floats specifying the *X*- and *Y*-coordinates of the object to find.

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

A *ConstrainedSketchGeometry* object.

Exceptions

None.

Members

The ConstrainedSketchGeometryArray object has no members.

ConstrainedSketchImageOptions object

The ConstrainedSketchImageOptions object is used to store values and attributes associated with the background image for a particular sketch. The ConstrainedSketchImageOptions object has no constructor.

Access

```
import sketch
mdb.models[name].sketches[name].imageOptions
```

`setValues(...)`

This method modifies the *ConstrainedSketchOptions* object.

Required arguments

None.

Optional arguments

imageName

A String specifying the name of the image. A list of valid image names is in the *images* repository in the *session* object.

showImage

A Boolean specifying whether an image should be displayed in the sketcher background. The default value is OFF.

origin

A pair of Floats specifying the *X*- and *Y*-offsets in millimeters from the lower-left corner of the viewport. The default value is (0, 0).

xScale

A Float specifying the scale applied to the image width. The default value is 1.0.

When *xScale* is negative, the image is mirrored about its y-axis but its position is not affected.

yScale

A Float specifying the scale applied to the image height. The default value is 1.0.

When *yScale* is negative, the image is mirrored about its x-axis but its position is not affected.

translucency

A Float specifying the translucency factor to use when displaying the image. Possible values are $0.0 \leq$ *translucency* ≤ 1.0 with 0.0 being invisible and 1.0 being opaque. The default value is 1.0.

Return value

`None.`

Exceptions

RangeError.

Members

The ConstrainedSketchImageOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

ConstrainedSketchOptions object

The ConstrainedSketchOptions object is used to store values and attributes associated with a particular sketch. The ConstrainedSketchOptions object has no constructor.

Access

```
import sketch
mdb.models[name].sketches[name].sketchOptions
```

setValues(...)

This method modifies the ConstrainedSketchOptions object.

Required arguments

None.

Optional arguments

sheetSize

A Float specifying the sheet size. Possible values are Floats $>$ 0. The default value is the *sheetSize* specified with the Sketch method.

gridSpacing

A Float specifying the spacing between gridlines. Possible values are Floats $>$ 0. The default value is approximately 2.5% of *sheetSize*.

grid

A Boolean specifying whether the grid is shown. The default value is ON.

gridFrequency

An Int specifying how often gridlines are shown. Possible values are *gridFrequency* $>$ 0. The default value is 1.

dimensionTextHeight

A Float specifying the height of the dimension text in points. Possible values are Floats $>$ 0. The default value is 12.0.

decimalPlaces

An Int specifying how many decimal places are shown in dimensions. Possible values are $0 \leq$ *decimalPlaces* ≤ 6. The initial value depends on the value of *sheetSize*.

constructionGeometry

A Boolean specifying whether construction geometry is shown. The default value is ON.

gridSnap

A Boolean specifying whether the cursor snaps to the grid. The default value is ON.

preselection

A Boolean specifying whether geometry will be preselected. The default value is ON.

sheetAuto

A Boolean specifying if the sheet size and the grid spacing are automatically computed. The default value is ON.

gridOrigin
A sequence of Floats specifying the *X*–*Y* coordinates for the origin of the grid. The default value is (0, 0).

gridAngle
A Float specifying the angle of the grid relative to the computer screen. The default value is 0.0.

viewStyle
A SymbolicConstant specifying the type of sketch displayed in the viewport. Possible values are REGULAR and AXISYM. The default value is REGULAR.

addImpliedConstraints
A Boolean specifying if implied constraints are added during sketching. The default value is ON.

maxCoplanarEntities
An Int specifying the maximum number of coplanar entities which should be automatically projected from the background, when a sketch based feature is created or edited. When this value is exceeded no entities are automatically projected and a warning issued. Possible values are *maxCoplanarEntities* > 0. The default value is 300.

autoConstrainAngularTolerance
A Float specifying the angular tolerance in degrees which is used to determine parallel and tangential conditions during the auto-constrain operation. For example any two lines which have an angle smaller than the given *autoConstrainAngularTolerance* will be assumed to be parallel, and a parallel constrain may be added during the auto-constrain operation. The default value is 0.01.

autoConstrainLinearTolerance
A Float specifying the linear tolerance which is used to determine when two points or geometries are coincident during the auto-constrain operation. The default value is 10^{-6}.

autoConstrainOptions
A sequence of SymbolicConstants specifying which type of constraints may be added by the auto-constraint tool. Possible values are PARALLEL, PERPENDICULAR, IDENTICAL, TANGENT, CONCENTRIC, and EQUALRADIUS. The default value is (PARALLEL,, PERPENDICULAR,, IDENTICAL,, TANGENT,, CONCENTRIC,, EQUALRADIUS).

dragMethod
A SymbolicConstant specifying the constraint solving mode used by the sketcher during drag operation. Possible values are MINIMUM_MOVE, STANDARD, WEIGHTED, and RELAXATION. The default value is MINIMUM_MOVE.

editMethod
A SymbolicConstant specifying the constraint solving mode used by the sketcher during regular sketch editing and adding new constraints and dimensions. Possible values are MINIMUM_MOVE, STANDARD, WEIGHTED, and RELAXATION. The default value is STANDARD.

Return value
`None.`

Exceptions
RangeError.

Members

The ConstrainedSketchOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

ConstrainedSketchParameter object

The ConstrainedSketchParameter object stores the definition of a parameter in the sketch.

Access

```
import sketch
mdb.models[name].sketches[name].parameters[i]
```

Parameter(...)

This method creates a parameter and optionally associates a dimension with this parameter.

Path

```
mdb.models[name].sketches[name].Parameter
```

Required arguments

name A String specifying the name of the ConstrainedSketchParameter object. No two parameters in the same *ConstrainedSketch* can have the same name.

Optional arguments

path A String specifying the *ConstrainedSketchDimension* object with which this parameter is associated.

expression A String specifying the expression or value associated with the ConstrainedSketchParameter.

previousParameter A String specifying the name of the previous ConstrainedSketchParameter, if it exists. The *previousParameter* argument implies an order among the parameters. No two parameters can reference the same parameter as the previous parameter.

Return value

A ConstrainedSketchParameter object.

Exceptions

None.

Members

The ConstrainedSketchParameter object has the following members:

name A String specifying the name of the ConstrainedSketchParameter object.

path A String specifying the path to the *ConstrainedSketchDimension* that depends on this ConstrainedSketchParameter.

expression
A String specifying an expression or value associated with this ConstrainedSketchParameter.

previousParameter
A String specifying the name of the ConstrainedSketchParameter that appears before this one in the ordered list.

ConstrainedSketchVertex object

The ConstrainedSketchVertex object stores the vertex position.

Access

```
import sketch
mdb.models[name].sketches[name].vertices[i]
mdb.models[name].sketches[name].vertices[i][i]
```

Spot(...)

This method creates a spot (construction point) located at the specified coordinates.

Path

```
mdb.models[name].sketches[name].Spot
```

Required arguments

point A pair of Floats specifying the coordinates of the construction point.

Optional arguments

None.

Return value

A ConstrainedSketchVertex object (`None` if the spot cannot be created).

Exceptions

None.

Members

The ConstrainedSketchVertex object has the following member:

coords A tuple of Floats specifying the*X*-, *Y*-, and *Z*-coordinates of the sketch vertex.

ConstrainedSketchVertexArray object

The ConstrainedSketchVertexArray is a sequence of *ConstrainedSketchVertex* objects.

Access

```
import sketch
mdb.models[name].sketches[name].vertices[i]
```

`findAt(...)`

This method returns the *ConstrainedSketchVertex* located at the given coordinates.

Required arguments

coordinates A sequence of Floats specifying the *X*- and *Y*-coordinates of the object to find.

Optional arguments

printWarning A Boolean specifying whether a message is to be printed to the CLI if no entity is found at the specified location. The default value is True.

Return value

A *ConstrainedSketchVertex* object.

Exceptions

None.

Members

The ConstrainedSketchVertexArray object has no members.

Step commands (step)

The Step commands described in this chapter are used to create and configure analysis steps. *Step commands (output)* describes the commands used to create and configure output requests and integrated output sections and the commands to configure diagnostic printing, monitoring, and restart. *Step commands (miscellaneous)* describes the commands used to configure controls, damping, and frequency tables.

In this section:

- *Step object*
- *AnalysisStep object*
- *AnnealStep object*
- *BuckleStep object*
- *ComplexFrequencyStep object*
- *CoupledTempDisplacementStep object*
- *CoupledThermalElectricalStructuralStep object*
- *CoupledThermalElectricStep object*
- *CoupledThermalElectrochemicalStep object*
- *DirectCyclicStep object*
- *EmagTimeHarmonicStep object*
- *ExplicitDynamicsStep object*
- *FrequencyStep object*
- *GeostaticStep object*
- *HeatTransferStep object*
- *ImplicitDynamicsStep object*
- *InitialStep object*
- *MassDiffusionStep object*
- *ModalDynamicsStep object*
- *RandomResponseStep object*
- *ResponseSpectrumStep object*
- *SoilsStep object*
- *StaticLinearPerturbationStep object*
- *StaticRiksStep object*
- *StaticStep object*
- *SteadyStateDirectStep object*
- *SteadyStateModalStep object*
- *SteadyStateSubspaceStep object*
- *SubspaceDynamicsStep object*
- *SubstructureGenerateStep object*
- *TempDisplacementDynamicsStep object*
- *ViscoStep object*

Step object

The Step object stores the parameters that determine the context of the step. The Step object is the abstract base type for other Step objects. The Step object has no explicit constructor. The methods and members of the Step object are common to all objects derived from the Step.

Access

```
import step
mdb.models[name].steps[name]
```

resume()

This method resumes the step that was previously suppressed.

Arguments

None.

Return value

None.

Exceptions

None.

suppress()

This method suppresses the step.

Arguments

None.

Return value

None.

Exceptions

None.

Members

The Step object can have the following members:

name	A String specifying the repository key.
explicit	A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).
perturbation	A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed

A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState

A repository of *FieldOutputRequestState* objects.

historyOutputRequestState

A repository of *HistoryOutputRequestState* objects.

diagnosticPrint

A *DiagnosticPrint* object.

monitor

A *Monitor* object.

restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.
activateElements	A repository of *ActivateElements* objects.

AnalysisStep object

The AnalysisStep object is the abstract base type for other Step objects, except the *InitialStep* object. It has no explicit constructor, no methods, and has only inherited members.

The AnalysisStep object is derived from the *Step* object.

Access

```
import step
mdb.models[name].steps[name]
```

Members

The AnalysisStep object can have the following members:

name

A String specifying the repository key.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM

- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates
A repository of *PredefinedFieldState* objects.

AnnealStep object

The AnnealStep object anneals a structure by setting the velocities and all appropriate state variables to zero.

The AnnealStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

AnnealStep(...)

This method creates an AnnealStep object.

Path

```
mdb.models[name].AnnealStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

refTemp A Float specifying the post-anneal reference temperature. The default value is the current temperature at all nodes in the model after the annealing has completed.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

Return value

An AnnealStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the AnnealStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *AnnealStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The AnnealStep object can have the following members:

name

A String specifying the repository key.

refTemp

A Float specifying the post-anneal reference temperature. The default value is the current temperature at all nodes in the model after the annealing has completed.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL

- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **ANNEAL*
- **STEP*

BuckleStep object

The BuckleStep object controls eigenvalue buckling estimation.

The BuckleStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

BuckleStep(...)

This method creates a BuckleStep object.

Path

```
mdb.models[name].BuckleStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

numEigen
An Int specifying the number of eigenvalues to be estimated.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

eigensolver
A SymbolicConstant specifying the eigensolver. Possible values are SUBSPACE and LANCZOS. The default value is SUBSPACE.

minEigen
`None` or a Float specifying the minimum eigenvalue of interest. The default value is `None`.

maxEigen
`None` or a Float specifying the maximum eigenvalue of interest. The default value is `None`.

vectors
An Int specifying the number of vectors used in the iteration. The default value is the minimum of $(2n, n + 8)$, where n is the number of eigenvalues requested.

maxIterations
An Int specifying the maximum number of iterations. The default value is 30.

blockSize
The SymbolicConstant DEFAULT or an Int specifying the size of the Lanczos block steps. The default value is DEFAULT.

maxBlocks
The SymbolicConstant DEFAULT or an Int specifying the maximum number of Lanczos block steps within each Lanczos run. The default value is DEFAULT.



Note:

minEigen, *blockSize*, and *maxBlocks* are ignored unless *eigensolver*=LANCZOS.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

Return value

A BuckleStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the BuckleStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BuckleStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The BuckleStep object can have the following members:

name

A String specifying the repository key.

numEigen

An Int specifying the number of eigenvalues to be estimated.

eigensolver

A SymbolicConstant specifying the eigensolver. Possible values are SUBSPACE and LANCZOS. The default value is SUBSPACE.

minEigen

`None` or a Float specifying the minimum eigenvalue of interest. The default value is `None`.

maxEigen

`None` or a Float specifying the maximum eigenvalue of interest. The default value is `None`.

vectors

An Int specifying the number of vectors used in the iteration. The default value is the minimum of ($2n$, $n + 8$), where n is the number of eigenvalues requested.

maxIterations

An Int specifying the maximum number of iterations. The default value is 30.

blockSize

The SymbolicConstant DEFAULT or an Int specifying the size of the Lanczos block steps. The default value is DEFAULT.

maxBlocks

The SymbolicConstant DEFAULT or an Int specifying the maximum number of Lanczos block steps within each Lanczos run. The default value is DEFAULT.



Note:

minEigen, *blockSize*, and *maxBlocks* are ignored unless *eigensolver*=LANCZOS.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC

- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **BUCKLE*
- **STEP*

ComplexFrequencyStep object

The ComplexFrequencyStep object is used to perform eigenvalue extraction to calculate the complex eigenvalues and corresponding complex mode shapes of a system.

The ComplexFrequencyStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ComplexFrequencyStep(...)

This method creates a ComplexFrequencyStep object.

Path

```
mdb.models[name].ComplexFrequencyStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

numEigen
The SymbolicConstant ALL or an Int specifying the number of complex eigenmodes to be calculated or a SymbolicConstant ALL. The default value is ALL.

description
A String specifying a description of the new step. The default value is an empty string.

shift
`None` or a Float specifying the shift point in cycles per time. The default value is `None`.

frictionDamping
A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

minEigen
`None` or a Float specifying the minimum frequency of interest in cycles per time. The default value is `None`.

maxEigen
`None` or a Float specifying the maximum frequency of interest in cycles per time. The default value is `None`.

propertyEvaluationFrequency
`None` or a Float specifying the frequency at which to evaluate frequency-dependent properties for viscoelasticity, springs, and dashpots during the eigenvalue extraction. If the value is `None`, the analysis product will evaluate the stiffness associated with frequency-dependent springs and dashpots at zero frequency and will not consider the stiffness contributions from frequency-domain viscoelasticity in the step. The default value is `None`.

Return value

A ComplexFrequencyStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ComplexFrequencyStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ComplexFrequencyStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ComplexFrequencyStep object can have the following members:

name
A String specifying the repository key.

numEigen
The SymbolicConstant ALL or an Int specifying the number of complex eigenmodes to be calculated or a SymbolicConstant ALL. The default value is ALL.

shift
`None` or a Float specifying the shift point in cycles per time. The default value is `None`.

frictionDamping
A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

minEigen
`None` or a Float specifying the minimum frequency of interest in cycles per time. The default value is `None`.

maxEigen
`None` or a Float specifying the maximum frequency of interest in cycles per time. The default value is `None`.

propertyEvaluationFrequency
`None` or a Float specifying the frequency at which to evaluate frequency-dependent properties for viscoelasticity, springs, and dashpots during the eigenvalue extraction. If the value is `None`, the analysis product will evaluate the stiffness associated with frequency-dependent springs and dashpots at zero frequency and will not consider the stiffness contributions from frequency-domain viscoelasticity in the step. The default value is `None`.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description
A String specifying a description of the new step. The default value is an empty string.

explicit
A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation
A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical
A Boolean specifying whether the step has a mechanical procedure type.

procedureType
A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL

- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **COMPLEX FREQUENCY*
- **STEP*

CoupledTempDisplacementStep object

The CoupledTempDisplacementStep object is used to analyze problems where the simultaneous solution of the temperature and stress/displacement fields is necessary.

The CoupledTempDisplacementStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

CoupledTempDisplacementStep(...)

This method creates a CoupledTempDisplacementStep object.

Path

```
mdb.models[name].CoupledTempDisplacementStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period for the step. The default value is 1.0.

nlgeom A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable and *stabilizationMethod* NONE. The default value is 2×10^{-4}.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

cetol

A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

creepIntegration

A SymbolicConstant specifying the type of integration to be used for creep and swelling effects throughout the step. Possible values are IMPLICIT, EXPLICIT, and NONE. The default value is IMPLICIT.

solutionTechnique

A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

Return value

A CoupledTempDisplacementStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the CoupledTempDisplacementStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CoupledTempDisplacementStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The CoupledTempDisplacementStep object can have the following members:

name A String specifying the repository key.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period for the step. The default value is 1.0.

nlgeom A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude
A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable and *stabilizationMethod* NONE. The default value is 2×10^{-4}.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

cetol
A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

creepIntegration
A SymbolicConstant specifying the type of integration to be used for creep and swelling effects throughout the step. Possible values are IMPLICIT, EXPLICIT, and NONE. The default value is IMPLICIT.

solutionTechnique
A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION

- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **COUPLED TEMPERATURE-DISPLACEMENT*
- **SOLUTION TECHNIQUE*
- **STEP*

CoupledThermalElectricalStructuralStep object

The CoupledThermalElectricalStructuralStep object is used to analyze problems where the simultaneous solution of the temperature, stress/displacement and electrical fields is necessary.

The CoupledThermalElectricalStructuralStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

CoupledThermalElectricalStructuralStep(...)

This method creates a CoupledThermalElectricalStructuralStep object.

Path

```
mdb.models[name].CoupledThermalElectricalStructuralStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period for the step. The default value is 1.0.

nlgeom A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable and *stabilizationMethod* NONE. The default value is 2×10^{-4}.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

cetol

A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

creepIntegration

A SymbolicConstant specifying the type of integration to be used for creep and swelling effects throughout the step. Possible values are IMPLICIT, EXPLICIT, and NONE. The default value is IMPLICIT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

Return value

A CoupledThermalElectricalStructuralStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the CoupledThermalElectricalStructuralStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CoupledThermalElectricalStructuralStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The CoupledThermalElectricalStructuralStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period for the step. The default value is 1.0.

nlgeom
A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod
A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude
A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable and *stabilizationMethod* NONE. The default value is 2×10^{-4}.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

cetol

A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

creepIntegration

A SymbolicConstant specifying the type of integration to be used for creep and swelling effects throughout the step. Possible values are IMPLICIT, EXPLICIT, and NONE. The default value is IMPLICIT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed

A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState	A repository of *FieldOutputRequestState* objects.
historyOutputRequestState	A repository of *HistoryOutputRequestState* objects.
diagnosticPrint	A *DiagnosticPrint* object.
monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **COUPLED TEMPERATURE-DISPLACEMENT*
- **SOLUTION TECHNIQUE*
- **STEP*

CoupledThermalElectricStep object

The CoupledThermalElectricStep object is used to analyze problems where the electrical potential and temperature fields must be solved simultaneously.

The CoupledThermalElectricStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

CoupledThermalElectricStep(...)

This method creates a CoupledThermalElectricStep object.

Path

```
mdb.models[name].CoupledThermalElectricStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period for the step. The default value is 1.0.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc A Float specifying the initial time increment. The default value is the total time period for the step.

minInc A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end
A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

mxdem
A Float specifying the maximum allowable emissivity change with temperature and field variables during an increment. The default value is 0.1.

solutionTechnique
A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A CoupledThermalElectricStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CoupledThermalElectricStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CoupledThermalElectricStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value
None.

Exceptions
RangeError.

Members
The CoupledThermalElectricStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period for the step. The default value is 1.0.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end
A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

mxdem
A Float specifying the maximum allowable emissivity change with temperature and field variables during an increment. The default value is 0.1.

solutionTechnique
A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM

- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **COUPLED THERMAL-ELECTRICAL*
- **SOLUTION TECHNIQUE*
- **STEP*

CoupledThermalElectrochemicalStep object

The CoupledThermalElectrochemicalStep object is used to analyze problems where the electrical potential, ion concentration, and temperature fields must be solved simultaneously.

The CoupledThermalElectrochemicalStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

CoupledThermalElectrochemicalStep(...)

This method creates a CoupledThermalElectrochemicalStep object.

Path

```
mdb.models[name].CoupledThermalElectrochemicalStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period for the step. The default value is 1.0.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

solutionTechnique
A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A CoupledThermalElectrochemicalStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the CoupledThermalElectrochemicalStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CoupledThermalElectrochemicalStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The CoupledThermalElectrochemicalStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period for the step. The default value is 1.0.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment in a transient analysis. The default value is 0.0.

solutionTechnique
A SymbolicConstant specifying the type of solution technique. Possible values are FULL_NEWTON and SEPARATED. The default value is FULL_NEWTON.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed	A Boolean specifying whether the step is suppressed or not. The default value is OFF.
fieldOutputRequestState	A repository of *FieldOutputRequestState* objects.
historyOutputRequestState	A repository of *HistoryOutputRequestState* objects.
diagnosticPrint	A *DiagnosticPrint* object.
monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **COUPLED THERMAL-ELECTROCHEMICAL*
- **SOLUTION TECHNIQUE*
- **STEP*

DirectCyclicStep object

The DirectCyclicStep object is used to provide a direct cyclic procedure for nonlinear, non-isothermal quasi-static analysis. It can also be used to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding growth at the interfaces in laminated composites in a low-cycle fatigue analysis.

The DirectCyclicStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

DirectCyclicStep(...)

This method creates a DirectCyclicStep object.

Path

```
mdb.models[name].DirectCyclicStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

timePeriod A Float specifying the time of single loading cycle. The default value is 1.0.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc A Float specifying the initial time increment. The default value is the total time period for the step.

minInc A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

maxNumIterations
An Int specifying the maximum number of iterations in a step. The default value is 200.

initialTerms
An Int specifying the initial number of terms in the Fourier series. The default value is 11.

maxTerms
An Int specifying the maximum number of terms in the Fourier series. The default value is 25.

termsIncrement
An Int specifying the increment in number of terms in the Fourier series. The default value is 5.

deltmx
A Float specifying the maximum temperature change to be allowed in an increment. The default value is 0.0.

cetol
A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

timePoints
`None` or a String specifying a String specifying the name of a time point object used to determine at which times the response of the structure will be evaluated. The default value is NONE.

fatigue
A Boolean specifying whether to include low-cycle fatigue analysis. The default value is OFF.

continueAnalysis
A Boolean specifying whether the displacement solution in the Fourier series obtained in the previous direct cyclic step is used as the starting values for the current step. The default value is OFF.

minCycleInc
An Int specifying the minimum number of cycles over which the damage is extrapolated forward. The default value is 100.

maxCycleInc
An Int specifying the maximum number of cycles over which the damage is extrapolated forward. The default value is 1000.

maxNumCycles
The SymbolicConstant DEFAULT or an Int specifying the maximum number of cycles allowed in a step or DEFAULT. A value of 1 plus half of the maximum number of cycles will be used if DEFAULT is specified. The default value is DEFAULT.

damageExtrapolationTolerance
A Float specifying the maximum extrapolated damage increment. The default value is 1.0.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

extrapolation A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A DirectCyclicStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the DirectCyclicStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DirectCyclicStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The DirectCyclicStep object can have the following members:

name A String specifying the repository key.

timePeriod A Float specifying the time of single loading cycle. The default value is 1.0.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

maxNumIterations

An Int specifying the maximum number of iterations in a step. The default value is 200.

initialTerms

An Int specifying the initial number of terms in the Fourier series. The default value is 11.

maxTerms

An Int specifying the maximum number of terms in the Fourier series. The default value is 25.

termsIncrement

An Int specifying the increment in number of terms in the Fourier series. The default value is 5.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment. The default value is 0.0.

cetol

A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

fatigue

A Boolean specifying whether to include low-cycle fatigue analysis. The default value is OFF.

continueAnalysis

A Boolean specifying whether the displacement solution in the Fourier series obtained in the previous direct cyclic step is used as the starting values for the current step. The default value is OFF.

minCycleInc

An Int specifying the minimum number of cycles over which the damage is extrapolated forward. The default value is 100.

maxCycleInc

An Int specifying the maximum number of cycles over which the damage is extrapolated forward. The default value is 1000.

maxNumCycles

The SymbolicConstant DEFAULT or an Int specifying the maximum number of cycles allowed in a step or DEFAULT. A value of 1 plus half of the maximum number of cycles will be used if DEFAULT is specified. The default value is DEFAULT.

damageExtrapolationTolerance A Float specifying the maximum extrapolated damage increment. The default value is 1.0.

matrixStorage A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

extrapolation A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description A String specifying a description of the new step. The default value is an empty string.

timePoints `None` or a String specifying a String specifying the name of a time point object used to determine at which times the response of the structure will be evaluated. The default value is NONE.

explicit A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical A Boolean specifying whether the step has a mechanical procedure type.

procedureType A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL

- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DIRECT CYCLIC*
- **STEP*

EmagTimeHarmonicStep object

The EmagTimeHarmonicStep object is used to calculate the electromagnetic response of the system to harmonic excitation of the model.

The EmagTimeHarmonicStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

EmagTimeHarmonicStep(...)

This method creates a EmagTimeHarmonicStep object.

Path

```
mdb.models[name].EmagTimeHarmonicStep
```

Required arguments

name	A String specifying the repository key.
previous	A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.
frequencyRange	An *EmagTimeHarmonicFrequencyArray* object.

Optional arguments

description	A String specifying a description of the new step. The default value is an empty string.
factorization	A SymbolicConstant specifying whether damping terms are to be ignored so that a real, rather than a complex, system matrix is factored. Possible values are REAL_ONLY and COMPLEX. The default value is COMPLEX.

Return value

An EmagTimeHarmonicStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the EmagTimeHarmonicStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EmagTimeHarmonicStep* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The EmagTimeHarmonicStep object has members with the same names and descriptions as the arguments to the *EmagTimeHarmonicStep* method.

In addition, the EmagTimeHarmonicStep object can have the following members:

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

subdivideUsingEigenfrequencies

A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is OFF.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT

- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **ELECTROMAGNETIC TIME HARMONIC*
- **STEP*

ExplicitDynamicsStep object

The ExplicitDynamicsStep object is used to perform a dynamic stress/displacement analysis using explicit integration in Abaqus/Explicit.

The ExplicitDynamicsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ExplicitDynamicsStep(...)

This method creates an ExplicitDynamicsStep object.

Path

```
mdb.models[name].ExplicitDynamicsStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

timePeriod A Float specifying the total time period for the step. The default value is 1.0.

nlgeom A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is ON.

adiabatic A Boolean specifying that an adiabatic stress analysis is to be performed. The default value is OFF.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are AUTOMATIC_GLOBAL, AUTOMATIC_EBE, FIXED_USER_DEFINED_INC, and FIXED_EBE. The default value is AUTOMATIC_GLOBAL.

maxIncrement `None` or a Float specifying the maximum time increment. If there is no upper limit, *maxIncrement*=`None`. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL or AUTOMATIC_EBE. The default value is `None`.

scaleFactor
A Float specifying the factor that is used to scale the time increment. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL, AUTOMATIC_EBE, or FIXED_EBE. The default value is 1.0.

massScaling
A tuple of tuples, where each tuple specifies the *MassScaling Object* controls. The default value is PREVIOUS_STEP.

linearBulkViscosity
A Float specifying the linear bulk viscosity parameter, b_1. The default value is 0.06.

quadBulkViscosity
A Float specifying the quadratic bulk viscosity parameter, b_2. The default value is 1.2.

userDefinedInc
`None` or a Float specifying the user-defined time increment. This argument is required only when *timeIncrementationMethod*=FIXED_USER_DEFINED_INC. The default value is `None`.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

improvedDtMethod
A Boolean specifying whether to use the "improved" (*improvedDtMethod*=ON) or "conservative" (*improvedDtMethod*=OFF) method to estimate the element stable time increment for three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements). The default value is ON.

Return value

An ExplicitDynamicsStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ExplicitDynamicsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ExplicitDynamicsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ExplicitDynamicsStep object can have the following members:

name

A String specifying the repository key.

timePeriod

A Float specifying the total time period for the step. The default value is 1.0.

nlgeom

A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is ON.

adiabatic

A Boolean specifying that an adiabatic stress analysis is to be performed. The default value is OFF.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are AUTOMATIC_GLOBAL, AUTOMATIC_EBE, FIXED_USER_DEFINED_INC, and FIXED_EBE. The default value is AUTOMATIC_GLOBAL.

maxIncrement

`None` or a Float specifying the maximum time increment. If there is no upper limit, *maxIncrement*=`None`. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL or AUTOMATIC_EBE. The default value is `None`.

scaleFactor

A Float specifying the factor that is used to scale the time increment. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL, AUTOMATIC_EBE, or FIXED_EBE. The default value is 1.0.

linearBulkViscosity

A Float specifying the linear bulk viscosity parameter, b_1. The default value is 0.06.

quadBulkViscosity

A Float specifying the quadratic bulk viscosity parameter, b_2. The default value is 1.2.

userDefinedInc

`None` or a Float specifying the user-defined time increment. This argument is required only when *timeIncrementationMethod*=FIXED_USER_DEFINED_INC. The default value is `None`.

improvedDtMethod

A Boolean specifying whether to use the "improved" (*improvedDtMethod*=ON) or "conservative" (*improvedDtMethod*=OFF) method to estimate the element stable time increment for three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements). The default value is ON.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

massScaling

A *MassScalingArray* object specifying mass scaling controls. The default value is PREVIOUS_STEP.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed

A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState	A repository of *FieldOutputRequestState* objects.
historyOutputRequestState	A repository of *HistoryOutputRequestState* objects.
diagnosticPrint	A *DiagnosticPrint* object.
monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **BULK VISCOSITY*
- **DYNAMIC*
- **FIXED MASS SCALING*
- **STEP*
- **VARIABLE MASS SCALING*

FrequencyStep object

The FrequencyStep object is used to perform eigenvalue extraction to calculate the natural frequencies and corresponding mode shapes of a system.

The FrequencyStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

FrequencyStep(...)

This method creates a FrequencyStep object.

Path

```
mdb.models[name].FrequencyStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

eigensolver
A SymbolicConstant specifying the eigensolver. Possible values are LANCZOS, SUBSPACE, and AMS.

The following optional arguments are ignored unless *eigensolver*=LANCZOS: *blockSize*, *maxBlocks*, *normalization*, *propertyEvaluationFrequency*.

The following optional arguments are ignored unless *eigensolver*=LANCZOS or AMS: *minEigen*, *maxEigen*, and *acousticCoupling*.

The following optional arguments are ignored unless *eigensolver*=AMS: *projectDamping*, *acousticRangeFactor*, *substructureCutoffMultiplier*, *firstCutoffMultiplier*, *secondCutoffMultiplier*, *residualModeRegion*, *regionalModeDof*, and *limitSavedEigenvectorRegion*.

Optional arguments

numEigen
The SymbolicConstant ALL or an Int specifying the number of eigenvalues to be calculated or ALL. The default value is ALL.

description
A String specifying a description of the new step. The default value is an empty string.

shift
A Float specifying the shift point in cycles per time. The default value is 0.0.

minEigen
`None` or a Float specifying the minimum frequency of interest in cycles per time. The default value is `None`.

maxEigen

`None` or a Float specifying the maximum frequency of interest in cycles per time. The default value is `None`.

vectors

`None` or an Int specifying the number of vectors used in the iteration. The default is the minimum of ($2n$, $n + 8$), where n is the number of eigenvalues requested. The default value is `None`.

maxIterations

An Int specifying the maximum number of iterations. The default value is 30.

blockSize

A SymbolicConstant specifying the size of the Lanczos block steps. The default value is DEFAULT.

maxBlocks

A SymbolicConstant specifying the maximum number of Lanczos block steps within each Lanczos run. The default value is DEFAULT.

normalization

A SymbolicConstant specifying the method for normalizing eigenvectors. Possible values are DISPLACEMENT and MASS. The default value is DISPLACEMENT.

A value of DISPLACEMENT indicates normalizing the eigenvectors so that the largest displacement entry in each vector is unity. A value of MASS indicates normalizing the eigenvectors with respect to the structure's mass matrix, which results in scaling the eigenvectors so that the generalized mass for each vector is unity.

propertyEvaluationFrequency

`None` or a Float specifying the frequency at which to evaluate frequency-dependent properties for viscoelasticity, springs, and dashpots during the eigenvalue extraction. If the value is `None`, the analysis product will evaluate the stiffness associated with frequency-dependent springs and dashpots at zero frequency and will not consider the stiffness contributions from frequency-domain viscoelasticity in the step. The default value is `None`.

projectDamping

A Boolean specifying whether to include projection of viscous and structural damping operators during *AMS* eigenvalue extraction. Valid only when *eigenSolver*=AMS. The default value is ON.

acousticCoupling

A SymbolicConstant specifying the type of acoustic-structural coupling in models with acoustic and structural elements coupled using the **TIE* option or in models with ASI-type elements. Possible values are AC_ON, AC_OFF, and AC_PROJECTION. The default value is AC_ON.

acousticRangeFactor

A Float specifying the ratio of the maximum acoustic frequency to the maximum structural frequency. The default value is 1.0.

frictionDamping

A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

simLinearDynamics A Boolean specifying whether to activate the SIM-based linear dynamics procedures. The default value is OFF.

residualModes A Boolean specifying whether to include residual modes from an immediately preceding Static, Linear Perturbation step. The default value is OFF.

substructureCutoffMultiplier A Float specifying the cutoff frequency for substructure eigenproblems, defined as a multiplier of the maximum frequency of interest. The default value is 5.0.

firstCutoffMultiplier A Float specifying the first cutoff frequency for a reduced eigenproblem, defined as a multiplier of the maximum frequency of interest. The default value is 1.7.

secondCutoffMultiplier A Float specifying the second cutoff frequency for a reduced eigenproblem defined as a multiplier of the maximum frequency of interest. The default value is 1.1.

residualModeRegion `None` or a sequence of Strings specifying the name of a region for which residual modes are requested. The default value is `None`.

residualModeDof `None` or a sequence of Ints specifying the degree of freedom for which residual modes are requested. The default value is `None`.

limitSavedEigenvectorRegion `None` or a *Region* object specifying a region for which eigenvectors should be saved or the SymbolicConstant None representing the whole model. The default value is `None`.

Return value

A FrequencyStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the FrequencyStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FrequencyStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None`.

Exceptions

RangeError.

Members

The FrequencyStep object can have the following members:

name

A String specifying the repository key.

eigensolver

A SymbolicConstant specifying the eigensolver. Possible values are LANCZOS, SUBSPACE, and AMS.

The following optional arguments are ignored unless *eigensolver*=LANCZOS: *blockSize*, *maxBlocks*, *normalization*, *propertyEvaluationFrequency*.

The following optional arguments are ignored unless *eigensolver*=LANCZOS or AMS: *minEigen*, *maxEigen*, and *acousticCoupling*.

The following optional arguments are ignored unless *eigensolver*=AMS: *projectDamping*, *acousticRangeFactor*, *substructureCutoffMultiplier*, *firstCutoffMultiplier*, *secondCutoffMultiplier*, *residualModeRegion*, *regionalModeDof*, and *limitSavedEigenvectorRegion*.

numEigen

The SymbolicConstant ALL or an Int specifying the number of eigenvalues to be calculated or ALL. The default value is ALL.

shift

A Float specifying the shift point in cycles per time. The default value is 0.0.

minEigen

`None` or a Float specifying the minimum frequency of interest in cycles per time. The default value is `None`.

maxEigen

`None` or a Float specifying the maximum frequency of interest in cycles per time. The default value is `None`.

vectors

`None` or an Int specifying the number of vectors used in the iteration. The default is the minimum of $(2n, n + 8)$, where n is the number of eigenvalues requested. The default value is `None`.

maxIterations

An Int specifying the maximum number of iterations. The default value is 30.

blockSize

A SymbolicConstant specifying the size of the Lanczos block steps. The default value is DEFAULT.

maxBlocks

A SymbolicConstant specifying the maximum number of Lanczos block steps within each Lanczos run. The default value is DEFAULT.

normalization

A SymbolicConstant specifying the method for normalizing eigenvectors. Possible values are DISPLACEMENT and MASS. The default value is DISPLACEMENT.

A value of DISPLACEMENT indicates normalizing the eigenvectors so that the largest displacement entry in each vector is unity. A value of MASS indicates

normalizing the eigenvectors with respect to the structure's mass matrix, which results in scaling the eigenvectors so that the generalized mass for each vector is unity.

propertyEvaluationFrequency

`None` or a Float specifying the frequency at which to evaluate frequency-dependent properties for viscoelasticity, springs, and dashpots during the eigenvalue extraction. If the value is `None`, the analysis product will evaluate the stiffness associated with frequency-dependent springs and dashpots at zero frequency and will not consider the stiffness contributions from frequency-domain viscoelasticity in the step. The default value is `None`.

projectDamping

A Boolean specifying whether to include projection of viscous and structural damping operators during *AMS* eigenvalue extraction. Valid only when *eigenSolver*=AMS. The default value is ON.

acousticCoupling

A SymbolicConstant specifying the type of acoustic-structural coupling in models with acoustic and structural elements coupled using the **TIE* option or in models with ASI-type elements. Possible values are AC_ON, AC_OFF, and AC_PROJECTION. The default value is AC_ON.

acousticRangeFactor

A Float specifying the ratio of the maximum acoustic frequency to the maximum structural frequency. The default value is 1.0.

frictionDamping

A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

simLinearDynamics

A Boolean specifying whether to activate the SIM-based linear dynamics procedures. The default value is OFF.

residualModes

A Boolean specifying whether to include residual modes from an immediately preceding Static, Linear Perturbation step. The default value is OFF.

substructureCutoffMultiplier

A Float specifying the cutoff frequency for substructure eigenproblems, defined as a multiplier of the maximum frequency of interest. The default value is 5.0.

firstCutoffMultiplier

A Float specifying the first cutoff frequency for a reduced eigenproblem, defined as a multiplier of the maximum frequency of interest. The default value is 1.7.

secondCutoffMultiplier

A Float specifying the second cutoff frequency for a reduced eigenproblem defined as a multiplier of the maximum frequency of interest. The default value is 1.1.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

residualModeRegion

`None` or a tuple of Strings specifying the name of a region for which residual modes are requested. The default value is `None`.

residualModeDof

`None` or a tuple of Ints specifying the degree of freedom for which residual modes are requested. The default value is `None`.

limitSavedEigenvectorRegion

`None` or a *Region* object specifying a region for which eigenvectors should be saved or the SymbolicConstant None representing the whole model. The default value is `None`.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT

- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **FREQUENCY*
- **STEP*

GeostaticStep object

The GeostaticStep object is used to verify that the geostatic stress field is in equilibrium with the applied loads and boundary conditions on the model and to iterate, if needed, to obtain equilibrium.

The GeostaticStep object is derived from the *AnalysisStep* object.

Access

```
import
step mdb.models[name].steps[name]
```

GeostaticStep(...)

This method creates a GeostaticStep object.

Path

```
mdb.models[name].GeostaticStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

nlgeom
A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

matrixSolver
A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique
A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel
An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

utol

`None` or a Float specifying the tolerance for maximum change of displacements. The default value is `None`.

timePeriod

A Float specifying the total time period. The default value is 1.0.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

Return value

A GeostaticStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the GeostaticStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *GeostaticStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The GeostaticStep object can have the following members:

name

A String specifying the repository key.

nlgeom

A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

utol

`None` or a Float specifying the tolerance for maximum change of displacements. The default value is `None`.

timePeriod

A Float specifying the total time period. The default value is 1.0.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.



Note:

This parameter is ignored unless *timeIncrementationMethod*=AUTOMATIC.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL

- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **GEOSTATIC*
- **STEP*

HeatTransferStep object

The HeatTransferStep object is used to control uncoupled heat transfer for either transient or steady-state response.

The HeatTransferStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

HeatTransferStep(...)

This method creates a HeatTransferStep object.

Path

```
mdb.models[name].HeatTransferStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period. The default value is 1.0.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc A Float specifying the initial time increment. The default value is the total time period for the step.

minInc A Float specifying the minimum time increment allowed. The default value is the smaller of 0.8 times the initial time increment or 10^{-5} times the total time period.

maxInc A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end

`None` or a Float specifying the temperature change rate (temperature per time) used to define steady state. When all nodal temperatures are changing at less than this rate, the solution terminates. The default value is `None`.



Note:

This parameter is ignored unless *response*=STEADY_STATE.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment during a transient heat transfer analysis. The default value is 0.0.

mxdem

A Float specifying the maximum allowable emissivity change with temperature and field variables during an increment. The default value is 0.1.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A HeatTransferStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the HeatTransferStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HeatTransferStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The HeatTransferStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period. The default value is 1.0.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of 0.8 times the initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end
`None` or a Float specifying the temperature change rate (temperature per time) used to define steady state. When all nodal temperatures are changing at less than this rate, the solution terminates. The default value is `None`.



Note:

This parameter is ignored unless *response*=STEADY_STATE.

deltmx

A Float specifying the maximum temperature change to be allowed in an increment during a transient heat transfer analysis. The default value is 0.0.

mxdem

A Float specifying the maximum allowable emissivity change with temperature and field variables during an increment. The default value is 0.1.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed

A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState

A repository of *FieldOutputRequestState* objects.

historyOutputRequestState

A repository of *HistoryOutputRequestState* objects.

diagnosticPrint

A *DiagnosticPrint* object.

monitor

A *Monitor* object.

restart

A *Restart* object.

adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **HEAT TRANSFER*
- **STEP*

ImplicitDynamicsStep object

The ImplicitDynamicsStep object is used to provide direct integration of a dynamic stress/displacement response in Abaqus/Standard analyses and is generally used for nonlinear cases.

The ImplicitDynamicsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ImplicitDynamicsStep(...)

This method creates an ImplicitDynamicsStep object.

Path

```
mdb.models[name].ImplicitDynamicsStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

timePeriod
A Float specifying the total time period of the step. The default value is 1.0.

nlgeom
A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is based on the previous step.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

application
A SymbolicConstant specifying the application type of the step. Possible values are ANALYSIS_PRODUCT_DEFAULT, TRANSIENT_FIDELITY, MODERATE_DISSIPATION, and QUASI_STATIC. The default value is ANALYSIS_PRODUCT_DEFAULT.

adiabatic
A Boolean specifying whether an adiabatic stress analysis is to be performed. The default value is OFF.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

The SymbolicConstant DEFAULT or a Float specifying the maximum time increment allowed.

hafTolMethod

A SymbolicConstant specifying the way of specifying half-increment residual tolerance with the automatic time incrementation scheme. Possible values are ANALYSIS_PRODUCT_DEFAULT, VALUE, and SCALE. The default value is VALUE.

haftol

`None` or a Float specifying the half-increment residual tolerance to be used with the automatic time incrementation scheme. The default value is `None`.

halfIncScaleFactor

`None` or a Float specifying the half-increment residual tolerance scale factor to be used with the automatic time incrementation scheme. The default value is `None`.

nohaf

A Boolean specifying whether to suppress calculation of the half-increment residual. The default value is OFF.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

alpha

The SymbolicConstant DEFAULT or a Float specifying the nondefault value of the numerical (artificial) damping control parameter, α, in the implicit operator. Possible values are $-.333 < \alpha < 0$. The default value is DEFAULT.

initialConditions

A SymbolicConstant specifying whether accelerations should be calculated or recalculated at the beginning of the step. Possible values are DEFAULT, BYPASS, and ALLOW. The default value is DEFAULT.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, PARABOLIC, VELOCITY_PARABOLIC, and ANALYSIS_PRODUCT_DEFAULT. The default value is ANALYSIS_PRODUCT_DEFAULT.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed have been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=OFF only in special cases when you have a thorough understanding of how to interpret the results.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

An ImplicitDynamicsStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ImplicitDynamicsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ImplicitDynamicsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ImplicitDynamicsStep object can have the following members:

name A String specifying the repository key.

timePeriod A Float specifying the total time period of the step. The default value is 1.0.

nlgeom

A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is based on the previous step.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

application

A SymbolicConstant specifying the application type of the step. Possible values are ANALYSIS_PRODUCT_DEFAULT, TRANSIENT_FIDELITY, MODERATE_DISSIPATION, and QUASI_STATIC. The default value is ANALYSIS_PRODUCT_DEFAULT.

adiabatic

A Boolean specifying whether an adiabatic stress analysis is to be performed. The default value is OFF.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

The SymbolicConstant DEFAULT or a Float specifying the maximum time increment allowed.

hafTolMethod

A SymbolicConstant specifying the way of specifying half-increment residual tolerance with the automatic time incrementation scheme. Possible values are ANALYSIS_PRODUCT_DEFAULT, VALUE, and SCALE. The default value is VALUE.

haftol

`None` or a Float specifying the half-increment residual tolerance to be used with the automatic time incrementation scheme. The default value is `None`.

halfIncScaleFactor

`None` or a Float specifying the half-increment residual tolerance scale factor to be used with the automatic time incrementation scheme. The default value is `None`.

nohaf

A Boolean specifying whether to suppress calculation of the half-increment residual. The default value is OFF.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

alpha

The SymbolicConstant DEFAULT or a Float specifying the nondefault value of the numerical (artificial) damping control parameter, α, in the implicit operator. Possible values are $-.333 < \alpha < 0$. The default value is DEFAULT.

initialConditions

A SymbolicConstant specifying whether accelerations should be calculated or recalculated at the beginning of the step. Possible values are DEFAULT, BYPASS, and ALLOW. The default value is DEFAULT.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, PARABOLIC, VELOCITY_PARABOLIC, and ANALYSIS_PRODUCT_DEFAULT. The default value is ANALYSIS_PRODUCT_DEFAULT.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed have been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=OFF only in special cases when you have a thorough understanding of how to interpret the results.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DYNAMIC*
- **STEP*

InitialStep object

The InitialStep object is a placeholder that you cannot create, delete, or modify. The InitialStep object exists in every model by default and is used to allow the preexisting boundary conditions and interactions to be defined in the model.

The InitialStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

Members

The InitialStep object can have the following members:

name

A String specifying the repository key.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM

- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates
A repository of *PredefinedFieldState* objects.

MassDiffusionStep object

The MassDiffusionStep object is used to control uncoupled transient or steady-state mass diffusion analysis.

The MassDiffusionStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

MassDiffusionStep(...)

This method creates a MassDiffusionStep object.

Path

```
mdb.models[name].MassDiffusionStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

response A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod A Float specifying the total time period. The default value is 1.0.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc A Float specifying the initial time increment. The default value is the total time period for the step.

minInc A Float specifying the minimum time increment allowed. The default value is the smaller of 0.8 times the initial time increment or 10^{-5} times the total time period.

maxInc A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end
A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

dcmax
A Float specifying the maximum normalized concentration change to be allowed in an increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A MassDiffusionStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the MassDiffusionStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *MassDiffusionStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The MassDiffusionStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period. The default value is 1.0.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of 0.8 times the initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

end
A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

dcmax
A Float specifying the maximum normalized concentration change to be allowed in an increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description
A String specifying a description of the new step. The default value is an empty string.

explicit
A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation
A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical
A Boolean specifying whether the step has a mechanical procedure type.

procedureType
A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **MASS DIFFUSION*
- **STEP*

ModalDynamicsStep object

The ModalDynamicsStep object is used to provide dynamic time history response as a linear perturbation procedure using modal superposition.

The ModalDynamicsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ModalDynamicsStep(...)

This method creates a ModalDynamicsStep object.

Path

```
mdb.models[name].ModalDynamicsStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

continueAnalysis
A Boolean specifying that the step starts with zero initial conditions. The default value is OFF.

timePeriod
A Float specifying the total time period. The default value is 1.0.

incSize
A Float specifying the time increment to be used. The default value is 1.0.

directDamping
A *DirectDamping* object.

compositeDamping
A *CompositeDamping* object.

rayleighDamping
A *RayleighDamping* object.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

directDampingByFrequency A *DirectDampingByFrequency* object.

rayleighDampingByFrequency A *RayleighDampingByFrequency* object.

Return value

A ModalDynamicsStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ModalDynamicsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ModalDynamicsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ModalDynamicsStep object can have the following members:

name A String specifying the repository key.

continueAnalysis A Boolean specifying that the step starts with zero initial conditions. The default value is OFF.

timePeriod A Float specifying the total time period. The default value is 1.0.

incSize A Float specifying the time increment to be used. The default value is 1.0.

amplitude A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

directDamping

A *DirectDamping* object.

compositeDamping

A *CompositeDamping* object.

rayleighDamping

A *RayleighDamping* object.

directDampingByFrequency

A *DirectDampingByFrequency* object.

rayleighDampingByFrequency

A *RayleighDampingByFrequency* object.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM

- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates
A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DAMPING*
- **MODAL DAMPING*
- **MODAL DYNAMIC*

- **STEP*

RandomResponseStep object

The RandomResponseStep object is used to give the linearized response of a model to random excitation.

The RandomResponseStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

RandomResponseStep(...)

This method creates a RandomResponseStep object.

Path

```
mdb.models[name].RandomResponseStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

freq A *RandomResponseFrequencyArray* object specifying frequencies over ranges of modes.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

scale A SymbolicConstant specifying the frequency scale. Possible values are LINEAR and LOG. The default value is LOG.

directDamping A *DirectDamping* object.

compositeDamping A *CompositeDamping* object.

rayleighDamping A *RayleighDamping* object.

structuralDamping A *StructuralDamping* object.

directDampingByFrequency A *DirectDampingByFrequency* object.

rayleighDampingByFrequency A *RayleighDampingByFrequency* object.

structuralDampingByFrequency A *StructuralDampingByFrequency* object.

maintainAttributes A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

Return value

A RandomResponseStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the RandomResponseStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *RandomResponseStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The RandomResponseStep object can have the following members:

name A String specifying the repository key.

scale A SymbolicConstant specifying the frequency scale. Possible values are LINEAR and LOG. The default value is LOG.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description A String specifying a description of the new step. The default value is an empty string.

freq A *RandomResponseFrequencyArray* object specifying frequencies over ranges of modes.

directDamping A *DirectDamping* object.

compositeDamping A *CompositeDamping* object.

rayleighDamping A *RayleighDamping* object.

structuralDamping A *StructuralDamping* object.

directDampingByFrequency A *DirectDampingByFrequency* object.

rayleighDampingByFrequency A *RayleighDampingByFrequency* object.

structuralDampingByFrequency A *StructuralDampingByFrequency* object.

explicit A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical A Boolean specifying whether the step has a mechanical procedure type.

procedureType A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION

- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates
A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DAMPING*
- **MODAL DAMPING*
- **RANDOM RESPONSE*
- **STEP*

ResponseSpectrumStep object

The ResponseSpectrumStep object is used to calculate estimates of peak values of displacements and stresses based on user-supplied response spectra and on the natural modes of the system.

The ResponseSpectrumStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ResponseSpectrumStep(...)

This method creates a ResponseSpectrumStep object.

Path

```
mdb.models[name].ResponseSpectrumStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

components
A *ResponseSpectrumComponentArray* object.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

comp
A SymbolicConstant specifying the order and method used to sum the components. Possible values are SINGLE_DIRECTION, MULTIPLE_DIRECTION_ABSOLUTE_SUM, MULTIPLE_DIRECTION_SRSS_SUM, MULTIPLE_DIRECTION_THIRTY_PERCENT_RULE, and MULTIPLE_DIRECTION_FORTY_PERCENT_RULE. The default value is SINGLE_DIRECTION.

sum
A SymbolicConstant specifying the method used to sum the components. Possible values are ABS, CQC, NRL, SRSS, TENP, DSC, and GRP. The default value is ABS.

directDamping
A *DirectDamping* object.

compositeDamping
A *CompositeDamping* object.

rayleighDamping	A *RayleighDamping* object.
directDampingByFrequency	A *DirectDampingByFrequency* object.
rayleighDampingByFrequency	A *RayleighDampingByFrequency* object.
maintainAttributes	A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

Return value

A ResponseSpectrumStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ResponseSpectrumStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ResponseSpectrumStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ResponseSpectrumStep object can have the following members:

name	A String specifying the repository key.
comp	A SymbolicConstant specifying the order and method used to sum the components. Possible values are SINGLE_DIRECTION, MULTIPLE_DIRECTION_ABSOLUTE_SUM, MULTIPLE_DIRECTION_SRSS_SUM, MULTIPLE_DIRECTION_THIRTY_PERCENT_RULE, and MULTIPLE_DIRECTION_FORTY_PERCENT_RULE. The default value is SINGLE_DIRECTION.

sum
A SymbolicConstant specifying the method used to sum the components. Possible values are ABS, CQC, NRL, SRSS, TENP, DSC, and GRP. The default value is ABS.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description
A String specifying a description of the new step. The default value is an empty string.

components
A *ResponseSpectrumComponentArray* object.

directDamping
A *DirectDamping* object.

compositeDamping
A *CompositeDamping* object.

rayleighDamping
A *RayleighDamping* object.

directDampingByFrequency
A *DirectDampingByFrequency* object.

rayleighDampingByFrequency
A *RayleighDampingByFrequency* object.

structuralDamping
A *StructuralDamping* object.

structuralDampingByFrequency
A *StructuralDampingByFrequency* object.

explicit
A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation
A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical
A Boolean specifying whether the step has a mechanical procedure type.

procedureType
A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT

- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **RESPONSE SPECTRUM*
- **STEP*

SoilsStep object

The SoilsStep object is used to specify transient (consolidation) or steady-state response analysis of partially or fully saturated fluid-filled porous media.

The SoilsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SoilsStep(...)

This method creates a SoilsStep object.

Path

```
mdb.models[name].SoilsStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period. The default value is 1.0.

nlgeom
A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod
A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude
A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

creep
A Boolean specifying whether a creep response occurs during this step. The default value is ON.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

end
A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

utol
`None` or a Float specifying the maximum pore pressure change permitted in any increment (in pressure units) in a transient consolidation analysis. The default value is `None`.

cetol
A Float specifying the maximum allowable difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

matrixSolver
A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique
A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel
An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio
A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors
A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

Return value

A SoilsStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SoilsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SoilsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The SoilsStep object can have the following members:

name
A String specifying the repository key.

response
A SymbolicConstant specifying the analysis type. Possible values are STEADY_STATE and TRANSIENT. The default value is TRANSIENT.

timePeriod
A Float specifying the total time period. The default value is 1.0.

nlgeom

A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

stabilizationMethod

A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude

A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

creep

A Boolean specifying whether a creep response occurs during this step. The default value is ON.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

end

A SymbolicConstant specifying the time period to be analyzed in a transient analysis. Possible values are PERIOD and SS. The default value is PERIOD.

utol

`None` or a Float specifying the maximum pore pressure change permitted in any increment (in pressure units) in a transient consolidation analysis. The default value is `None`.

cetol

A Float specifying the maximum allowable difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. The default value is STEP. Possible values are STEP and RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT

- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **SOILS*
- **STEP*

StaticLinearPerturbationStep object

The StaticLinearPerturbationStep object is used to indicate that the static step should be analyzed as a linear perturbation load step.

The StaticLinearPerturbationStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

StaticLinearPerturbationStep(...)

This method creates a StaticLinearPerturbationStep object.

Path

```
mdb.models[name].StaticLinearPerturbationStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

matrixSolver
A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique
A SymbolicConstant specifying the technique used for solving nonlinear equations. Possible values are FULL_NEWTON and LCP_CONTACT.

gapDistance
A Float specifying the zone within which potential contact constraints are exposed to the LCP solver, applicable when solutionTechnique is LCP_CONTACT.

scaleFactor
A Float specifying the scaling factor for the initial gap distance specified by gapDistance or the internally computed (contact) pair-wise gap distances, applicable when solutionTechnique is LCP_CONTACT.

Return value

A StaticLinearPerturbationStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the StaticLinearPerturbationStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StaticLinearPerturbationStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

```
None.
```

Exceptions

RangeError.

Members

The StaticLinearPerturbationStep object can have the following members:

name

A String specifying the repository key.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

solutionTechnique

A SymbolicConstant specifying the technique used for solving nonlinear equations. Possible values are FULL_NEWTON and LCP_CONTACT.

gapDistance

A Float specifying the zone within which potential contact constraints are exposed to the LCP solver, applicable when solutionTechnique is LCP_CONTACT.

scaleFactor

A Float specifying the scaling factor for the initial gap distance specified by gapDistance or the internally computed (contact) pair-wise gap distances, applicable when solutionTechnique is LCP_CONTACT.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed	A Boolean specifying whether the step is suppressed or not. The default value is OFF.
fieldOutputRequestState	A repository of *FieldOutputRequestState* objects.
historyOutputRequestState	A repository of *HistoryOutputRequestState* objects.
diagnosticPrint	A *DiagnosticPrint* object.
monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **STATIC*
- **STEP*

StaticRiksStep object

The StaticRiksStep object is used to indicate that the step should be analyzed as a static load step using the modified Riks method for proportional loading cases.

The StaticRiksStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

StaticRiksStep(...)

This method creates a StaticRiksStep object.

Path

```
mdb.models[name].StaticRiksStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

nlgeom A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

adiabatic A Boolean specifying whether to perform an adiabatic stress analysis. The default value is OFF.

maxLPF `None` or a Float specifying the maximum value of the load proportionality factor. The default value is `None`.

nodeOn A Boolean specifying whether to monitor the finishing displacement value at a node. The default value is OFF.

maximumDisplacement A Float specifying the value of the total displacement (or rotation) at the node and degree of freedom that, if crossed during an increment, ends the step at the current increment. This argument is required when *nodeOn*=ON. The default value is 0.0.

dof An Int specifying the degree of freedom being monitored. This argument is required when *nodeOn*=ON. The default value is 0.

region

A *Region* object specifying the vertex at which the finishing displacement value is being monitored. This argument is required when *nodeOn*=ON.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

totalArcLength

A Float specifying the total load proportionality factor associated with the load in this step. The default value is 1.0.

initialArcInc

A Float specifying the initial load proportionality factor. The default value is the total load proportionality factor for the step.

minArcInc

A Float specifying the minimum arc length increment allowed. The default value is the smaller of the suggested initial load proportionality factor or 10^{-5} times the total load proportionality factor for the step.

maxArcInc

A Float specifying the maximum arc length increment allowed. The default value is the total load proportionality factor for the step.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

fullyPlastic

A String specifying the name of the region being monitored for fully plastic behavior. The default value is an empty string.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed have been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=ON only in special cases when you have a thorough understanding of how to interpret the results.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

useLongTermSolution

A Boolean specifying wether to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity. The default value is OFF.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

Return value

A StaticRiksStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the StaticRiksStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StaticRiksStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The StaticRiksStep object can have the following members:

name

A String specifying the repository key.

nlgeom

A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

adiabatic

A Boolean specifying whether to perform an adiabatic stress analysis. The default value is OFF.

maxLPF

`None` or a Float specifying the maximum value of the load proportionality factor. The default value is `None`.

nodeOn

A Boolean specifying whether to monitor the finishing displacement value at a node. The default value is OFF.

maximumDisplacement

A Float specifying the value of the total displacement (or rotation) at the node and degree of freedom that, if crossed during an increment, ends the step at the current increment. This argument is required when *nodeOn*=ON. The default value is 0.0.

dof

An Int specifying the degree of freedom being monitored. This argument is required when *nodeOn*=ON. The default value is 0.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

totalArcLength

A Float specifying the total load proportionality factor associated with the load in this step. The default value is 1.0.

initialArcInc

A Float specifying the initial load proportionality factor. The default value is the total load proportionality factor for the step.

minArcInc

A Float specifying the minimum arc length increment allowed. The default value is the smaller of the suggested initial load proportionality factor or 10^{-5} times the total load proportionality factor for the step.

maxArcInc

A Float specifying the maximum arc length increment allowed. The default value is the total load proportionality factor for the step.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed have been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=ON only in special cases when you have a thorough understanding of how to interpret the results.

useLongTermSolution

A Boolean specifying wether to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity. The default value is OFF.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED,

CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

fullyPlastic

A String specifying the name of the region being monitored for fully plastic behavior. The default value is an empty string.

region

A *Region* object specifying the vertex at which the finishing displacement value is being monitored. This argument is required when *nodeOn*=ON.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL

- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **STATIC*
- **STEP*

StaticStep object

The StaticStep object is used to indicate that the step should be analyzed as a static load step.

The StaticStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

StaticStep(...)

This method creates a StaticStep object.

Path

```
mdb.models[name].StaticStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

timePeriod
A Float specifying the total time period. The default value is 1.0.

nlgeom
A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

stabilizationMethod
A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude
A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

adiabatic
A Boolean specifying whether to perform an adiabatic stress analysis. The default value is OFF.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

fullyPlastic

A String specifying the region being monitored for fully plastic behavior. The default value is an empty string.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed has been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=ON only in special cases when you have a thorough understanding of how to interpret the results.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

useLongTermSolution

A Boolean specifying wether to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity. The default value is OFF.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

Return value

A StaticStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the StaticStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StaticStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The StaticStep object can have the following members:

name A String specifying the repository key.

timePeriod A Float specifying the total time period. The default value is 1.0.

nlgeom

A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

stabilizationMethod

A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude

A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

adiabatic

A Boolean specifying whether to perform an adiabatic stress analysis. The default value is OFF.

timeIncrementationMethod

A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

maxNumInc

An Int specifying the maximum number of increments in a step. The default value is 100.

initialInc

A Float specifying the initial time increment. The default value is the total time period for the step.

minInc

A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc

A Float specifying the maximum time increment allowed. The default value is the total time period for the step.

matrixSolver

A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

amplitude

A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is RAMP.

extrapolation

A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

noStop

A Boolean specifying whether to accept the solution to an increment after the maximum number of iterations allowed has been completed, even if the equilibrium tolerances are not satisfied. The default value is OFF.



Warning:

You should set *noStop*=ON only in special cases when you have a thorough understanding of how to interpret the results.

useLongTermSolution

A Boolean specifying wether to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity. The default value is OFF.

solutionTechnique

A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel

An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI

A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

fullyPlastic

A String specifying the region being monitored for fully plastic behavior. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE

- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **STATIC*
- **STEP*

SteadyStateDirectStep object

The SteadyStateDirectStep object is used to calculate the linearized steady-state response of the system to harmonic excitation in terms of the physical degrees of freedom of the model.

The SteadyStateDirectStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SteadyStateDirectStep(...)

This method creates a SteadyStateDirectStep object.

Path

```
mdb.models[name].SteadyStateDirectStep
```

Required arguments

name

A String specifying the repository key.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

frequencyRange

A *SteadyStateDirectFrequencyArray* object.

Optional arguments

description

A String specifying a description of the new step. The default value is an empty string.

factorization

A SymbolicConstant specifying whether damping terms are to be ignored so that a real, rather than a complex, system matrix is factored. Possible values are REAL_ONLY and COMPLEX. The default value is COMPLEX.

scale

A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

subdivideUsingEigenfrequencies A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is OFF.

frictionDamping A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

Return value

A SteadyStateDirectStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SteadyStateDirectStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SteadyStateDirectStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The SteadyStateDirectStep object can have the following members:

name A String specifying the repository key.

factorization A SymbolicConstant specifying whether damping terms are to be ignored so that a real, rather than a complex, system matrix is factored. Possible values are REAL_ONLY and COMPLEX. The default value is COMPLEX.

scale A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.

matrixStorage A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

subdivideUsingEigenfrequencies
A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is OFF.

frictionDamping
A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description
A String specifying a description of the new step. The default value is an empty string.

frequencyRange
A *SteadyStateDirectFrequencyArray* object.

explicit
A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation
A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical
A Boolean specifying whether the step has a mechanical procedure type.

procedureType
A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL

- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **STEADY STATE DYNAMICS*
- **STEP*

SteadyStateModalStep object

The SteadyStateModalStep object is used to calculate the linearized steady-state response of the system to harmonic excitation.

The SteadyStateModalStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SteadyStateModalStep(...)

This method creates a SteadyStateModalStep object.

Path

```
mdb.models[name].SteadyStateModalStep
```

Required arguments

name	A String specifying the repository key.
previous	A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.
frequencyRange	A *SteadyStateModalFrequencyArray* object.

Optional arguments

description	A String specifying a description of the new step. The default value is an empty string.
scale	A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.
directDamping	A *DirectDamping* object.
compositeDamping	A *CompositeDamping* object.
rayleighDamping	A *RayleighDamping* object.
structuralDamping	A *StructuralDamping* object.
directDampingByFrequency	A *DirectDampingByFrequency* object.
rayleighDampingByFrequency	A *RayleighDampingByFrequency* object.

structuralDampingByFrequency	A *StructuralDampingByFrequency* object.
maintainAttributes	A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.
subdivideUsingEigenfrequencies	A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is ON.

Return value

A SteadyStateModalStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SteadyStateModalStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SteadyStateModalStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The SteadyStateModalStep object can have the following members:

name	A String specifying the repository key.
scale	A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.
subdivideUsingEigenfrequencies	A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is ON.
previous	A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

frequencyRange

A *SteadyStateModalFrequencyArray* object.

directDamping

A *DirectDamping* object.

compositeDamping

A *CompositeDamping* object.

rayleighDamping

A *RayleighDamping* object.

structuralDamping

A *StructuralDamping* object.

directDampingByFrequency

A *DirectDampingByFrequency* object.

rayleighDampingByFrequency

A *RayleighDampingByFrequency* object.

structuralDampingByFrequency

A *StructuralDampingByFrequency* object.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION

- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control
A *Control* object.

solverControl
A *SolverControl* object.

boundaryConditionStates
A repository of *BoundaryConditionState* objects.

interactionStates
A repository of *InteractionState* objects.

loadStates
A repository of *LoadState* objects.

loadCases
A repository of *LoadCase* objects.

predefinedFieldStates
A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DAMPING*
- **MODAL DAMPING*
- **STEADY STATE DYNAMICS*
- **STEP*

SteadyStateSubspaceStep object

The SteadyStateSubspaceStep object is used to calculate the linearized steady-state response of the system to harmonic excitation on the basis of the subspace projection method.

The SteadyStateSubspaceStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SteadyStateSubspaceStep(...)

This method creates a SteadyStateSubspaceStep object.

Path

```
mdb.models[name].SteadyStateSubspaceStep
```

Required arguments

name

A String specifying the repository key.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

frequencyRange

A *SteadyStateSubspaceFrequencyArray* object.

Optional arguments

description

A String specifying a description of the new step. The default value is an empty string.

factorization

A SymbolicConstant specifying whether damping terms are to be ignored so that a real, rather than a complex, system matrix is factored. Possible values are REAL_ONLY and COMPLEX. The default value is COMPLEX.

scale

A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

maintainAttributes

A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

subdivideUsingEigenfrequencies A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is ON.

projection A SymbolicConstant specifying how often to perform subspace projections onto the modal subspace. Possible values are ALL_FREQUENCIES, CONSTANT, EIGENFREQUENCY, PROPERTY_CHANGE, and RANGE. The default value is ALL_FREQUENCIES.

maxDampingChange A Float specifying the maximum relative change in damping material properties before a new projection is to be performed. The default value is 0.1.

maxStiffnessChange A Float specifying the maximum relative change in stiffness material properties before a new projection is to be performed. The default value is 0.1.

frictionDamping A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

Return value

A SteadyStateSubspaceStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SteadyStateSubspaceStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SteadyStateSubspaceStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The SteadyStateSubspaceStep object can have the following members:

name A String specifying the repository key.

factorization

A SymbolicConstant specifying whether damping terms are to be ignored so that a real, rather than a complex, system matrix is factored. Possible values are REAL_ONLY and COMPLEX. The default value is COMPLEX.

scale

A SymbolicConstant specifying whether a logarithmic or linear scale is used for output. Possible values are LOGARITHMIC and LINEAR. The default value is LOGARITHMIC.

matrixStorage

A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

subdivideUsingEigenfrequencies

A Boolean specifying whether to subdivide each frequency range using the eigenfrequencies of the system. The default value is ON.

projection

A SymbolicConstant specifying how often to perform subspace projections onto the modal subspace. Possible values are ALL_FREQUENCIES, CONSTANT, EIGENFREQUENCY, PROPERTY_CHANGE, and RANGE. The default value is ALL_FREQUENCIES.

maxDampingChange

A Float specifying the maximum relative change in damping material properties before a new projection is to be performed. The default value is 0.1.

maxStiffnessChange

A Float specifying the maximum relative change in stiffness material properties before a new projection is to be performed. The default value is 0.1.

frictionDamping

A Boolean specifying whether to add to the damping matrix contributions due to friction effects. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

frequencyRange

A *SteadyStateSubspaceFrequencyArray* object.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE

- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed
A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState
A repository of *FieldOutputRequestState* objects.

historyOutputRequestState
A repository of *HistoryOutputRequestState* objects.

diagnosticPrint
A *DiagnosticPrint* object.

monitor
A *Monitor* object.

restart
A *Restart* object.

adaptiveMeshConstraintStates
A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains
A repository of *AdaptiveMeshDomain* objects.

control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **STEADY STATE DYNAMICS*
- **STEP*

SubspaceDynamicsStep object

The SubspaceDynamicsStep object is used to calculate the linearized steady-state response of the system to harmonic excitation on the basis of the subspace projection method.

The SubspaceDynamicsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SubspaceDynamicsStep(...)

This method creates a SubspaceDynamicsStep object.

Path

```
mdb.models[name].SubspaceDynamicsStep
```

Required arguments

name
A String specifying the repository key.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description
A String specifying a description of the new step. The default value is an empty string.

timePeriod
A Float specifying the total time period of the step. The default value is 1.0.

vectors
The SymbolicConstant ALL or an Int specifying the number of modes to be used for subspace projection. The possible value for the SymbolicConstant is ALL. The default value is ALL.

nlgeom
A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

incSize
A Float specifying the suggested time increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

Return value

A SubspaceDynamicsStep object.

Exceptions

RangeError.

setValues(...)

This method modifies the SubspaceDynamicsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SubspaceDynamicsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The SubspaceDynamicsStep object can have the following members:

name
A String specifying the repository key.

timePeriod
A Float specifying the total time period of the step. The default value is 1.0.

vectors
The SymbolicConstant ALL or an Int specifying the number of modes to be used for subspace projection. The possible value for the SymbolicConstant is ALL. The default value is ALL.

nlgeom
A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

incSize
A Float specifying the suggested time increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed	A Boolean specifying whether the step is suppressed or not. The default value is OFF.
fieldOutputRequestState	A repository of *FieldOutputRequestState* objects.
historyOutputRequestState	A repository of *HistoryOutputRequestState* objects.
diagnosticPrint	A *DiagnosticPrint* object.
monitor	A *Monitor* object.
restart	A *Restart* object.
adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **DYNAMIC*
- **STEP*

SubstructureGenerateStep object

TheSubstructureGenerateStep object is used to generate a substructure.

The SubstructureGenerateStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

SubstructureGenerateStep(...)

This method creates a SubstructureGenerateStep object.

Path

```
mdb.models[name].SubstructureGenerateStep
```

Required arguments

name	A String specifying the repository key.
previous	A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.
substructureIdentifier	An Integer specifying a unique identifier for the substructure.

Optional arguments

description	A String specifying a description of the new step. The default value is an empty string.
recoveryMatrix	A SymbolicConstant specifying the subtructure recovery to be computed. Possible values are WHOLE_MODEL, REGION, and NONE. The default value is WHOLE_MODEL.
recoveryRegion	A *Region* object specifying the region for substructure recovery. This argument is required when *recoveryMatrix*=REGION.
computeGravityLoadVectors	A Boolean specifying whether to compute the gravity load vectors. The default value is False.
computeReducedMassMatrix	A Boolean specifying whether to compute the reduced mass matrix. The default value is False.
computeReducedStructuralDampingMatrix	A Boolean specifying whether to compute the reduced structural damping matrix. The default value is False.

computeReducedViscousDampingMatrix A Boolean specifying whether to compute the reduced viscous damping matrix. The default value is False.

evaluateFrequencyDependentProperties A Boolean specifying whether to evaluate the frequency dependent properties. The default value is False.

frequency A Float specifying the frequency at which to evaluate the frequency dependent properties. The default value is 0.0.

retainedEigenmodesMethod A SymbolicConstant specifying the eigenmodes to be retained. Possible values are MODE_RANGE, FREQUENCY_RANGE, and NONE. The default value is NONE.

modeRange A *SubstructureGenerateModesArray* object.

frequencyRange A *SubstructureGenerateFrequencyArray* object.

globalDampingField A SymbolicConstant specifying the field to which the global damping factors should be applied. Possible values are ALL, ACOUSTIC, MECHANICAL, and NONE. The default value is NONE.

alphaDampingRatio A Float specifying the factor to create global Rayleigh mass proportional damping. The default value is 0.0.

betaDampingRatio A Float specifying the factor to create global Rayleigh stiffness proportional damping. The default value is 0.0.

structuralDampingRatio A Float specifying the factor to create frequency-independent stiffness proportional structural damping. The default value is 0.0.

viscousDampingControl A SymbolicConstant specifying the damping control to include the viscous damping matrix. Possible values are ELEMENT, FACTOR, COMBINED, and NONE. The default value is NONE.

structuralDampingControl A SymbolicConstant specifying the damping control to include the structural damping matrix. Possible values are ELEMENT, FACTOR, COMBINED, and NONE. The default value is NONE.

Return value

A SubstructureGenerateStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the SubstructureGenerateStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SubstructureGenerateStep* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SubstructureGenerateStep object can have the following members:

name
A String specifying the repository key.

recoveryMatrix
A SymbolicConstant specifying the subtructure recovery to be computed. Possible values are WHOLE_MODEL, REGION, and NONE. The default value is WHOLE_MODEL.

frequency
A Float specifying the frequency at which to evaluate the frequency dependent properties. The default value is 0.0.

retainedEigenmodesMethod
A SymbolicConstant specifying the eigenmodes to be retained. Possible values are MODE_RANGE, FREQUENCY_RANGE, and NONE. The default value is NONE.

globalDampingField
A SymbolicConstant specifying the field to which the global damping factors should be applied. Possible values are ALL, ACOUSTIC, MECHANICAL, and NONE. The default value is NONE.

alphaDampingRatio
A Float specifying the factor to create global Rayleigh mass proportional damping. The default value is 0.0.

betaDampingRatio
A Float specifying the factor to create global Rayleigh stiffness proportional damping. The default value is 0.0.

structuralDampingRatio
A Float specifying the factor to create frequency-independent stiffness proportional structural damping. The default value is 0.0.

viscousDampingControl
A SymbolicConstant specifying the damping control to include the viscous damping matrix. Possible values are ELEMENT, FACTOR, COMBINED, and NONE. The default value is NONE.

structuralDampingControl

A SymbolicConstant specifying the damping control to include the structural damping matrix. Possible values are ELEMENT, FACTOR, COMBINED, and NONE. The default value is NONE.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

substructureIdentifier

A String specifying a unique identifier for the substructure. The default value is an empty string.

recoveryRegion

A *Region* object specifying the region for substructure recovery. This argument is required when *recoveryMatrix*=REGION.

frequencyRange

A *SubstructureGenerateFrequencyArray* object.

modeRange

A *SubstructureGenerateModesArray* object.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION

- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **SUBSTRUCTURE GENERATE*
- **STEP*

TempDisplacementDynamicsStep object

The TempDisplacementDynamicsStep object is used to perform a dynamic coupled thermal-stress analysis using explicit integration.

The TempDisplacementDynamicsStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

TempDisplacementDynamicsStep(...)

This method creates a TempDisplacementDynamicsStep object.

Path

```
mdb.models[name].TempDisplacementDynamicsStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

timePeriod A Float specifying the time period of the step. The default value is 1.0.

nlgeom A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are AUTOMATIC_GLOBAL, AUTOMATIC_EBE, FIXED_USER_DEFINED_INC, and FIXED_EBE. The default value is AUTOMATIC_GLOBAL.

maxIncrement `None` or a Float specifying the maximum time increment allowed. If there is no upper limit, *maxIncrement*=`None`. The default value is `None`.

scaleFactor A Float specifying the factor that is used to scale the time increment. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL, AUTOMATIC_EBE, or FIXED_EBE. The default value is 1.0.

userDefinedInc `None` or a Float specifying the user-defined time increment. The default value is `None`.

massScaling
A tuple of tuples, where each tuple specifies the *MassScaling Object* controls. The default value is PREVIOUS_STEP.

linearBulkViscosity
A Float specifying the linear bulk viscosity parameter, b_1. The default value is 0.06.

quadBulkViscosity
A Float specifying the quadratic bulk viscosity parameter, b_2. The default value is 1.2.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

improvedDtMethod
A Boolean specifying whether to use the "improved" (*improvedDtMethod*=ON) or "conservative" (*improvedDtMethod*=OFF) method to estimate the element stable time increment for three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements). The default value is ON.

Return value

A TempDisplacementDynamicsStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the TempDisplacementDynamicsStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TempDisplacementDynamicsStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The TempDisplacementDynamicsStep object can have the following members:

name
A String specifying the repository key.

timePeriod
A Float specifying the time period of the step. The default value is 1.0.

nlgeom
A Boolean specifying whether geometric nonlinearities should be accounted for during the step. The default value is OFF.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are AUTOMATIC_GLOBAL, AUTOMATIC_EBE, FIXED_USER_DEFINED_INC, and FIXED_EBE. The default value is AUTOMATIC_GLOBAL.

maxIncrement
`None` or a Float specifying the maximum time increment allowed. If there is no upper limit, *maxIncrement*=`None`. The default value is `None`.

scaleFactor
A Float specifying the factor that is used to scale the time increment. This argument is required only when *timeIncrementationMethod*=AUTOMATIC_GLOBAL, AUTOMATIC_EBE, or FIXED_EBE. The default value is 1.0.

userDefinedInc
`None` or a Float specifying the user-defined time increment. The default value is `None`.

linearBulkViscosity
A Float specifying the linear bulk viscosity parameter, b_1. The default value is 0.06.

quadBulkViscosity
A Float specifying the quadratic bulk viscosity parameter, b_2. The default value is 1.2.

improvedDtMethod
A Boolean specifying whether to use the "improved" (*improvedDtMethod*=ON) or "conservative" (*improvedDtMethod*=OFF) method to estimate the element stable time increment for three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements). The default value is ON.

previous
A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description
A String specifying a description of the new step. The default value is an empty string.

massScaling
A *MassScalingArray* object specifying mass scaling controls. The default value is PREVIOUS_STEP.

explicit
A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation
A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical
A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION
- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed

A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState

A repository of *FieldOutputRequestState* objects.

historyOutputRequestState

A repository of *HistoryOutputRequestState* objects.

diagnosticPrint

A *DiagnosticPrint* object.

monitor

A *Monitor* object.

restart

A *Restart* object.

adaptiveMeshConstraintStates	A repository of *AdaptiveMeshConstraintState* objects.
adaptiveMeshDomains	A repository of *AdaptiveMeshDomain* objects.
control	A *Control* object.
solverControl	A *SolverControl* object.
boundaryConditionStates	A repository of *BoundaryConditionState* objects.
interactionStates	A repository of *InteractionState* objects.
loadStates	A repository of *LoadState* objects.
loadCases	A repository of *LoadCase* objects.
predefinedFieldStates	A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **BULK VISCOSITY*
- **DYNAMIC*
- **FIXED MASS SCALING*
- **STEP*
- **VARIABLE MASS SCALING*

ViscoStep object

The ViscoStep object is used to obtain a transient static response in an analysis with time-dependent material behavior (creep, swelling, and viscoelasticity).

The ViscoStep object is derived from the *AnalysisStep* object.

Access

```
import step
mdb.models[name].steps[name]
```

ViscoStep(...)

This method creates a ViscoStep object.

Path

```
mdb.models[name].ViscoStep
```

Required arguments

name A String specifying the repository key.

previous A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

Optional arguments

description A String specifying a description of the new step. The default value is an empty string.

timePeriod A Float specifying the total time period. The default value is 1.0.

nlgeom A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

stabilizationMethod A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

timeIncrementationMethod A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

matrixSolver A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default is the total time period for the step. The default value is 1.0.

integration
A SymbolicConstant specifying which type of integration to use throughout the step. Possible values are IMPLICIT_EXPLICIT and EXPLICIT_ONLY. The default value is IMPLICIT_EXPLICIT.

cetol
A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

maintainAttributes
A Boolean specifying whether to retain attributes from an existing step with the same name. The default value is False.

solutionTechnique
A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel
An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio
A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

Return value

A ViscoStep object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ViscoStep object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ViscoStep* method, except for the *name*, *previous*, and *maintainAttributes* arguments.

Return value

`None.`

Exceptions

RangeError.

Members

The ViscoStep object can have the following members:

name A String specifying the repository key.

timePeriod A Float specifying the total time period. The default value is 1.0.

nlgeom A Boolean specifying whether to allow for geometric nonlinearity. The default value is OFF.

stabilizationMethod A SymbolicConstant specifying the stabilization type. Possible values are NONE, DISSIPATED_ENERGY_FRACTION, and DAMPING_FACTOR. The default value is NONE.

stabilizationMagnitude A Float specifying the damping intensity of the automatic damping algorithm if the problem is expected to be unstable, and *stabilizationMethod* is not NONE. The default value is 2×10^{-4}.

timeIncrementationMethod
A SymbolicConstant specifying the time incrementation method to be used. Possible values are FIXED and AUTOMATIC. The default value is AUTOMATIC.

matrixSolver
A SymbolicConstant specifying the type of solver. Possible values are DIRECT and ITERATIVE. The default value is DIRECT.

matrixStorage
A SymbolicConstant specifying the type of matrix storage. Possible values are SYMMETRIC, UNSYMMETRIC, and SOLVER_DEFAULT. The default value is SOLVER_DEFAULT.

initialInc
A Float specifying the initial time increment. The default value is the total time period for the step.

maxNumInc
An Int specifying the maximum number of increments in a step. The default value is 100.

minInc
A Float specifying the minimum time increment allowed. The default value is the smaller of the suggested initial time increment or 10^{-5} times the total time period.

maxInc
A Float specifying the maximum time increment allowed. The default is the total time period for the step. The default value is 1.0.

integration
A SymbolicConstant specifying which type of integration to use throughout the step. Possible values are IMPLICIT_EXPLICIT and EXPLICIT_ONLY. The default value is IMPLICIT_EXPLICIT.

cetol
A Float specifying the maximum difference in the creep strain increment calculated from the creep strain rates at the beginning and end of the increment. The default value is 0.0.

amplitude
A SymbolicConstant specifying the amplitude variation for loading magnitudes during the step. Possible values are STEP and RAMP. The default value is STEP.

extrapolation
A SymbolicConstant specifying the type of extrapolation to use in determining the incremental solution for a nonlinear analysis. Possible values are NONE, LINEAR, and PARABOLIC. The default value is LINEAR.

solutionTechnique
A SymbolicConstant specifying the technique used to for solving nonlinear equations. Possible values are FULL_NEWTON and QUASI_NEWTON. The default value is FULL_NEWTON.

reformKernel
An Int specifying the number of quasi-Newton iterations allowed before the kernel matrix is reformed.. The default value is 8.

convertSDI
A SymbolicConstant specifying whether to force a new iteration if severe discontinuities occur during an iteration. Possible values are PROPAGATED, CONVERT_SDI_OFF, and CONVERT_SDI_ON. The default value is PROPAGATED.

adaptiveDampingRatio

A Float specifying the maximum allowable ratio of the stabilization energy to the total strain energy and can be used only if *stabilizationMethod* is not NONE. The default value is 0.05.

continueDampingFactors

A Boolean specifying whether this step will carry over the damping factors from the results of the preceding general step. This parameter must be used in conjunction with the *adaptiveDampingRatio* parameter. The default value is OFF.

previous

A String specifying the name of the previous step. The new step appears after this step in the list of analysis steps.

description

A String specifying a description of the new step. The default value is an empty string.

explicit

A SymbolicConstant specifying whether the step has an explicit procedure type (*procedureType*=ANNEAL, DYNAMIC_EXPLICIT, or DYNAMIC_TEMP_DISPLACEMENT).

perturbation

A Boolean specifying whether the step has a perturbation procedure type.

nonmechanical

A Boolean specifying whether the step has a mechanical procedure type.

procedureType

A SymbolicConstant specifying the Abaqus procedure. Possible values are:

- ANNEAL
- BUCKLE
- COMPLEX_FREQUENCY
- COUPLED_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRIC
- DIRECT_CYCLIC
- DYNAMIC_IMPLICIT
- DYNAMIC_EXPLICIT
- DYNAMIC_SUBSPACE
- DYNAMIC_TEMP_DISPLACEMENT
- COUPLED_THERMAL_ELECTRICAL_STRUCTURAL
- FREQUENCY
- GEOSTATIC
- HEAT_TRANSFER
- MASS_DIFFUSION
- MODAL_DYNAMICS
- RANDOM_RESPONSE
- RESPONSE_SPECTRUM
- SOILS
- STATIC_GENERAL
- STATIC_LINEAR_PERTURBATION

- STATIC_RIKS
- STEADY_STATE_DIRECT
- STEADY_STATE_MODAL
- STEADY_STATE_SUBSPACE
- VISCO

suppressed A Boolean specifying whether the step is suppressed or not. The default value is OFF.

fieldOutputRequestState A repository of *FieldOutputRequestState* objects.

historyOutputRequestState A repository of *HistoryOutputRequestState* objects.

diagnosticPrint A *DiagnosticPrint* object.

monitor A *Monitor* object.

restart A *Restart* object.

adaptiveMeshConstraintStates A repository of *AdaptiveMeshConstraintState* objects.

adaptiveMeshDomains A repository of *AdaptiveMeshDomain* objects.

control A *Control* object.

solverControl A *SolverControl* object.

boundaryConditionStates A repository of *BoundaryConditionState* objects.

interactionStates A repository of *InteractionState* objects.

loadStates A repository of *LoadState* objects.

loadCases A repository of *LoadCase* objects.

predefinedFieldStates A repository of *PredefinedFieldState* objects.

Corresponding analysis keywords

- **VISCO*
- **STEP*

Step commands (miscellaneous)

Miscellaneous Step commands are used for configuring controls, damping, and frequency tables.

In this section:

- *CompositeDamping object*
- *CompositeDampingComponent object*
- *Control object*
- *DirectDamping object*
- *DirectDampingByFrequency object*
- *DirectDampingByFrequencyComponent object*
- *DirectDampingComponent object*
- *EmagTimeHarmonicFrequency object*
- *MassScaling object*
- *RandomResponseFrequency object*
- *RayleighDamping object*
- *RayleighDampingByFrequency object*
- *RayleighDampingByFrequencyComponent object*
- *RayleighDampingComponent object*
- *ResponseSpectrumComponent object*
- *SolverControl object*
- *SteadyStateDirectFrequency object*
- *SteadyStateModalFrequency object*
- *SteadyStateSubspaceFrequency object*
- *StepControl object*
- *StructuralDamping object*
- *StructuralDampingByFrequency object*
- *StructuralDampingByFrequencyComponent object*
- *StructuralDampingComponent object*
- *SubstructureGenerateFrequency object*
- *SubstructureGenerateModes object*

CompositeDamping object

A CompositeDamping object contains composite modal damping parameters.

Access

```
import step
mdb.models[name].steps[name].compositeDamping
```

Members

The CompositeDamping object can have the following member:

components A *CompositeDampingComponentArray* object.

CompositeDampingComponent object

A CompositeDampingComponent object is used to define composite damping over a range of modes.

Access

```
import step
mdb.models[name].steps[name].compositeDamping.components[i]
```

Members

The CompositeDampingComponent object has the following members:

start
An Int specifying the mode number of the lowest mode of a range.

end
An Int specifying the mode number of the highest mode of a range.

Control object

The Control object is used to provide additional optional general solution controls.

Access

```
import step
mdb.models[name].steps[name].control
```

`setValues(...)`

This method modifies the Control object.

Required arguments

None.

Optional arguments

allowPropagation

A Boolean specifying whether to allow all control values to propagate from a previous step. Setting this argument to ON automatically sets *resetDefaultValues* to OFF. The default value is ON.

resetDefaultValues

A Boolean specifying whether to use all default control values. Setting this argument to ON automatically sets *allowPropagation* to OFF. The default value is OFF.

discontinuous

A Boolean specifying whether to set *timeIncrementation* values that will usually improve efficiency for analyses with severely discontinuous behavior. The default value is OFF.

constraints

The SymbolicConstant DEFAULT or a sequence of Floats specifying tolerances on constraint equations. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

lineSearch

The SymbolicConstant DEFAULT or a sequence of Floats specifying line search control parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

timeIncrementation

The SymbolicConstant DEFAULT or a sequence of Floats specifying time incrementation control parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

directCyclic

The SymbolicConstant DEFAULT or a sequence of Floats specifying direct cyclic control parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

concentrationField
The SymbolicConstant DEFAULT or a sequence of Floats specifying mass concentration field equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

displacementField
The SymbolicConstant DEFAULT or a sequence of Floats specifying displacement field and warping degree of freedom field equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

electricalPotentialField
The SymbolicConstant DEFAULT or a sequence of Floats specifying electrical potential field equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

globalField
The SymbolicConstant DEFAULT or a sequence of Floats specifying parameters for all applicable field equilibrium equations. This argument overwrites all other field equilibrium equation control values. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

hydrostaticFluidPressureField
The SymbolicConstant DEFAULT or a sequence of Floats specifying hydrostatic fluid element volume constraint parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

poreFluidPressureField
The SymbolicConstant DEFAULT or a sequence of Floats specifying pore liquid volumetric continuity equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

rotationField
The SymbolicConstant DEFAULT or a sequence of Floats specifying rotation field equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

temperatureField
The SymbolicConstant DEFAULT or a sequence of Floats specifying temperature field equilibrium equation parameters. If a specified sequence contains a value of 0, that item in the sequence will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

vcctLinearScaling
The SymbolicConstant DEFAULT or a Float specifying linear scaling parameter for a VCCT debonding analysis. If a specified value is 0, it will be set to its system-defined value. The value can also be the SymbolicConstant DEFAULT. The default value is DEFAULT.

Return value
`None.`

Exceptions

RangeError.

Members

The Control object has members with the same names and descriptions as the arguments to the *setValues* method.

DirectDamping object

A DirectDamping object contains direct modal damping parameters.

Access

```
import step
mdb.models[name].steps[name].directDamping
```

Members

The DirectDamping object can have the following member:

components A *DirectDampingComponentArray* object.

DirectDampingByFrequency object

A DirectDampingByFrequency object contains direct damping parameters.

Access

```
import step
mdb.models[name].steps[name].directDampingByFrequency
```

Members

The DirectDampingByFrequency object can have the following member:

components A *DirectDampingByFrequencyComponentArray* object.

DirectDampingByFrequencyComponent object

A DirectDampingByFrequencyComponent object is used to define direct damping over a range of frequencies.

Access

```
import step
mdb.models[name].steps[name].directDampingByFrequency.components[i]
```

Members

The DirectDampingByFrequencyComponent object has the following members:

frequency
A Float specifying the frequency value in cycles/time.

fraction
A Float specifying the fraction of critical damping.

DirectDampingComponent object

A DirectDampingComponent object is used to define direct damping over a range of modes.

Access

```
import step
mdb.models[name].steps[name].directDamping.components[i]
```

Members

The DirectDampingComponent object has the following members:

start An Int specifying the mode number of the lowest mode of a range.

end An Int specifying the mode number of the highest mode of a range.

fraction A Float specifying the fraction of critical damping.

EmagTimeHarmonicFrequency object

A EmagTimeHarmonicFrequency is an object used to define frequency over range of modes.

Access

```
import step
mdb.models[name].steps[name].frequencyRange[i]
```

Members

The EmagTimeHarmonicFrequency object has the following members:

lower A Float specifying the lower limit of frequency range or a single frequency, in cycles/time.

upper A Float specifying the upper limit of frequency range, in cycles/time.

nPoints An Int specifying the number of points in the frequency range at which results should be given.

MassScaling object

A MassScaling object defines the region and controls that govern mass scaling.

Access

```
import step
mdb.models[name].steps[name].massScaling[i]
```

Members

The MassScaling object can have the following members:

objective
A SymbolicConstant specifying the objective of the mass scaling definition. Possible values are SEMI_AUTOMATIC, AUTOMATIC, and REINITIALIZE. The default value is SEMI_AUTOMATIC.

occurs
A SymbolicConstant specifying whether mass scaling should be performed at the beginning of the step or throughout the step. Possible values are AT_BEGINNING and THROUGHOUT_STEP.

type
A SymbolicConstant specifying the type of scaling. Possible values are UNIFORM, BELOW_MIN, SET_EQUAL_DT, and ROLLING. The default value is BELOW_MIN.

factor
A Float specifying a scaling factor.

dt
A Float specifying a target time increment.

frequency
An Int specifying the frequency at which mass scaling calculations are performed.

numberInterval
An Int specifying the number of intervals at which mass scaling calculations are performed.

feedRate
A Float specifying the estimated average velocity of the workpiece in the rolling direction at steady-state conditions.

extrudedLength
A Float specifying the average element length in the extruded direction.

crossSection
An Int specifying the number of nodes in the cross-section of the workpiece.

direction
A SymbolicConstant specifying the rolling direction. Possible values are GLOBAL_X, GLOBAL_Y, GLOBAL_Z, and GLOBAL_NONE. The default value is GLOBAL_X.

region
The SymbolicConstant MODEL or a *Region* object specifying where the mass scaling is applied. The default value is MODEL.

RandomResponseFrequency object

A RandomResponseFrequency is an object used to define frequency over a range of modes.

Access

```
import step
mdb.models[name].steps[name].freq[i]
```

Members

The RandomResponseFrequency object has the following members:

lower
A Float specifying the lower limit of the frequency range in cycles per time.

upper
A Float specifying the upper limit of the frequency range in cycles per time.

nCalcs
An Int specifying the number of points between eigenfrequencies at which the response should be calculated.

bias
A Float specifying the bias parameter.

RayleighDamping object

A RayleighDamping object contains Rayleigh Damping parameters.

Access

```
import step
mdb.models[name].steps[name].rayleighDamping
```

Members

The RayleighDamping object can have the following member:

components A *RayleighDampingComponentArray* object.

RayleighDampingByFrequency object

A RayleighDampingByFrequency object contains Rayleigh Damping parameters.

Access

```
import step
mdb.models[name].steps[name].rayleighDampingByFrequency
```

Members

The RayleighDampingByFrequency object can have the following member:

components A *RayleighDampingByFrequencyComponentArray* object.

RayleighDampingByFrequencyComponent object

A RayleighDampingByFrequencyComponent object is used to define Rayleigh damping over a range of frequencies.

Access

```
import step
mdb.models[name].steps[name].rayleighDampingByFrequency.components[i]
```

Members

The RayleighDampingByFrequencyComponent object has the following members:

frequency
A Float specifying the frequency value in cycles/time.

alpha
A Float specifying the mass proportional damping, α_M.

beta
A Float specifying the stiffness proportional damping, β_M.

RayleighDampingComponent object

A RayleighDampingComponent object is used to define Rayleigh damping over a range of modes.

Access

```
import step
mdb.models[name].steps[name].rayleighDamping.components[i]
```

Members

The RayleighDampingComponent object has the following members:

start
An Int specifying the mode number of the lowest mode of a range.

end
An Int specifying the mode number of the highest mode of a range.

alpha
A Float specifying the mass proportional damping, α_M.

beta
A Float specifying the stiffness proportional damping, β_M.

ResponseSpectrumComponent object

A ResponseSpectrumComponent is an element of the ResponseSpectrumComponentArray.

Access

```
import step
mdb.models[name].steps[name].components[i]
```

Members

The ResponseSpectrumComponent object has the following members:

x

A Float specifying the *X*-direction cosine.

y

A Float specifying the *Y*-direction cosine.

z

A Float specifying the *Z*-direction cosine.

scale

A Float specifying the scale factor.

timeDuration

A Float specifying the time duration of the dynamic event, from which this spectrum was created.



Note:

This parameter is ignored unless used with the DSC modal summation rule.

respSpectrum

A String specifying the name of the response spectrum specified with the keyword **SPECTRUM*.

SolverControl object

The SolverControl object is used to provide additional optional solver controls.

Access

```
import step
mdb.models[name].steps[name].solverControl
```

setValues(...)

This method modifies the SolverControl object.

Required arguments

None.

Optional arguments

allowPropagation
A Boolean specifying whether to allow all solver control values to propagate from a previous step. Setting this argument to ON automatically sets *resetDefaultValues* to OFF. The default value is ON.

resetDefaultValues
A Boolean specifying whether to use all default solver control values. Setting this argument to ON automatically sets *allowPropagation*to OFF. The default value is OFF.

relativeTolerance
The SymbolicConstant DEFAULT or a Float specifying the relative tolerance for convergence of the domain decomposition iterative solver. The default value is DEFAULT.

maxIterations
The SymbolicConstant DEFAULT or an Int specifying the maximum number of linear solver iterations. The default value is DEFAULT.

fillInLevel
The SymbolicConstant DEFAULT or an Int specifying the incomplete LU factorization fill-in level (for geostatic and soil analysis only). The default value is DEFAULT.

Return value

None.

Exceptions

RangeError.

Members

The SolverControl object has members with the same names and descriptions as the arguments to the *setValues* method.

SteadyStateDirectFrequency object

A SteadyStateDirectFrequency is an object used to define frequency over range of modes.

Access

```
import step
mdb.models[name].steps[name].frequencyRange[i]
```

Members

The SteadyStateDirectFrequency object has the following members:

lower
A Float specifying the lower limit of frequency range or a single frequency, in cycles/time.

upper
A Float specifying the upper limit of frequency range, in cycles/time.

nPoints
An Int specifying the number of points in the frequency range at which results should be given.

bias
A Float specifying the Bias parameter. When results are requested at four or more frequency points, Abaqus biases the results toward the ends of the intervals to obtain better resolution. The default value is 3.0.

SteadyStateModalFrequency object

A SteadyStateModalFrequency is an object used to define frequency over a range of modes.

Access

```
import step
mdb.models[name].steps[name].frequencyRange[i]
```

Members

The SteadyStateModalFrequency object has the following members:

lower
A Float specifying the lower limit of frequency range or a single frequency, in cycles/time.

upper
A Float specifying the upper limit of frequency range, in cycles/time.

nPoints
An Int specifying the number of points in the frequency range at which results should be given.

bias
A Float specifying the bias parameter. When results are requested at four or more frequency points, Abaqus biases the results toward the ends of the intervals to obtain better resolution. The default value is 3.0.

SteadyStateSubspaceFrequency object

A SteadyStateSubspaceFrequency is an object used to define frequency over range of modes.

Access

```
import step
mdb.models[name].steps[name].frequencyRange[i]
```

Members

The SteadyStateSubspaceFrequency object has the following members:

lower
A Float specifying the lower limit of frequency range or a single frequency, in cycles/time.

upper
A Float specifying the upper limit of frequency range, in cycles/time.

nPoints
An Int specifying the number of points in the frequency range at which results should be given.

bias
A Float specifying the Bias parameter. When results are requested at four or more frequency points, Abaqus biases the results toward the ends of the intervals to obtain better resolution. The default value is 3.0.

StepControl object

The StepControl object is used to control the progress of a simulation based on the solution state.

Access

```
import step
mdb.models[name].steps[name].stepControls[name]
```

StepControl(...)

This method creates a step control in a step.

Path

```
mdb.models[name].steps[name].StepControl
```

Required arguments

name A String specifying the name of the object.

data A Tuple of tuples specifying the values according to the data lines given in **STEP CONTROL*.

Example: If DT REFINEMENT = YES and ACTION = CONTINUE, then data = ((sensor name, Time increment value, sensor limit when the time increment refinement starts, Sensor limit value when the time increment refinement ends),). You can specify multiple tuples according to the number of sensors to be introduced; each inner tuple is considered as a new data line.

Optional arguments

action A symbolic constant defining a value of ACTION. The default value is CONTINUE.

dtRefinement A symbolic constant defining a value of DT REFINEMENT. The default value is YES.

Return value

A StepControl object.

Exceptions

RangeError.

`resume()`

This method resumes the step control that was previously suppressed.

Arguments

None.

Return value
`None.`

Exceptions

None.

`suppress()`

This method suppresses the step control.

Arguments

None.

Return value
`None.`

Exceptions

None.

`setValues(...)`

This method modifies the StepControl object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *StepControl* method, except for the *name* argument.

Return value
`None.`

Exceptions

RangeError.

Members

The StepControl object has members with the same names and descriptions as the arguments to the *StepControl* method.

In addition, the StepControl object has the following member:

suppressed
A Boolean specifying whether the step control is suppressed or not. The default value is OFF.

StructuralDamping object

A StructuralDamping object contains structural damping parameters.

Access

```
import step
mdb.models[name].steps[name].structuralDamping
```

Members

The StructuralDamping object can have the following member:

components A *StructuralDampingComponentArray* object.

StructuralDampingByFrequency object

A StructuralDampingByFrequency object contains structural damping parameters.

Access

```
import step
mdb.models[name].steps[name].structuralDampingByFrequency
```

Members

The StructuralDampingByFrequency object can have the following member:

components A *StructuralDampingByFrequencyComponentArray* object.

StructuralDampingByFrequencyComponent object

A StructuralDampingByFrequencyComponent object is used to define structural damping over a range of frequencies.

Access

```
import step
mdb.models[name].steps[name].structuralDampingByFrequency\
.components[i]
```

Members

The StructuralDampingByFrequencyComponent object has the following members:

frequency A Float specifying the frequency value in cycles/time.

factor A Float specifying the damping factor, s.

StructuralDampingComponent object

A StructuralDampingComponent object is used to define structural damping over a range of modes.

Access

```
import step
mdb.models[name].steps[name].structuralDamping.components[i]
```

Members

The StructuralDampingComponent object has the following members:

start An Int specifying the mode number of the lowest mode of a range.

end An Int specifying the mode number of the highest mode of a range.

factor A Float specifying the damping factor, s.

SubstructureGenerateFrequency object

A SubstructureGenerateFrequency object is used to define the modes to be used in a modal dynamic analysis. These modes are selected from the specified frequency range including the frequency boundary.

Access

```
import step
mdb.models[name].steps[name].frequencyRange[i]
```

Members

The SubstructureGenerateFrequency object has the following members:

lower
A Float specifying the lower limit of the frequency range, in cycles/time.

upper
A Float specifying the upper limit of the frequency range, in cycles/time.

SubstructureGenerateModes object

A SubstructureGenerateModes object is used to define the modes to be used in a modal dynamic analysis.

Access

```
import step
mdb.models[name].steps[name].modeRange[i]
```

Members

The SubstructureGenerateModes object has the following members:

start
An Int specifying the mode number of the lowest mode of a range.

end
An Int specifying the mode number of the highest mode of a range.

increment
An Int specifying the increment used to define the intermediate mode numbers beginning from the lowest mode to the highest mode.

Step commands (output)

Step output commands are used for configuring output requests, integrated output sections, diagnostic printing, monitoring, and restart.

In this section:

- *DiagnosticPrint object*
- *FieldOutputRequest object*
- *FieldOutputRequestState object*
- *HistoryOutputRequest object*
- *HistoryOutputRequestState object*
- *IntegratedOutputSection object*
- *Monitor object*
- *Restart object*
- *TimePoint object*

DiagnosticPrint object

The DiagnosticPrint object is used to request detailed diagnostic output or to disable specific diagnostic checks.

Access

```
import step
mdb.models[name].steps[name].diagnosticPrint
```

DiagnosticPrint(...)

This method creates a DiagnosticPrint object.

Path

```
mdb.models[name].steps[name].DiagnosticPrint
```

Required arguments

None.

Optional arguments

allke

A Boolean specifying a request for a column containing the total kinetic energy. This argument is valid only for an Abaqus/Explicit analysis. The default value is ON.

criticalElement

A Boolean specifying a request for a column containing the element that has the smallest stable time increment and a column listing the value. This argument is valid only for an Abaqus/Explicit analysis. The default value is ON.

dmass

A Boolean specifying a request for a column containing the percent change in total mass of the model as a result of mass scaling. This argument is valid only for an Abaqus/Explicit analysis. The default value is OFF unless mass scaling is present in the model.

etotal

A Boolean specifying a request for a column containing the energy balance of the model. This argument is valid only for an Abaqus/Explicit analysis. The default value is OFF.

contact

A Boolean specifying a request for detailed output of points that are contacting or separating in interface and gap problems. This argument is valid only for an Abaqus/Standard analysis. The default value is ON.

modelChange

A Boolean specifying a request for detailed output of which elements are being removed or reactivated in the step. This argument is valid only for an Abaqus/Standard analysis. The default value is OFF.

plasticity

A Boolean specifying a request for detailed output of element and integration point numbers for which the plasticity algorithms have failed to converge in the material routines. This argument is valid only for an Abaqus/Standard analysis. The default value is OFF.

residual

A Boolean specifying a request for output of equilibrium residuals during the equilibrium iterations. This argument is valid only for an Abaqus/Standard analysis. The default value is ON.

frequency An Int specifying the frequency of output, in increments. The default value is 1.

solve A Boolean specifying a request for information regarding the actual number of equations and the wavefront in each iteration. This argument is valid only for an Abaqus/Standard analysis. The default value is ON.

mass A Boolean specifying a request for a column containing the total mass of the model as a result of mass scaling. This argument is valid only for an Abaqus/Explicit analysis. The default value is OFF.

Return value

A DiagnosticPrint object.

Exceptions

None.

`setValues(...)`

This method modifies the DiagnosticPrint object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DiagnosticPrint* method.

Return value

`None.`

Exceptions

None.

Members

The DiagnosticPrint object has members with the same names and descriptions as the arguments to the *DiagnosticPrint* method.

Corresponding analysis keywords

- **DIAGNOSTICS*

FieldOutputRequest object

The FieldOutputRequest object defines a field output request.

Access

```
import step
mdb.models[name].fieldOutputRequests[name]
```

FieldOutputRequest(...)

This method creates a FieldOutputRequest object.

Path

```
mdb.models[name].FieldOutputRequest
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the object is created.

Optional arguments

region
The SymbolicConstant MODEL or a *Region* object specifying the region from which output is requested. The SymbolicConstant MODEL represents the whole model. The default value is MODEL.

variables
A sequence of Strings specifying output request variable or component names, or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables. The default value is PRESELECT.

frequency
The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

modes
The SymbolicConstant ALL or a sequence of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timeInterval
The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

numIntervals
An Int specifying the number of intervals during the step at which output database states are to be written. The default value is 20.

timeMarks
A Boolean specifying when to write results to the output database. OFF indicates that output is written immediately after the time dictated by the specified number of intervals.

ON indicates that output is written at the exact times dictated by the specified number of intervals. The default value is OFF.

boltLoad
A String specifying a bolt load from which output is requested.

sectionPoints
The SymbolicConstant DEFAULT or a sequence of Ints specifying the section points for which output requested. The default is DEFAULT.

interactions
`None` or a sequence of Strings specifying the interaction names. The default value is `None`.

The sequence can contain only one String.

rebar
A SymbolicConstant specifying whether output is requested for rebar. Possible values are EXCLUDE, INCLUDE, and ONLY. The default value is EXCLUDE.

filter
The SymbolicConstant ANTIALIASING or a String specifying the name of an output filter object. The default value is `None`.

directions
A Boolean specifying whether to output directions of the local material coordinate system. The default value is ON.

fasteners
A String specifying the fastener name. The default value is an empty string.

assembledFastener
A String specifying the assembled fastener name. The default value is an empty string.

assembledFastenerSet
A String specifying the set name from the model referenced by the assembled fastener, *assembledFastener*. The default value is an empty string.

exteriorOnly
A Boolean specifying whether the output domain is restricted to the exterior of the model. This argument is only valid if *region*=MODEL. The default value is OFF.

layupNames
A List of Composite Layer Names.

layupLocationMethod
A Symbolic constant specifying the method used to indicate the output locations for composite layups. Possible values are ALL_LOCATIONS, SPECIFIED and, TYPED_IN. The default value is SPECIFIED.

outputAtPlyTop
A Boolean specifying whether to output at the ply top section point. The default value is False.

outputAtPlyMid
A Boolean specifying whether to output at the ply midsection point. The default value is True.

outputAtPlyBottom
A Boolean specifying whether to output at the ply bottom section point. The default value is False.

position
A SymbolicConstant specifying the position on an element where output needs to be written. Possible values are INTEGRATION_POINTS, AVERAGED_AT_NODES, CENTROIDAL, and NODES. The default value is INTEGRATION_POINTS.

Return value

A FieldOutputRequest object.

Exceptions

None.

deactivate(...)

This method deactivates the field output request in the specified step and all its subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the field output request is deactivated.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

move(...)

This method moves the field output request state object from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the field output request state is moved.

toStepName A String specifying the name of the step to which the field output request state is moved.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

reset(...)

This method resets the field output request state of the specified step to the state of the previous step.

Required arguments

stepName	A String specifying the name of the step in which the field output request state is reset.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

resume()

This method resumes the field output request that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

TextError.

suppress()

This method suppresses the field output request.

Arguments

None.

Return value

`None.`

Exceptions

TextError.

setValues(...)

This method modifies the data for an existing FieldOutputRequest object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *FieldOutputRequest* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

`setValuesInStep(...)`

This method modifies the propagating data for an existing FieldOutputRequest object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the field output request is modified.

Optional arguments

variables A sequence of Strings specifying output request variable or component names, or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables.

frequency The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

modes The SymbolicConstant ALL or a sequence of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timeInterval The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

numIntervals An Int equal to the number of intervals during the step at which output database states are to be written. The default value is 20.

timePoint A String specifying the name of a time point object. The default value is equal to the number of intervals during the step at which output database states are to be written. The default value is `None`.

timeMarks A Boolean specifying when to write results to the output database. The default value is OFF.

Return value
`None`.

Exceptions
None.

Members
The FieldOutputRequest object can have the following members:

boltLoad
A String specifying a bolt load from which output is requested.

region
The SymbolicConstant MODEL or a *Region* object specifying the region from which output is requested. The SymbolicConstant MODEL represents the whole model. The default value is MODEL.

interactions
`None` or a tuple of Strings specifying the interaction names. The default value is `None`.

The sequence can contain only one String.

Corresponding analysis keywords
- **CONTACT OUTPUT*
- **ELEMENT OUTPUT*
- **ENERGY OUTPUT*
- **INCREMENTATION OUTPUT*
- **MODAL OUTPUT*
- **NODE OUTPUT*
- **OUTPUT*
- **RADIATION OUTPUT*

FieldOutputRequestState object

The FieldOutputRequestState object stores the propagating data of a field output request current in a step. One instance of this object is created internally by the *FieldOutputRequest* object for each step. The instance is also deleted internally by the *FieldOutputRequest* object.

The FieldOutputRequestState object has no constructor or methods.

Access

```
import step
mdb.models[name].steps[name].fieldOutputRequestState[name]
```

Members

The FieldOutputRequestState object can have the following members:

variablesState

A SymbolicConstant specifying the propagation state of the field output request variables. Possible values are UNSET, SET, and UNCHANGED.

frequency

The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

frequencyState

A SymbolicConstant specifying the propagation state of the field output request frequency. Possible values are UNSET, SET, and UNCHANGED.

modesState

A SymbolicConstant specifying the propagation state of the field output request modes. Possible values are UNSET, SET, and UNCHANGED.

timeInterval

The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

timeIntervalState

A SymbolicConstant specifying the propagation state of the field output request time interval. Possible values are UNSET, SET, and UNCHANGED.

numIntervals

An Int specifying the number of intervals during the step at which output database states are to be written. The default value is 20.

numIntervalsState

A SymbolicConstant specifying the propagation state of the field output request. Possible values are UNSET, SET, and UNCHANGED.

timeMarks

A Boolean specifying when to write results to the output database. The default value is OFF.

timeMarksState

A SymbolicConstant specifying the propagation state of the field output request. Possible values are UNSET, SET, and UNCHANGED.

timePointState

A SymbolicConstant specifying the propagation state of the field output request. Possible values are UNSET, SET, and UNCHANGED.

status

A SymbolicConstant specifying the propagation state of the FieldOutputRequestState object. Possible values are NOT_YET_ACTIVE, CREATED, PROPAGATED, MODIFIED, DEACTIVATED, NO_LONGER_ACTIVE, TYPE_NOT_APPLICABLE, and INSTANCE_NOT_APPLICABLE.

variables

A tuple of Strings specifying output request variable or component names, or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables.

modes

The SymbolicConstant ALL or a tuple of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timePoint

A String specifying the name of a time point object used to determine which output database states are to be written. The default value is an empty string.

frequencyType

A String specifying a read-only SymbolicConstant describing which type of frequency of output is used. Possible values areFREQUENCY, NUMBER_INTERVALS, TIME_INTERVAL, TIME_POINT and MODES. The default value depends on the procedure. The default value is an empty string.

HistoryOutputRequest object

The HistoryOutputRequest object defines a history output request.

Access

```
import step
mdb.models[name].historyOutputRequests[name]
```

HistoryOutputRequest(...)

This method creates a HistoryOutputRequest object.

Path

```
mdb.models[name].HistoryOutputRequest
```

Required arguments

name
A String specifying the repository key.

createStepName
A String specifying the name of the step in which the object is created.

Optional arguments

region
The SymbolicConstant MODEL or a *Region* object specifying the region from which output is requested. The SymbolicConstant MODEL represents the whole model. The default value is MODEL.

If the region is a surface region, the surface must lie within the general contact surface domain.

variables
A sequence of Strings specifying output request variable or component names, or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables. The default value is PRESELECT.

frequency
The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

modes
The SymbolicConstant ALL or a sequence of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timeInterval
The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

numIntervals
An Int specifying the number of intervals during the step at which output database states are to be written. The default value is 20.

boltLoad
A String specifying a bolt load from which output is requested. The default value is an empty string.

sectionPoints
The SymbolicConstant DEFAULT or a sequence of Ints specifying the section points for which output is requested. The default value is DEFAULT.

stepName
A String specifying the name of the step. The default value is an empty string.

interactions
`None` or a sequence of Strings specifying the interaction names. The default value is `None`.

The sequence can contain only one String.

contourIntegral
A String specifying the contour integral name. The default value is `None`.

numberOfContours
An Int specifying the number of contour integrals to output for the contour integral object. The default value is 0.

stressInitializationStep
A String specifying the name of the stress initialization step. The default value is `None`.

contourType
A SymbolicConstant specifying the type of contour integral. Possible values are J_INTEGRAL, C_INTEGRAL, T_STRESS, and K_FACTORS. The default value is J_INTEGRAL.

kFactorDirection
A SymbolicConstant specifying the stress intensity factor direction. Possible values are MTS, MERR, and K110. The *kFactorDirection* argument is valid only if *contourType*=K_FACTORS. The default value is MTS.

rebar
A SymbolicConstant specifying whether output is requested for rebar. Possible values are EXCLUDE, INCLUDE, and ONLY. The default value is EXCLUDE.

integratedOutputSection
A String specifying the integrated output section. The default value is an empty string.

springs
A sequence of Strings specifying the springs/dashpots names. The default value is `None`. The sequence can contain only one String.

filter
The SymbolicConstant ANTIALIASING or a String specifying the name of an output filter object. The default value is `None`.

fasteners
A String specifying the fastener name. The default value is an empty string.

assembledFastener
A String specifying the assembled fastener name. The default value is an empty string.

assembledFastenerSet
A String specifying the set name from the model referenced by the assembled fastener, *assembledFastener*. The default value is an empty string.

sensor
A Boolean specifying whether to associate the output request with a sensor definition. The default value is OFF.

useGlobal A Boolean specifying whether to output vector-valued nodal variables in the global directions. The default value is True.

Return value

A HistoryOutputRequest object.

Exceptions

None.

`deactivate(...)`

This method deactivates the history output request in the specified step and all subsequent steps.

Required arguments

stepName A String specifying the name of the step in which the history output request is deactivated.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

`move(...)`

This method moves the history output request state object from one step to a different step.

Required arguments

fromStepName A String specifying the name of the step from which the history output request state is moved.

toStepName A String specifying the name of the step to which the history output request state is moved.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

`reset(...)`

This method resets the history output request state of the specified step to the state of the previous step.

Required arguments

stepName	A String specifying the name of the step in which the history output request state is reset.

Optional arguments

None.

Return value

`None.`

Exceptions

TextError.

`resume()`

This method resumes the history output request that was previously suppressed.

Arguments

None.

Return value

`None.`

Exceptions

TextError.

`suppress()`

This method suppresses the history output request.

Arguments

None.

Return value

`None.`

Exceptions

TextError.

setValues(...)

This method modifies the data for an existing HistoryOutputRequest object in the step where it is created.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *HistoryOutputRequest* method, except for the *name* and *createStepName* arguments.

Return value

`None.`

Exceptions

None.

setValuesInStep(...)

This method modifies the propagating data for an existing HistoryOutputRequest object in the specified step.

Required arguments

stepName A String specifying the name of the step in which the history output request is modified.

Optional arguments

variables A sequence of Strings specifying output request variable or component names or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables.

frequency The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

modes The SymbolicConstant ALL or a sequence of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timeInterval The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

numIntervals An Int specifying the number of intervals during the step at which output database states are to be written. The default value is 20.

timePoints A String specifying the name of a time point object. The default value is equal to the number of intervals during the step at which output database states are to be written. The default value is `None`.

Return value
`None`.

Exceptions
None.

Members

The HistoryOutputRequest object can have the following members:

boltLoad
A String specifying a bolt load from which output is requested. The default value is an empty string.

region
The SymbolicConstant MODEL or a *Region* object specifying the region from which output is requested. The SymbolicConstant MODEL represents the whole model. The default value is MODEL.

If the region is a surface region, the surface must lie within the general contact surface domain.

sectionPoints
The SymbolicConstant DEFAULT or a tuple of Ints specifying the section points for which output is requested. The default value is DEFAULT.

interactions
`None` or a tuple of Strings specifying the interaction names. The default value is `None`.

The sequence can contain only one String.

Corresponding analysis keywords

- **CONTACT OUTPUT*
- **ELEMENT OUTPUT*
- **ENERGY OUTPUT*
- **INCREMENTATION OUTPUT*
- **MODAL OUTPUT*
- **NODE OUTPUT*
- **OUTPUT*
- **RADIATION OUTPUT*

HistoryOutputRequestState object

The HistoryOutputRequestState object stores the propagating data of a History output request current in a step. One instance of this object is created internally by the *HistoryOutputRequest* object for each step. The instance is also deleted internally by the *HistoryOutputRequest* object.

The HistoryOutputRequestState object has no constructor or methods.

Access

```
import step
mdb.models[name].steps[name].historyOutputRequestState[name]
```

Members

The HistoryOutputRequestState object can have the following members:

variablesState

A SymbolicConstant specifying the propagation state of the history output request variables. Possible values are UNSET, SET, and UNCHANGED.

frequency

The SymbolicConstant LAST_INCREMENT or an Int specifying the output frequency in increments. The default value is 1.

frequencyState

A SymbolicConstant specifying the propagation state of the history output request frequency. Possible values are UNSET, SET, and UNCHANGED.

modesState

A SymbolicConstant specifying the propagation state of the history output request modes. Possible values are UNSET, SET, and UNCHANGED.

timeInterval

The SymbolicConstant EVERY_TIME_INCREMENT or a Float specifying the time interval at which the output states are to be written. The default value is EVERY_TIME_INCREMENT.

timeIntervalState

A SymbolicConstant specifying the propagation state of the history output request time interval. Possible values are UNSET, SET, and UNCHANGED.

numIntervals

An Int specifying the number of intervals during the step at which output database states are to be written. The default value is 20.

timePointState

A SymbolicConstant specifying the propagation state of the history output request. Possible values are UNSET, SET, and UNCHANGED.

numIntervalsState

A SymbolicConstant specifying the propagation state of the history output request. Possible values are UNSET, SET, and UNCHANGED.

status

A SymbolicConstant specifying the propagation state of the HistoryOutputRequestState object. Possible values are NOT_YET_ACTIVE, CREATED, PROPAGATED, MODIFIED, DEACTIVATED, NO_LONGER_ACTIVE, TYPE_NOT_APPLICABLE, and INSTANCE_NOT_APPLICABLE.

variables

A tuple of Strings specifying output request variable or component names, or the SymbolicConstant PRESELECT or ALL. PRESELECT represents all default output variables for the given step. ALL represents all valid output variables.

modes

The SymbolicConstant ALL or a tuple of Ints specifying a list of eigenmodes for which output is desired. The default value is ALL.

timePoint

A String specifying the name of a time point object used to determine at which points in the time period data is written to the output database. The default value is an empty string.

frequencyType

A String specifying a read-only SymbolicConstant describing which type of frequency of output is used. Possible values areFREQUENCY, NUMBER_INTERVALS, TIME_INTERVAL, TIME_POINT and MODES. The default value depends on the procedure. The default value is an empty string.

IntegratedOutputSection object

The IntegratedOutputSection object specifies parameters used for integrated output.

Access

```
import step
mdb.models[name].integratedOutputSections[name]
```

IntegratedOutputSection(...)

This method creates an IntegratedOutputSection object.

Path

```
mdb.models[name].IntegratedOutputSection
```

Required arguments

name A String specifying the repository key.

Optional arguments

surface A *Region* object specifying the surface over which the output is based.

refPoint `None` or a *Region* object specifying the anchor point about which the integrated moment over the output region is computed or the SymbolicConstant None representing the global origin. The default value is `None`.

refPointAtCenter A Boolean specifying that the *refPoint* be adjusted so that it coincides with the center of the output region in the initial configuration. This argument is valid only when you include the *refPoint* argument. The default value is OFF.

refPointMotion A SymbolicConstant specifying how to relate the motion of *refPoint* to the average motion of the output region. A value of INDEPENDENT will allow the *refPoint* to move independent of the output region. A value of AVERAGE_TRANSLATION will set the displacement of the *refPoint* equal to the average translation of the output region. A value of AVERAGE will set the displacement and rotation of the *refPoint* equal to the average translation of the output region. The default value is INDEPENDENT.

This argument is valid only when you include the *refPoint* argument.

localCsys `None` or a *DatumCsys* object specifying the local coordinate system used to express vector output. If *localCsys*=`None`, the degrees of freedom are defined in the global coordinate system. The default value is `None`.

projectOrientation A Boolean specifying that the coordinate system be projected onto the *surface* such that the 1–axis is normal to the *surface*. Projection onto a planar *surface* is such that the 1-axis is normal to the surface, and a projection onto a nonplanar *surface* is such that a least-squares fit surface will be used. The default value is OFF.

Return value

An IntegratedOutputSection object.

Exceptions

None.

setValues(...)

This method modifies the IntegratedOutputSection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *IntegratedOutputSection* method, except for the *name* argument.

Return value

`None.`

Exceptions

None.

Members

The IntegratedOutputSection object has members with the same names and descriptions as the arguments to the *IntegratedOutputSection* method.

Monitor object

The Monitor object defines a degree of freedom to monitor.

Access

```
import step
mdb.models[name].steps[name].monitor
```

Monitor(...)

This method creates a request for a degree of freedom to be monitored in a general or modal procedure.

Path

```
mdb.models[name].steps[name].Monitor
```

Required arguments

node

A String specifying the name of the region to be monitored.

dof

A SymbolicConstant specifying the degree of freedom to be monitored at the node. Possible values are:

- U1
- U2
- U3
- UR1
- UR2
- UR3
- WARP
- FLUID_PRESSURE
- ELECTRICAL_POTENTIAL
- NT11
- NT30
- NN11
- NN30

The NT identifiers are not available for mass diffusion. The NN identifiers are available only for mass diffusion.

frequency

An Int specifying the output frequency in increments. This argument is valid only for an Abaqus/Standard analysis.

Optional arguments

None.

Return value

A Monitor object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the Monitor object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Monitor* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Monitor object has members with the same names and descriptions as the arguments to the *Monitor* method.

Corresponding analysis keywords

- **MONITOR*

Restart object

The Restart object defines a restart request.

Access

```
import step
mdb.models[name].steps[name].restart
```

Restart(...)

This method creates a restart request.

Path

```
mdb.models[name].steps[name].Restart
```

Required arguments

None.

Optional arguments

numberIntervals

An Int specifying the number of intervals during the step at which restart information will be written. The default value is 0. The default value is 1.

timeMarks

A Boolean specifying whether to use exact time marks for writing during an analysis. The default value is OFF. The default value is OFF.

overlay

A Boolean specifying that only one increment per step should be retained on the restart file, thus minimizing the size of the restart file. The default value is OFF. The default value is ON.

frequency

An Int specifying the increments at which restart information will be written. The default value is 0. The default value is 0.

This argument applies only to Abaqus/Standard analyses.

Return value

A Restart object.

Exceptions

RangeError.

setValues(...)

This method modifies the Restart object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Restart* method.

Return value

`None.`

Exceptions

RangeError.

Members

The Restart object has members with the same names and descriptions as the arguments to the *Restart* method.

Corresponding analysis keywords

- **RESTART*

TimePoint object

The TimePoint object defines time points at which data are written to the output database or restart files.

Access

```
import step
mdb.models[name].timePoints[name]
```

TimePoint(...)

This method creates a TimePoint object.

Path

```
mdb.models[name].TimePoint
```

Required arguments

name A String specifying the repository key.

points A sequence of sequences of Floats specifying time points at which data are written to the output database or restart files.

Optional arguments

None.

Return value

A TimePoint object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the TimePoint object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TimePoint* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TimePoint object has members with the same names and descriptions as the arguments to the *TimePoint* method.

Corresponding analysis keywords

- **TIME POINTS*

Table Collection commands

Table Collection commands are used to create dynamic table types and tables.

In this section:

- *ActivateElements object*
- *DataTable object*
- *ElementProgressiveActivation object*
- *EventSeries object*
- *EventSeriesType object*
- *ParameterColumn object*
- *ParameterTable object*
- *PropertyTable object*
- *PropertyTableData object*
- *TableCollection object*

ActivateElements object

The ActivateElements object is used turn on progressive element activation within a step definition.

Access

```
mdb.models[name].steps[name].activateElements[key]
```

ActivateElements(...)

This method creates an ActivateElements object.

Path

```
mdb.models[name].ActivateElements
```

Required arguments

tableCollection	A String specifying the name of the tableCollection object.
activation	A string specifying the name of progressive element activation.

Optional arguments

eigenTimeConst	A Double specifying the time constant used to ramp up the eigenstrains at element activation.
expansionTimeConstant	A Double specifying the time constant used to ramp up the thermal strains at element activation.

Return value

An ActivateElements object.

Exceptions

RangeError.

`setValues(...)`

This method modifies the ActivateElements object.

Optional arguments

The optional arguments to `setValues` are the same as the optional arguments to the *ActivateElements* method.

Exceptions

RangeError.

Members

The ActivateElements object has members with the same names and descriptions as the arguments to the *ActivateElements* method.

Corresponding analysis keywords

- **ActivateElements*
- **ElementProgressiveActivation*

DataTable object

The DataTable object is used to specify the parameter table of the respective parameter table type.

The data type of the values in each column in the DataTable object corresponds to the data type mentioned for the respective *ParameterColumn* object. The DataTable object should be created when all the required ParameterColumn objects are created for the current ParameterTable.

Access

```
mdb.models[name].tableCollections[name].parameterTables[name].dataTables[i]
```

DataTable(...)

This method creates a DataTable object and places it in the dataTables array.

Path

```
mdb.models[name].tableCollections[name].parameterTables[name].DataTable
```

Required arguments

label A String specifying a unique label name for the current ParameterTable object.

Optional arguments

None.

Return value

A DataTable object.

Exceptions

AbaqusException.

Members

The DataTable object has the following members.

label A String specifying the label of the data table.

columns A DataColumnArray specifying all the dataColumns in the DataTable object.

Corresponding analysis keywords

- **PARAMETER TABLE*

ElementProgressiveActivation object

The ElementProgressiveActivation object is used to specify elements that can be activated during an analysis.

Access

```
mdb.models[name].rootAssembly.elementProgressiveActivations[name]
```

ElementProgressiveActivation(...)

This method creates an ElementProgressiveActivation object and places it in the elementProgressiveActivation repository.

Path

```
mdb.models[name].rootAssembly.ElementProgressiveActivation
```

Required arguments

name	A String specifying the key of the repository.

Optional arguments

elset	A *Region* object specifying the region containing the elements that will be activated during the analysis.
deformation	A Boolean value specifying whether the elements that have not yet been activated will follow the deformations of the active elements. Set *deformation*=ON when the deformation of the active elements is excessive. The default value is OFF.
freeSurfaceType	A SymbolicConstant specifying the exposed areas of the element facets that are active for convection or radiation boundary conditions to be applied. Possible values are NONE and FACET. If *freeSurfaceType*=FACET, user subroutine *UEPACTIVATIONFACET* will be called at the start of the increment for each element. If *freeSurfaceType*=NONE, all the exposed areas of the element facets are considered. The default value is NONE.

Return value

An ElementProgressiveActivation object.

Exceptions

AbaqusException: If the region does not contain only elements.

setValues

Members

The ElementProgressiveActivation object has members with the same names and descriptions as the arguments to the *ElementProgressiveActivation* method.

Corresponding analysis keywords

- **ELEMENT PROGRESSIVE ACTIVATION*

EventSeries object

The EventSeries object is used to define an event based on an already defined EventSeriesType object.

After EventSeries is instantiated, making changes to EventSeriesType may lead to data corruption.

Access

```
mdb.models[name].eventSeriesDatas[name]
```

EventSeries(...)

This method creates an EventSeries object.

Path

```
mdb.models[name].EventSeriesData
```

Required arguments

name	A String specifying the repository key.
createStepName	A string specifying the step name.
eventSeriesType	A string specifying the type of event series.

Optional arguments

transformType	A Symbolic constant specifying the type of transformation. Possible values are NONE, BOTH, TRANSLATE, and ROTATE. The default value is NONE.
timeSpan	A Symbolic constant specifying time. Possible values are TOTAL_TIME and STEP_TIME. The default value is STEP_TIME.
transformations	An Array specifying the required transformations over event series data.
fileName	A String specifying the filename.
data	An Array of double specifying the values of fields provided in EventSeriesType.

Return value

An EventSeries object.

Exceptions

RangeError.

setValues(...)

This method modifies the EventSeries object.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EventSeries* method, except for the *name* argument.

Exceptions

RangeError.

Members

The EventSeries object has members with the same names and descriptions as the arguments to the *EventSeries* method.

Corresponding analysis keywords

- **EVENT SERIES TYPE*
- **EVENT SERIES*

EventSeriesType object

The EventSeriesType object is used to define a type of event in a process.

Access

```
mdb.models[name].eventSeriesTypes[name]
```

EventSeriesType(...)

This method creates an EventSeriesType object.

Path

```
mdb.models[name].EventSeriesType
```

Required arguments

name	A String specifying the repository key.
createStepName	A string specifying the step name.

Optional arguments

fields	A String array specifying fields. The default value is an empty array.

Return value

A EventSeriesType object.

Exceptions

RangeError.

setValues(...)

This method modifies the EventSeriesType object.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EventSeriesType* method, except for the *name* argument.

Exceptions

RangeError.

Members

The EventSeriesType object has members with the same names and descriptions as the arguments to the *EventSeriesType* method.

Corresponding analysis keywords

- **EVENT SERIES TYPE*
- **EVENT SERIES*

ParameterColumn object

The ParameterColumn object is used to define the type of parameters that will collectively build the type of parameter table.
Once the object is created, it is noneditable.

Access

```
mdb.models[name].tableCollections[name].parameterTables[name].columns[i]
```

ParameterColumn(...)

ParameterColumn is a constructor method that creates a ParameterColumn object and stores it in the array data structure. It is accessible from the ParameterTable object using a column member.

Path

```
mdb.models[name].tableCollections[name].parameterTables[name].Column
```

Required arguments

type	A SymbolicConstant specifying the data type of the parameter. Possible values are STRING, INTEGER, and FLOAT.

Optional arguments

unit	A String specifying the unit of the parameter.
description	A String specifying the description of the parameter.
default	The default value of the first parameter. The data type of the value depends on the value of *type* argument.
allowedValues	A Set of allowed values for the parameter.

Return value

A ParameterTable object.

Exceptions

Incompatible data are given.

Members

The ParameterTable object has members with the same names and descriptions as the arguments to the *ParameterColumn* method.

Corresponding analysis keywords

- **PARAMETER TABLE TYPE*
- **PARAMETER TABLE*

ParameterTable object

A ParameterTable is an object that is used to define the containers that encapsulate ParameterColumn and DataTable objects.
The data of DataTable is dependent on the contents of ParameterColumn. After DataTable is instantiated, making changes to ParameterColumn may lead to data corruption.

The following topics are discussed:

Access

```
mdb.models[name].tableCollections[name].parameterTables[name]
```

ParameterTable(...)

This method creates a ParameterTable object and places it in the parameterTables repository.

Path

```
mdb.models[name]tableCollections[name].ParameterTable
```

Required arguments

name A String specifying the repository key.

Optional arguments

None.

Return value

A ParameterTable object.

Exceptions

None.

Members

The ParameterTable object has the following members:

columns A *ParameterColumnArray* specifying all the columns in the ParameterTable.

dataTables A *DataTableArray* specifying all the dataTables in the ParameterTable.

Corresponding analysis keywords

- **PARAMETER TABLE TYPE*
- **PARAMETER TABLE*

PropertyTable object

A PropertyTable is an object that is used to define the container that encapsulates the PropertyTableData object.

The data of the PropertyTableData object is dependent on the contents of the PropertyTable object.

After PropertyTableDatais instantiated, making changes to PropertyTable may lead to data corruption.

Access

```
mdb.models[name].tableCollections[name].propertyTables[name]
```

PropertyTable(...)

This method creates a PropertyTable object.

Path

```
mdb.models[name].tableCollections[name].PropertyTable
```

Required arguments

name	A String specifying the repository key.
properties	A string array specifying the multiple properties to build the parameter table type.

Optional arguments

variables	A String array specifying multiple independent variables. The default value is an empty array.

Return value

A PropertyTable object.

Exceptions

RangeError.

setValues(...)

This method modifies the PropertyTable object.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PropertyTable* method, except for the *name* argument.

Exceptions

RangeError.

Members

The PropertyTableType object has members with the same names and descriptions as the arguments to the *PropertyTable* method.

In addition, the PropertyTable object has following members:

propertyTableDatas A repository of *PropertyTableData*. Specifies all the propertyTableData in PropertyTable

Corresponding analysis keywords

- **PROPERTY TABLE TYPE*
- **PROPERTY TABLE*

PropertyTableData object

A PropertyTableData is an object that is used to specify the property table of the respective property table type.

The values in each column in the PropertyTableData object corresponds to the properties and variables mentioned in the PropertyTable object.

Access

```
mdb.models[name].tableCollections[name].propertyTables[name].propertyTableDatas[name]
```

PropertyTableData(...)

This method creates a PropertyTableData object.

Path

```
mdb.models[name].tableCollections[name].propertyTables[name].PropertTableData
```

Optional arguments

label	A String specifying a unique label name for the current PropertyTable object.
regularize	A SymbolicConstant specifying the type of regularize to the user-defined property data.
extrapolate	A SymbolicConstant specifying the type of extrapolation of dependent variables outside the specified range of the independent variables.
isTemp	A Boolean specifying the dependency of properties on temperature.
fieldNums	An Int specifying the field variables on which properties are dependent.
regularizeTolerance	A Double specifying the tolerance to be used to regularize the property table data.
data	An Array of doubles specifying the values of the properties, the variables mentioned in PropertyTable, and the field variables mentioned in PropertyTableData.

Return value

A PropertyTableData object.

Exceptions

RangeError.

setValues(...)

This method modifies the PropertyTableData object.

Optional arguments

Exceptions

RangeError.

Members

The PropertyTableData object has members with the same names and descriptions as the arguments to the *PropertyTableData* method.

Corresponding analysis keywords

- **PROPERTY TABLE TYPE*
- **PROPERTY TABLE*

TableCollection object

A TableCollection is an object used to define the containers that encapsulate the ParameterTable and PropertyTable objects.

Access

```
mdb.models[name].tableCollections[name]
```

TableCollection(...)

This method creates a TableCollection object and places it in the tableCollections repository.

Path

```
mdb.models[name].TableCollection
```

Required arguments

name A String specifying the repository key.

Optional arguments

None.

Return value

A TableCollection object.

Exceptions

None.

Members

The TableCollection object has the following members:

propertyTables A repository of the PropertyTable object.

parameterTables A repository of the ParameterTable object

Corresponding analysis keywords

- **TABLE COLLECTION*

Text Representation commands

Text representation commands are used to determine how objects are printed on the Python command line.

In this section:

- *redentABQ module*
- *textRepr module*
- *TextReprOptions object*

redentABQ module

The Python module redentABQ is a wrapper around the Python redent module. The module makes the indentation in a Python text file consistent.

The module can be run as a script from a command prompt using the following statement:

```
abaqus Python -m redentABQ [options] pathToFile
```

where the following options are available to customize the redentABQ process:

- `-b`: Create a backup of the selected file if any modifications are made.
- `-i`: Specify a number of spaces to be used for indentation.
- `-t`: Test the newly indented file after any changes are made.

For more information on running the **abaqus Python** execution procedure, see *Python Execution*.

Access

```
import redentABQ
```

`indentFile(...)`

This method outputs the indented file to the terminal window or backs up the specified file and replaces it.

Required arguments

path A String specifying the file to be processed.

Optional arguments

indent A String specifying the amount of indentation to be used. The default value is four spaces of indentation. Custom indentation strings must be flanked by quotation marks (`" "`).

backup A Boolean specifying whether to back up the file specified in the *path* argument if changes are made. The default value is False.

runTest A Boolean specifying whether to test the newly indented file to ensure that it is semantically the same after any changes. The default value is False.

Return value

A Boolean. True, if the indentation is successful; otherwise, False.

Exceptions

None.

textRepr module

The Python module `textRepr` contains functions to print an Abaqus object and to convert the output from the `str()` function into a formatted string, indented for each object in the recursive listing of a Python object.

Access

```
import textRepr
```

`getIndentedRepr(...)`

This method returns a String with each level of parentheses indented.

Required arguments

object
A Python object to be processed. (The Python object can be an Abaqus object.) This argument can also be a String representation of a Python object obtained from str(*object*).

Optional arguments

maxRecursionDepth
An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. If *object* is not an Abaqus object, *maxRecursionDepth* has no effect. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence
An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

significantDigits
An Int specifying the number of significant digits for Floats in the output. Possible values are $0 < significantDigits \leq 15$. The default value is 6.

Return value

A String.

Exceptions

None.

`getPaths(...)`

This method processes the argument and interprets its structure. It then returns a String containing the object paths of all the child objects found.

Required arguments

object
A Python object to be processed. (The Python object can be an Abaqus object.) This argument can also be a String representation of a Python object.

Optional arguments

maxRecursionDepth
An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. If *object* is not an Abaqus object, *maxRecursionDepth* has no effect. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence
An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

pathRoot
A String specifying the root of the paths. This String is prepended to each path found. The default value is `None`, implying that the path to *object* will be used unless *object* is a String.

Return value

A String.

Exceptions

None.

`getTypes(...)`

This method processes the argument, interprets its structure, and returns a String containing all the object types within the object in the form '*TypeNamePath*'.

Required arguments

object
An Abaqus object.

Optional arguments

maxRecursionDepth
An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

pathRoot A String specifying the root of the paths. This String is prepended to each path found. The default value is `None`, implying that the path to *object* will be used unless *object* is a String.

Return value

A String.

Exceptions

None.

`prettyPrint(...)`

This method prints a formatted version of the object. The `prettyPrint` function uses `getIndentedRepr` to format the String representation, but does not save the full text output of `getIndentedRepr`. Therefore this function can be used in cases where `getIndentedRepr` would run out of memory.

Required arguments

object An Abaqus object.

Optional arguments

maxRecursionDepth An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

significantDigits An Int specifying the number of significant digits for Floats in the output. Possible values are $0 < significantDigits \leq 15$. The default value is 6.

Return value

`None.`

Exceptions

None.

printPaths(...)

This method prints the object path of the *object* argument and its members, depending on the *maxRecursionDepth* argument. The `printPaths` function uses `getPaths` to print a list of paths to each of the child objects in *object*.

Required arguments

object
An Abaqus object. This argument can also be a String representation of an Abaqus object obtained from str(*object*).

Optional arguments

maxRecursionDepth
An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence
An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

pathRoot
A String specifying the root of the paths to be printed. This String is prepended to each path found. The default value is `None`, implying that the path to *object* will be used.

Return value

`None.`

Exceptions

None.

printTypes(...)

This method prints the object type. The `printTypes` function uses `getTypes` to print a list of all the object types in *object*.

Required arguments

object
An Abaqus object.

Optional arguments

maxRecursionDepth
An Int specifying the maximum depth to which the object will be printed, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. The default value is `None`.

You should take care when setting *maxRecursionDepth*=UNLIMITED because the resulting output can be very large. To limit the output, you should set *maxElementsInSequence* to a small number.

maxElementsInSequence

An Int specifying the maximum number of elements of a sequence to return or the SymbolicConstant UNLIMITED. The initial value is 100. After the maximum number of elements have been printed, the remainder are indicated by the string '...'.

pathRoot

A String specifying the root of the paths to be printed. This String is prepended to each path found. The default value is `None`, implying that the path to *object* will be used.

Return value

`None`.

Exceptions

None.

TextReprOptions object

The TextReprOptions object is used to configure the output of the Python`str()` command within Abaqus/CAE. (The `str()` command is used by the Python print function.) The TextReprOptions object stores the various settings that determine how objects are printed on the Python command line.

The TextReprOptions object has no constructor. Abaqus creates the *textReprOptions* member when a session is started.

Access

```
import textRepr
textReprOptions
session.textReprOptions
```

`setValues(...)`

This method modifies the TextReprOptions object.

Required arguments

None.

Optional arguments

style

A SymbolicConstant specifying the style of the text representation. Possible values are:

- RECURSIVE: return a recursive representation of the object.
- SIMPLE: return a String representing the object as either 'interface object', 'dictionary object', or 'sequence object'.

The initial value is RECURSIVE.

maxRecursionDepth

An Int, SymbolicConstant, or `None` specifying the maximum depth to which the object will be printed. Possible values for the depth are Ints ≥ 0, the SymbolicConstant UNLIMITED, or `None`. A value of `None` implies that the current setting in the TextReprOptions object will be used. If *object* is not an Abaqus object, *maxRecursionDepth* has no effect. The default value is `None`.

maxRecursionString

A String specifying the string to be returned when the maximum depth of recursion is reached. The string can include a format specifier (%s), which will be substituted by the object type. The initial value is '%s object'.

maxElementsInSequence

An Int or the SymbolicConstant UNLIMITED specifying the maximum number of elements of a sequence to return. Possible values are UNLIMITED or Ints > 0. The initial value is 100. After the maximum number of elements, the remainder are indicated by the string '...'.

Return value

`None.`

Exceptions

None.

Members

The TextReprOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

Utility and View commands

Utility commands are used to do the following:

- Create SymbolicConstant and Boolean objects.
- Change the name of a key to a Repository object.
- Control the use of deprecated commands.
- Request input from the user.
- Display status information.
- Execute Python commands or functions.
- Register a callback that will be invoked when certain Abaqus Scripting Interface commands are about to be executed.
- Register a callback that will be invoked when certain Abaqus Scripting Interface objects are about to be deleted.
- Upgrade Abaqus kernel or GUI scripts from one release of Abaqus to a later release.

View commands are used to transform three-dimensional views stored in Session and Viewport objects to the screen. (See *Session object* and *Viewport object*.)

In this section:

- *SymbolicConstant object*
- *AbaqusBoolean object*
- *Repository object*
- *BackwardCompatibility object*
- *User input commands*
- *Status commands*
- *Callback commands*
- *Method callback command*
- *Delete object callback command*
- *Upgrade script commands*
- *Customization commands*
- *View object*

SymbolicConstant object

The SymbolicConstant object represents a string in a way that can be stored in a replay file and used as an argument to Abaqus Scripting Interface methods and functions. By convention the string representation of the SymbolicConstant object is the same as its variable name. If you pass a SymbolicConstant object to the Python `repr()` function, the function returns the text without quotes. In effect, the text is the variable that, by convention, refers to the SymbolicConstant object.

Two SymbolicConstant objects with the same text are the same Python object, although you can assign them to different variables. All of the SymbolicConstant objects that are required in Abaqus Scripting Interface methods are defined in the abaqusConstants module. Some SymbolicConstant objects and the SymbolicConstant constructor are defined in the abaqus module. The SymbolicConstant constructor is also defined in the symbolicConstants module.

Access

```
from symbolicConstants import *
from abaqusConstants import *
```

SymbolicConstant(...)

The `SymbolicConstant` method creates a SymbolicConstant object.

Path

SymbolicConstant

Required arguments

text
A String specifying the text of the SymbolicConstant object. The String must contain only capital letters, digits, or underscores and must not start with a digit.

Optional arguments

None.

Return value

A SymbolicConstant object.

Exceptions

None.

AbaqusBoolean object

The AbaqusBoolean object is used in a similar way to the SymbolicConstant object. If you pass an AbaqusBoolean object to the Python `repr()` function, the function returns the text without quotes. In effect, the text is the variable that, by convention, refers to the AbaqusBoolean object.

An AbaqusBoolean object has a value of 0 or 1, and an AbaqusBoolean object can be tested in an `if` statement. You can use an AbaqusBoolean object as an argument to a method in place of 1 or 0. Conversely, you can pass a 0 or a 1 to an Abaqus Scripting Interface method that expects a Boolean argument, and the 0 or 1 will be coerced to the appropriate AbaqusBoolean value.

There are only two possible values for an AbaqusBoolean object: 1 and 0. You can import both values from the symbolicConstants module or from the abaqus module. Abaqus Scripting Interface commands that expect an AbaqusBoolean object will also accept a Python bool (True, False), or a Python int (1, 0).

Access

```
from symbolicConstants import *
from abaqusConstants import *
```

AbaqusBoolean(...)

The `AbaqusBoolean` method creates an AbaqusBoolean object.

Path

AbaqusBoolean

Required arguments

value
An Int specifying whether the AbaqusBoolean object will test true or false. Possible values are 0 and 1, which will create the `AbaqusBoolean` method OFF and ON, respectively.

Optional arguments

None.

Return value

An AbaqusBoolean object.

Exceptions

None.

Repository object

Repositories are containers that store a particular type of object; for example, the steps repository contains all the steps defined in the model. An Abaqus Scripting Interface Repository maps a key to a value. The key is usually a String, and the value is any Python object, usually an Abaqus object. A repository is similar to a Python dictionary; however, only a constructor can add an object to a repository. In addition, all of the objects in a repository are of the same base type. For more information, see *Repositories*. A Repository has no constructor. Abaqus creates empty repositories when you import a module. For example, Abaqus creates an empty parts repository when you import the `part` module.

The following methods of the Repository object are standard Python dictionary methods and are not described here:

- `has_key`
- `items`
- `keys`
- `values`

`changeKey(...)`

This method changes the name of a key in a repository and the *name* member of the value object.

Required arguments

fromName A String specifying the old name of the repository key.

toName A String specifying the new name of the repository key.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

BackwardCompatibility object

The BackwardCompatibility object enables the user to control access to deprecated commands in the Abaqus Scripting Interface and to collect data on which deprecated commands have been used. This enables the user to check that no deprecated methods, members, or arguments have been used. Data are recorded on any deprecated members, methods, or arguments that are used.

The BackwardCompatibility object has no constructor. The abaqus module and the odbAccess module each have a member named backwardCompatibility.

By default, the value of the *includeDeprecated* member is ON, and Abaqus will execute a script containing deprecated commands with no indication that you should update the script. You can do either of the following to change the value of the *includeDeprecated* member and to determine which commands have been deprecated:

- From the command line interface in Abaqus/CAE or from an Abaqus Scripting Interface script that is run from within Abaqus/CAE, use the following command:

  ```
  backwardCompatibility.setValues(includeDeprecated=OFF)
  ```

- From an Abaqus Scripting Interface script that is run using `abaqus python` at the system prompt, use the following two commands:

  ```
  from odbAccess import *
  backwardCompatibility.setValues(includeDeprecated=OFF)
  ```

In addition, the BackwardCompatibility object provides tools to assist you in determining the deprecated commands that have been used. For example, to determine the deprecated commands used in the script `createLug.py`, use the following commands:

```
backwardCompatibility.resetDeprecatedMethodsUsed()
backwardCompatibility.resetDeprecatedMembersUsed()
backwardCompatibility.resetDeprecatedArgsUsed()
execfile('createLug.py')
print(backwardCompatibility.getDeprecatedMethodsUsed())
print(backwardCompatibility.getDeprecatedMembersUsed())
print(backwardCompatibility.getDeprecatedArgsUsed())
```

Access

```
backwardCompatibility
```

getDeprecatedMethodsUsed()

This method returns a list of deprecated methods used since the last call to `resetDeprecatedMethodsUsed`.

Arguments

None.

Return value

A list of Strings.

Exceptions

None.

getDeprecatedMembersUsed()

This method returns a list of deprecated members used since the last call to `resetDeprecatedMembersUsed`.

Arguments

None.

Return value

A list of Strings.

Exceptions

None.

getDeprecatedArgsUsed()

This method returns a list of deprecated arguments used since the last call to `resetDeprecatedArgsUsed`.

Arguments

None.

Return value

A list of Strings.

Exceptions

None.

resetDeprecatedMethodsUsed()

This method clears the list of deprecated methods used.

Arguments

None.

Return value

`None.`

Exceptions

None.

resetDeprecatedMembersUsed()

This method clears the list of deprecated members used.

Arguments

None.

Return value
`None.`

Exceptions

None.

`resetDeprecatedArgsUsed()`

This method clears the list of deprecated arguments used.

Arguments

None.

Return value
`None.`

Exceptions

None.

`setValues(...)`

This method modifies the BackwardCompatibility object.

Required arguments

None.

Optional arguments

includeDeprecated	A Boolean specifying whether deprecated members, methods, and arguments can be seen and used. The default value is ON.
reportDeprecated	A Boolean specifying whether a warning is displayed after running a script that contains deprecated commands. The default value is True.
showKeysInReport	A Boolean specifying whether the keys and indices are included in the report that is displayed when *reportDeprecated* is True. The default value is False.

Return value
`None.`

Exceptions

None.

Members

The BackwardCompatibility object has the following members:

includeDeprecated

A Boolean specifying whether deprecated members, methods, and arguments can be seen and used. Possible values are ON and OFF. The default value is ON.

reportDeprecated

A Boolean specifying whether a warning is displayed after running a script that contains deprecated commands. The default value is True.

showKeysInReport

A Boolean specifying whether the keys and indices are included in the report that is displayed when *reportDeprecated* is True. The default value is False.

User input commands

The following commands are used to request data entry from a user.

getInput

This method is used to obtain a single input from a user from a dialog box. The method displays a modal dialog box with a text field prefaced by the specified label. The user enters data in the text field or accepts the optional default value that is displayed in the text field. When the user clicks the **OK** button, the `getInput` method reads the data from the dialog box and closes the dialog box. You can use a "\t" separator in the label argument to provide a tooltip from the dialog box; for example,

```
getInput('Enter a number:\tEnter the number of nodes to delete')
```



Note:

You cannot use a script that contains `getInput` if you are running the script from the command line and passing the script name to the command line options **-start**, **-script**, **-replay**, or **-noGUI**.

Path

getInput

Required arguments

prompt A String specifying the text field in the dialog box.

Optional arguments

default A String specifying a default value to be displayed in the text field in the dialog box.

Return value

A String or `None` if the user clicks **Cancel**.

Exceptions

None.

getInputs

This method is used to obtain multiple inputs from a user from a dialog box. The method displays a modal dialog box with a column of text fields prefaced by the specified labels. The user enters data in the text fields or accepts the optional default values that are displayed in the text field. When the user clicks the **OK** button, the `getInputs` method reads the data from the dialog box and closes the dialog box.



Note:

You cannot use a script that contains `getInputs` if you are running the script from the command line and passing the script name to the command line options **-start**, **-script**, **-replay**, or **-noGUI**.

Path

getInputs

Required arguments

fields A sequence of sequence of Strings specifying the text fields in the dialog box. Each inner sequence is a pair of Strings specifying a label that describes a text field along with a default value that appears in the text field. If the field has no default value, you should enter an empty string in the second string in the pair.

Optional arguments

label A String specifying a label to be placed at the top of the dialog box. The default is an empty string, indicating that no label will be shown.

dialogTitle A String specifying the text to be shown in the title bar of the dialog box. The default is "Get Inputs".

Return value

A sequence of Strings representing the data in each of the text fields in the dialog box. If the user clicks the **Cancel** button, the method returns a sequence of `None` objects.

Exceptions

None.

`getWarningReply`

This method is used to obtain a reply from a user from a warning dialog box. The method displays a modal warning dialog box with a message and standard buttons. The user clicks the one of the standard buttons, the `getWarningReply` returns the corresponding button value and closes the dialog box. You can use a "\n" separator in the message argument to provide a multi-line message in the warning dialog box; for example,

```
getWarningReply('Out of disk space!\nOkay to continue', (YES,NO))
```



Note:

You cannot use a script that contains `getWarningReply` if you are running the script from the command line and passing the script name to the command line options **-start**, **-script**, **-replay**, or **-noGUI**.

Path

getWarningReply

Required arguments

message A String specifying the message in the warning dialog box.

buttons A Sequence of standard buttons to be displayed in the warning dialog box. Permissible values are YES, NO, YES_TO_ALL and CANCEL.

Optional arguments

None.

Return value

The button that the user clicks.

Exceptions

None.

Status commands

These functions display status information.

milestone

This function displays a string in the prompt area.

Path

milestone

Required arguments

message	A String specifying the text to display.

Optional arguments

None.

Return value

None.

Exceptions

None.

milestone

This function displays a percentage complete message in the prompt area.

Path

milestone

Required arguments

message	A String specifying the text to display.
percent	An Int specifying the percentage complete.

Optional arguments

None.

Return value

None.

Exceptions

None.

milestone

This function displays a message in the prompt area indicating the number done out of a total. The form of the message is `operation: object nn out of nn`

Path

milestone

Required arguments

message	A String specifying the operation.
object	A String specifying the object.
done	An Int specifying the number being processed.
total	An Int specifying the total number.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Callback commands

These functions execute Python commands or functions.

addImportCallback

This function defines a function to be called when a specified Abaqus/CAE module is imported. You cannot specify a custom module.

For more information, see *An example of a callback function*.

Path

addImportCallback

Required arguments

moduleName A String specifying the name of a specified Abaqus/CAE module.

callback A Python function to be called. The interface definition of the callback function is :

```
def functionName(moduleName, userData)
```

- *moduleName* is a String.
- *userData* is the object passed as the *userData* argument to the `addImportCallback` method.

Optional arguments

userData Any Python object or `None`. This object is passed to the callback function.

Return value

`None`.

Exceptions

None.

removeImportCallback

This function removes a callback added in `addImportCallback`.

Path

removeImportCallback

Required arguments

callback A Python function to be called; it must be the same as the *callback* argument specified in the original call to `addImportCallback`.

userData Any Python object or `None`; it must be the same as the *userData* argument specified in the original call to `addImportCallback`.

Optional arguments

None.

Return value

`None`.

Exceptions

None.

Method callback command

The `methodCallback` module provides functions that allow you to register a callback that will be invoked when certain Abaqus Scripting Interface commands are about to be executed. This callback allows you to augment the standard behavior of Abaqus Scripting Interface commands.

For example, in the following script the `myCallback` function is executed when the `writeInput` method of an object whose type is JobType (in other words, any Job object) is about to be called:

```
import methodCallback
from job import JobType

def myCallback(callingObject, arguments, keywordArguments, userData):

    print('An input file  is about to be written.')

methodCallback.addCallback(JobType, 'writeInput', myCallback)
```

Access

```
import methodCallback
```

addCallback

This method adds a callback function that will be invoked when certain Abaqus/CAE commands are about to be executed.

Path

methodCallback.addCallback

Required arguments

caller

An object or type object specifying which object will trigger the callback function to be invoked or the Symbolic Constant ALL_TYPES.

methodName

A String specifying the name of the method on the caller that will trigger the callback function to be invoked or the Symbolic Constant ALL_METHODS.

callback

A Python function to be called when a command matching the specified caller and method name is about to be executed. The interface definition of the callback function is:

```
def functionName(callingMethod, args, keywordArgs, userData)
```

where:

- *callingMethod* is the method that called this function.
- *args* is the sequence of non-keyword arguments that was passed to the calling method.
- *keywordArgs* is the dictionary of keyword arguments that was passed to the calling method.
- *userData* is the object passed as the *userData* argument to the `addCallback` method.

Optional arguments

userData

Any type of data. This data will be passed to the callback function. The default value is `None`.

callAfter

A Boolean specifying that the callback should be called after the method has executed (instead of before the method is called). The default value is False, which indicates that the callback should be called before the method has executed.

If *callAfter*=True, you can also access the return value of the command from within the callback by including the following statement:

```
returnValue = getMethodReturnValue()
```

The getMethodReturnValue function is in the global namespace of the callback function.

Return value

`None.`

Exceptions

None.

removeCallback

This method removes a callback added by the `addCallback` method. To successfully remove a callback, all arguments must exactly match those used when the callback was added.

Path

methodCallback.removeCallback

Required arguments

caller

An object or type object specifying which object will trigger the callback function to be invoked or the Symbolic Constant ALL_TYPES.

methodName

A String specifying the name of the method on the caller that will trigger the callback function to be invoked or the Symbolic Constant ALL_METHODS.

callback

A Python function to be called when a command matching the specified caller and method name is about to be executed. The interface definition of the callback function is:

```
def functionName(callingMethod, args, keywordArgs, userData)
```

where:

- *callingMethod* is the method that called this function.
- *args* is the sequence of nonkeyword arguments that was passed to the calling method.
- *keywordArgs* is the dictionary of keyword arguments that was passed to the calling method.
- *userData* is the object passed as the *userData* argument to the `removeCallback` method.

Optional arguments

userData

Any type of data. This data will be passed to the callback function. The default value is `None`.

Return value

`None`.

Exceptions

None.

Delete object callback command

The deleteObjectCallback module provides methods that allow you to register a callback that will be invoked when specified Abaqus Scripting Interface objects are about to be deleted. This callback allows you to examine an object before it is deleted.

For example, in the following script the myCallback function is executed when any Job object is about to be deleted:

```
import deleteObjectCallback

def myCallback(path, userData):

    print('About to delete', path)

deleteObjectCallback.addCallback(path='mdb.jobs[*]')
```

Access

```
import deleteObjectCallback
```

deleteObjectCallback

This method adds a callback function that will be invoked when the specified Abaqus Scripting Interface objects are about to be deleted. The callback is invoked only when the object is deleted using the Python statement del object. The callback is not invoked when the object is deleted using an Abaqus Scripting Interface command such as mdb.models[name].parts[name].deleteFeature().

Path

deleteObjectCallback.addCallback

Required arguments

callback

A Python function to be called when an object matching the specified path is about to be deleted. The interface definition of the callback function is:

```
def functionName(objectPath, userData)
```

where:

- *objectPath* is the path to the object about to be deleted.
- *userData* is the object passed as the *userData* argument to the addCallback method.

path

A String specifying the path to an object or the SymbolicConstant ANY. You can include wildcards in the path to specify a pattern to be matched. Examples of valid paths are:

```
path='mdb.models[*]'
path=”mdb.models['Axle*'].parts[*]”
path='mdb.models[*].materials[*]'
```

Optional arguments

userData

Any type of data. This data will be passed to the callback function. The default value is None.

includeChildren A Boolean specifying that the callback should be called if an object owned by the object specified by the *path* argument is about to be deleted. The default value is False.

Return value

`None.`

Exceptions

None.

Upgrade script commands

The `upgradeScript` module allows you to upgrade scripts from one release of Abaqus to a later release. The `upgradeScript` module allows you to upgrade over several releases at one time. It also allows you to upgrade only kernel commands, only GUI commands, or both.

Access

```
import upgradeScript
```

upgradeScript

This function can be used to upgrade a directory, a file, or a list of both directories and files. You can preview the changes before you choose to upgrade the file. A user interface to this function is available via the Plug-ins menu. For more information, see *Upgrading a script*. You can also use a simpler interface from the command line to upgrade scripts. For more information, see the summary of changed commands at the end of this guide.

Path

upgradeScript.upgradeScript

Required arguments

fileNames A String or sequence of Strings specifying the files or directories to upgrade. If a directory path is found, it will be searched for files with the suffix `.py` or `.guiLog`. If *searchSubdirectories* is True, any directories found within a directory will also be searched.

Optional arguments

searchSubdirectories A Boolean specifying whether to search any subdirectories (if the file name is a directory). If *searchSubdirectories* is True, the command will search the subdirectories of any directory in *fileNames*. If *searchSubdirectories* is False, the command will search the directory but not the subdirectories. The default value is True.

backup A Boolean specifying whether to make backups of the files. The default value is True. This argument is ignored if *preview* is set.

preview A Boolean specifying whether to preview the changes to the file instead of changing the file. If *preview* is True, display a preview of the file changes. The default value is True.

By default, the preview is displayed in a web browser. The *diffExecutable* argument allows you to specify a different application in which to preview the changes.

diffExecutable A String specifying the application used to display the differences between the script and the upgraded script. The default value is an empty string, and the differences are displayed by a web browser. Examples of values for *diffExecutable* are winmerge and diff. This argument is ignored if *preview* is False.

logFileName A String specifying the name of the log file where any warnings and changes found during the upgrade are written. The default value is `asiUpgrade.log`.

fromVersion A String specifying the Abaqus release from which to upgrade. The default value is EARLIEST.

toVersion A String specifying the Abaqus release to which to upgrade. The default value is LATEST.

scriptType A SymbolicConstant specifying the type of scripting commands to be upgraded. Possible values are KERNEL, GUI, or BOTH. If *scriptType* is KERNEL only Abaqus Scripting Interface commands will be upgraded; if *scriptType* is GUI only Abaqus GUI Toolkit commands will be upgraded. The default value is BOTH.

Return value

The number of changes made or, if *preview* was used, the number of changes that would have been made.

Exceptions

None.

Customization commands

The following command is used to record a user-defined command in the Abaqus journal file.

journalMethodCall

This function may be used by a user-defined command to record itself in the Abaqus journal file.

For example,

```
def setValues( self, **kargs ):
        for arg,value in kargs.items():
            setattr(self, arg, value)
        from abaqus import journalMethodCall
        objPath = '%s[%r]' % (self.reposPath, self.name)
        journalMethodCall(objPath, 'setValues', (), kargs)
```



Note:

Your command should not call `journalMethodCall` if the command changes the mdb using built-in Abaqus Scripting Interface commands, because built-in commands are journaled by default. A command that changes the mdb customData is one example of a command that should call `journalMethodCall`.

Path

journalMethodCall

Required arguments

objectPath	A String specifying the path to the object.
methodName	A String specifying the name of the method.
args	A sequence specifying the positional arguments to be written to the journal file.
kargs	A Python dict object specifying the keyword arguments to be written to the journal file.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

View object

The *Session* and *Viewport*View objects store view settings for custom (both predefined and user-defined) views. The paradigm used to define a view is based on a camera analogy. Similar to taking a photograph with a camera, features such as camera position, view direction, orientation, depth of field, and projection are specified to transform three-dimensional views to the screen.

The *Layer*View objects store a transformation matrix used to position the contents of the *Layer* within a viewport.

Access

```
session.viewports[name].layers[name].view
session.viewports[name].view
session.views[name]
```

View(...)

This method creates a View object.



Note:

All dimensions and coordinates are specified in the model coordinate system.



Note:

This method cannot be used to create a View for a *Layer* object.

Path

session.View

Required arguments

name

A String specifying the name of the view (also used as the repository key). Possible values are 'Front', 'Back', 'Top', 'Bottom', 'Left', `Right', 'Iso', 'User-1', 'User-2', 'User-3', and 'User-4'. The object member associated with this argument is a SymbolicConstant. Possible values of the *name* member are:FRONT, BACK, TOP, BOTTOM, LEFT, RIGHT, ISO, USER1, USER2, USER3, and USER4.

nearPlane

A Float specifying the distance from the camera to the near clipping plane. Possible values are *nearPlane* > 0.0.

farPlane

A Float specifying the distance from the camera to the far clipping plane when *farPlaneMode* =SPECIFY. Possible values are *farPlane* > *nearPlane*.

width

A Float specifying the width of the front clipping plane. Possible values are *width* > 0.0.

height

A Float specifying the height of the front clipping plane. Possible values are *height* > 0.0.

projection

A SymbolicConstant specifying the projection mode. Possible values are PERSPECTIVE and PARALLEL.

cameraPosition A sequence of three Floats specifying the camera position.

cameraUpVector A sequence of three Floats specifying the camera's up vector (the screen's positive *Y*-axis). The initial value is (0, 0, 0).

cameraTarget A sequence of three Floats specifying the center of the scene.

viewOffsetX A Float specifying the amount to pan the model in the screen *X*-direction as a fraction of the viewport width. A positive value pans the model to the right. A negative value pans the model to the left.

The*viewOffsetX* and *viewOffsetY* arguments allow you to pan the view without changing the position of the camera or the target (*cameraPosition* and *cameraTarget* arguments to the `View` method). The resulting change in the view allows you to pan a perspective display without producing an apparent rotation of the model.

viewOffsetY A Float specifying the amount to pan the model in the screen *Y*-direction as a fraction of the viewport height. A positive value pans the model upward. A negative value pans the model downward.

autoFit A Boolean specifying whether the view is auto-fit when applied.

Optional arguments

movieMode A Boolean specifying whether or not the camera is in movie mode. The default value is OFF.

Return value

A View object.

Exceptions

RangeError.

`fitView(...)`

This method scales the displayable object (such as a part, the assembly, or an *X*–*Y* plot) to fit the viewport.

Required arguments

None.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This is typically only used when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value
None.

Exceptions
None.

next(...)

This method restores the view in the viewport to the next view setting in the list. (There is a list of eight views stored for each viewport.) If there is no next view, no action is taken.



Note:

This method is not available for a *Layer*View.

Required arguments
None.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This is typically only used when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value
None.

Exceptions
None.

pan(...)

This method pans the view in the viewport using absolute, not relative, mode.

Required arguments
None.

Optional arguments

xFraction A Float specifying the amount to pan the model in the screen *X*-direction as a fraction of the viewport width. A positive value pans the model to the right. A negative value pans the model to the left. The default value is 0.0.

yFraction A Float specifying the amount to pan the model in the screen *Y*-direction as a fraction of the viewport height. A positive value pans the model upward. A negative value pans the model downward. The default value is 0.0.

asMovie A Boolean specifying the alternate mode of the `pan` view manipulation should be used. The default value is OFF. This argument is ignored for a *Layer*View.

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

`previous(...)`

This method restores the view in the viewport to the previous view setting in the list. (There is a list of eight views stored for each viewport.) If there is no previous view, no action is taken.



Note:

This method is not available for a *Layer*View.

Required arguments

None.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

`rotate(...)`

This method rotates the view in the viewport. If a center of rotation has been previously specified and *asMovie* is OFF then this method will honor that rotation center.

Required arguments

None.

Optional arguments

xAngle A Float specifying the degrees to rotate about the *X*-axis. The default value is 0.0.

yAngle A Float specifying the degrees to rotate about the *Y*-axis. The default value is 0.0.

zAngle A Float specifying the degrees to rotate about the*Z*-axis. The default value is 0.0.

mode A SymbolicConstant specifying the rotation mode. Possible values are:

- TOTAL : Set the view to (0, 0, 1), then rotate about the screen's axes (an absolute rotation).
- SCREEN : Rotate incrementally about the screen's axes (a relative rotation).
- MODEL : Rotate incrementally about the model's axes (a relative rotation).

The default value is MODEL.

asMovie A Boolean specifying the alternate mode of the `rotate` view manipulation should be used. The default value is OFF.

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

`setLayerTransform(...)`

This method modifies the transformation used to position a *Layer*.



Note:

This method is not available for *Session* and *Viewport*Views.

Required arguments

None.

Optional arguments

layerTransform A sequence of 16 Floats specifying the transformation matrix.

options A View object from which the view settings are to be copied. If the *layerTransform* argument is also supplied to `setLayerTransform`, it will override the values in the View object specified by *view*.

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

`setProjection(...)`

This method modifies the appearance of three-dimensional models in the viewport. Choosing PERSPECTIVE makes a model appear more realistic by decreasing the apparent size of features that are farther away from the viewing point.



Note:

This method is not available for a *Layer*View.

Required arguments

projection A SymbolicConstant specifying the projection mode. Possible values are PERSPECTIVE and PARALLEL.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

RangeError.

`setRotationCenter(...)`

This method sets the center of rotation to the specified location.

Required arguments

rotationCenter A sequence of a String and an Int specifying a part instance name and a node label or a sequence of 3 Floats specifying a point.

Optional arguments

None.

Return value

`None.`

Exceptions

TypeError: rotationCenter cannot be set using a part instance and node label unless the displayed object is an ODB

setValues(...)

This method modifies the View object.



Note:

This method is not available for a *Layer*View.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the `View` method, except for the *name* and *autoFit* arguments. In addition, `setValues` has the following optional arguments:

options
A View object from which the view settings are to be copied. If other arguments are also supplied to `setValues`, they will override the values in the View object specified by *view*.

drawImmediately
A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The member is False by default.

fieldOfViewAngle
A Float specifying the viewing angle of the camera. Possible values are 0.0 < filedOfViewAngle <180.0

farPlaneMode
A SymbolicConstant specifying how the distance from the camera to the far clipping plane is set. Possible values are AUTOCOMPUTE and SPECIFY.

Return value

`None.`

Exceptions

RangeError.

setViewpoint(...)

This method sets the camera's position in the viewport.



Note:

This method is not available for a *Layer*View.

Required arguments

viewVector

A sequence of three Floats specifying the viewing vector (from the camera to the origin of the model).

Optional arguments

cameraUpVector

A sequence of three Floats specifying the camera's up vector (the screen's positive *Y*-axis). The initial value is (0, 0, 0).

drawImmediately

A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

`zoom(...)`

This method magnifies the view in the viewport.

Required arguments

zoomFactor

A Float specifying the amount to zoom. Possible values are $0.000001 \leq$ *zoomFactor* ≤ 1000000. A *zoomFactor* less than one reduces the image. A *zoomFactor* greater than one enlarges the image.

Optional arguments

mode

A SymbolicConstant specifying the way the zoom is executed. Possible values are:

- ABSOLUTE : Execute `fitView`, then zoom.
- RELATIVE : Zoom from the current camera settings.

The default value is ABSOLUTE.

asMovie

A Boolean specifying the alternate mode of the `zoom` view manipulation should be used. The default value is OFF. This argument is ignored for a *Layer*View.

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

RangeError.

`zoomRectangle(...)`

This method fills the viewport with the graphics located within the given rectangle.

Required arguments

point1 A pair of Floats specifying the*X*- and *Y*-coordinates of one corner of the rectangle in fractions of the viewport width and height.

point2 A pair of Floats specifying the*X*- and *Y*-coordinates of the other corner of the rectangle in fractions of the viewport width and height.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This argument is typically used only when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

Members

The View object has members with the same names and descriptions as the arguments to the *View* method.

In addition, the View object has the following members:

displayedObjectScreenWidth A Float specifying the width in viewport millimeters of the bounding rectangle around the viewport contents. This value does not include annotations or symbols and it is not clipped to the size of the viewport window.

displayedObjectScreenHeight A Float specifying the height in viewport millimeters of the bounding rectangle around the viewport contents. This value does not include annotations or symbols and it is not clipped to the size of the viewport window.

layerTransform

A tuple of Floats specifying a transformation matrix used to position the contents of the *Layer* within a viewport.

XY commands

XY commands are used to plot X–Y data and store its display attributes and to write X–Y data to an ASCII file.

In this section:

- *XYData object*
- *Area object*
- *AreaStyle object*
- *Axis object*
- *AxisData object*
- *Chart object*
- *CurveOptions object*
- *DefaultChartOptions object*
- *DefaultPlot object*
- *Legend object*
- *LineStyle object*
- *QuantityType object*
- *SymbolStyle object*
- *TextStyle object*
- *Title object*
- *XYCurve object*
- *XYPlot object*
- *XYReportOptions object*
- *Session object*
- *writeXYReport*

XYData object

The XYData object is used to store values and attributes associated with XYData type objects.

XYData objects can be created using the methods described below. The methods accessed via the Session object cause the XYData object to be added to the session.xyData repository.

Temporary XYData objects will be created if no name is supplied. Temporary XYData objects will be added to the session.xyData repository but automatically deleted when they are not used anymore. Temporary XYData objects are also created as a result of math operations found in the `abaqusMath` module.

Access

```
import visualization
session.charts[name].axes1[i].axisData.curves[i].data
session.charts[name].axes2[i].axisData.curves[i].data
session.charts[name].curves[name].data
session.curves[name].data
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.data
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.data
import odbAccess
session.odbs[name].userData.xyDataObjects[name]
session.xyDataObjects[name]
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i].data
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i].data
session.xyPlots[name].charts[name].curves[name].data
session.xyPlots[name].curves[name].data
```

XYData(...)

This method creates an XYData object from a sequence of *X*–*Y* data pairs.

Path

session.XYData

xyPlot.XYData

Required arguments

data
A sequence of pairs of Floats specifying the *X*–*Y* data pairs.

Optional arguments

name
The repository key. If the name is not supplied while creating the XYData object using xyPlot.XYData, a default name in the form _temp#_ is generated and the XYData object is temporary. (This argument is required if the method is accessed from the session object.)

sourceDescription
A String specifying the source of the *X*–*Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription A String specifying the content of the *X–Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription A String specifying additional information about the *X–Y* data (e.g., “for whole model”). The default value is an empty string.

legendLabel A String specifying the label to be used in the legend. The default value is the name of the XYData object.

xValuesLabel A String specifying the label for the X-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

yValuesLabel A String specifying the label for the Y-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

axis1QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the X -axis1-values.

axis2QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the Y -axis2-values.

Return value

An XYData object.

Exceptions

InvalidNameError.

XYData(...)

This method creates an XYData object by copying an existing XYData object.

Path

session.odbs[*name*].userData.XYData

session.XYData

xyPlot.XYData

Required arguments

objectToCopy An XYData object to be copied.

Optional arguments

The optional arguments are the same as the optional arguments to the `XYData` method.

Return value

An XYData object.

Exceptions

InvalidNameError.

XYDataFromFile(...)

This method creates an XYData object from data in an ASCII file.

Path

session.XYDataFromFile

xyPlot.XYDataFromFile

Required arguments

fileName A String specifying the name of the file from which the *X*–*Y* data will be read.

Optional arguments

name The repository key. If the name is not supplied, a default name in the form _temp#_ is generated and the XYData object is temporary.

sourceDescription A String specifying the source of the *X*–*Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription A String specifying the content of the *X*–*Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription A String specifying additional information about the *X*–*Y* data (e.g., “for whole model”). The default value is an empty string.

legendLabel A String specifying the label to be used in the legend. The default value is the name of the XYData object.

xValuesLabel A String specifying the label for the X-values. This value may be overridden if the *X*–*Y* data are combined with other *X*–*Y* data. The default value is an empty string.

yValuesLabel A String specifying the label for the Y-values. This value may be overridden if the *X*–*Y* data are combined with other *X*–*Y* data. The default value is an empty string.

axis1QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the X -axis1- values.

axis2QuantityType A *QuantityType* object specifying the *QuantityType* object associated to the Y -axis2- values.

xField An Int specifying the field from which the *X*-data will be read. Fields are delimited by spaces, tabs, or commas. The default value is 1.

yField

An Int specifying the field from which the *Y*-data will be read. Fields are delimited by spaces, tabs, or commas. The default value is 2.

skipFrequency

An Int specifying how often data rows will be skipped. A *skipFrequency* of 1 means skip every other row. The first row is always read. Possible values are *skipFrequency* $\geq$ 0. The default value is 0 (data are read from every row).

Return value

An XYData object.

Exceptions

InvalidNameError and RangeError.

XYDataFromHistory(...)

This method creates an XYData object by reading history data from an Odb object.

Path

session.XYDataFromHistory

xyPlot.XYDataFromHistory

Required arguments

odb

An Odb object specifying the output database from which data will be read.

outputVariableName

A String specifying the output variable from which the *X*–*Y* data will be read.

steps

A sequence of Strings specifying the names of the steps from which data will be extracted.

Optional arguments

name

The repository key. If the name is not supplied, a default name in the form _temp#_ is generated and the XYData object is temporary (this argument is required if the method is accessed from the session object).

sourceDescription

A String specifying the source of the *X*–*Y* data (for example, “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription

A String specifying the content of the *X*–*Y* data (for example, “field 1 vs. field 2”). The default value is an empty string.

positionDescription

A String specifying additional information about the *X*–*Y* data (for example, “for whole model”). The default value is an empty string.

legendLabel A String specifying the label to be used in the legend. The default value is the name of the XYData object.

skipFrequency An Int specifying how often data frames will be skipped. If *skipFrequency*=1, Abaqus will skip every other frame. The first frame is always read. Possible values are *skipFrequency*≥ 0. The default value is 0 (data are read from every frame).

numericForm A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_VAL_AT_ANGLE. The default value is REAL.

complexAngle A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_VAL_AT_ANGLE. The default value is 0.

stepTuple A tuple of Integers specifying the steps to include when extracting data.

Return value

An XYData object.

Exceptions

InvalidNameError and RangeError.

xyDataListFromField(...)

This method creates a list of XYData objects by reading field data from an Odb object.

Path

session.xyDataListFromField

xyPlot.xyDataListFromField

Required arguments

odb An Odb object specifying the output database from which data will be read.

outputPosition A SymbolicConstant specifying the position from which output will be read. Possible values are ELEMENT_CENTROID, ELEMENT_NODAL, INTEGRATION_POINT, and NODAL.

variable A tuple of tuples containing the descriptions of variables for which to extract data from the field. Each tuple specifies the following:

- Variable label: A String specifying the variable; for example, 'U'.
- Variable output position: A SymbolicConstant specifying the output position. Possible values are ELEMENT_CENTROID, ELEMENT_FACE, ELEMENT_NODAL, GENERAL_PARTICLE, INTEGRATION_POINT, NODAL, WHOLE_ELEMENT, WHOLE_MODEL, WHOLE_PART_INSTANCE, and WHOLE_REGION.

- Refinement: A tuple specifying the refinement. If the refinement tuple is omitted, data are written for all components and invariants (if applicable). This element is required if the location dictionary (the following element in the tuple) is included. The refinement tuple contains the following:
  - Type: A SymbolicConstant specifying the type of refinement. Possible values are INVARIANT and COMPONENT.
  - Label: A String specifying the invariant or the component; for example, 'Mises' or 'S22'.
- Location: An optional Dictionary specifying the location. The dictionary contains pairs of the following:
  - A String specifying the category selection label.
  - A String specifying the section point label.

For example,

```
variable= ('S',INTEGRATION_POINT, ( (COMPONENT, 'S22' ), ), )
variable= (('S',INTEGRATION_POINT, ((COMPONENT, 'S11' ), ), ),
           ('U',NODAL,((COMPONENT, 'U1'),)),)
variable= (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises' ), ),
           {'shell < STEEL > < 3 section points >':'SNEG,
                                    (fraction = -1.0)', }), )
```

Optional arguments

At least one of the following arguments *elementSets*, *elementLabels*, *nodeSets*, and *nodeLabels* is required.

elementSets

A sequence of Strings specifying element sets or a String specifying a single element set.

elementLabels

A sequence of expressions specifying element labels per part instance in the model. Each part instance element expression is a sequence of a String specifying the part instance name and a sequence of element expressions; for example, `(('partInstance1',(1,'7','3:15;3'),), ('partInstance2','8'),))`. The element expressions can be any of the following:

- An Int specifying a single element label; for example, `1`.
- A String specifying a single element label; for example, `'7'`.
- A String specifying a sequence of element labels; for example, `'3:5'` and `'3:15:3'`.

nodeSets

A sequence of Strings specifying node sets or a String specifying a single node set.

nodeLabels

A sequence of expressions specifying node labels per part instance in the model. Each part instance node expression is a sequence of a String specifying the part instance name and a sequence of node expressions; for example, `(('partInstance1',(1,'7','3:15;3'),), ('partInstance2','8'),))`. The node expressions can be any of the following:

- An Int specifying a single node label; for example, `1`.
- A String specifying a single node label; for example, `'7'`.
- A String specifying a sequence of node labels; for example, `'3:5'` and `'3:15:3'`.

numericForm A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_VAL_AT_ANGLE. The default value is REAL.

complexAngle A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_VAL_AT_ANGLE. The default value is 0.

operator A SymbolicConstant specifying the mathematical, trigonometric, logarithmic, exponential, or other operations. Possible values are ADD, SUBTRACT, MULTIPLY, DIVIDE, POWER, MINIMUM, MAXIMUM, AVERAGE, RANGE, SRSS, ABSOLUTE, UNARY_NEGATIVE, COSINE, HYPERBOLIC_COSINE, INVERSE_COSINE, SINE, HYPERBOLIC_SINE, INVERSE_SINE, TANGENT, HYPERBOLIC_TANGENT, INVERSE_TANGENT, EXPONENTIAL, NATURAL_LOG, LOG, SQUARE_ROOT, NORMALIZE, DEG2RAD, RAD2DEG, SMOOTH, SWAP, AVERAGE_ALL, MAXIMUM_ENVELOPE, MINIMUM_ENVELOPE, and RANGE_ALL. If no value is defined, no operation will be performed on the data, and the data will be saved as is.

Return value

A list of XYData objects.

Exceptions

InvalidNameError and RangeError.

XYDataFromFreeBody(...)

This method creates a list of XYData objects by computing free body data from an Odb object.

Path

session.XYDataFromFreeBody

xyPlot.XYDataFromFreeBody

Required arguments

odb An Odb object specifying the output database from which data will be read.

Optional arguments

force A boolean indicating whether to compute the force. The default is ON.

moment A boolean indicating whether to compute the moment. The default is OFF.

heatFlowRate A boolean indicating whether to compute the heat flow rate resultant magnitude. It is extracted only for viewcut based freebodies. The default is OFF.

resultant A boolean indicating whether to compute the resultant. It applies only to *force* and *moment*. The default is ON.

comp1
A boolean indicating whether to compute the first component. It applies only to *force* and *moment*. The default is OFF.

comp2
A boolean indicating whether to compute the second component. It applies only to *force* and *moment*. The default is OFF.

comp3
A boolean indicating whether to compute the third component. It applies only to *force* and *moment*. The default is OFF.

Return value

A list of XYData objects.

Exceptions

InvalidNameError and RangeError.

XYDataFromShellThickness(...)

This method creates a list of XYData objects by reading through the thickness field data from an Odb object.

Path

xyPlot.XYDataFromShellThickness

Required arguments

odb
An Odb object specifying the output database from which data will be read.

outputPosition
A SymbolicConstant specifying the position from which output will be read. Possible values are ELEMENT_CENTROID, ELEMENT_NODAL, INTEGRATION_POINT, and NODAL.

variable
A tuple of tuples containing the descriptions of variables for which to extract data from the field. Each tuple specifies the following:

- Variable label: A String specifying the variable; for example, 'U'.
- Variable output position: A SymbolicConstant specifying the output position. Possible values are ELEMENT_CENTROID, ELEMENT_FACE, ELEMENT_NODAL, GENERAL_PARTICLE, INTEGRATION_POINT, NODAL, WHOLE_ELEMENT, WHOLE_MODEL, WHOLE_PART_INSTANCE, and WHOLE_REGION.
- Refinement: A tuple specifying the refinement. If the refinement tuple is omitted, data are written for all components and invariants (if applicable). This element is required if the location dictionary (the following element in the tuple) is included. The refinement tuple contains the following:
  - Type: A SymbolicConstant specifying the type of refinement. Possible values are INVARIANT and COMPONENT.
  - Label: A String specifying the invariant or the component; for example, 'Mises' or 'S22'.
- Location: An optional Dictionary specifying the location. The dictionary contains pairs of the following:

- A String specifying the category selection label.
- A String specifying the section point label.

For example,

```
variable= ('S',INTEGRATION_POINT, ( (COMPONENT, 'S22' ), ), )
variable= (('S',INTEGRATION_POINT, ((COMPONENT, 'S11' ), ), ),
           ('U',NODAL,((COMPONENT, 'U1'),)),)
variable= (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises' ), ),
           {'shell < STEEL > < 3 section points >':'SNEG,
                                    (fraction = -1.0)', }), )
```

Optional arguments

At least one of the following arguments *elementSets*, *elementLabels*, *nodeSets*, and *nodeLabels* is required.

elementSets
A sequence of Strings specifying element sets or a String specifying a single element set.

elementLabels
A sequence of expressions specifying element labels per part instance in the model. Each part instance element expression is a sequence of a String specifying the part instance name and a sequence of element expressions; for example, `(('partInstance1',(1,'7','3:15;3'),), ('partInstance2','8'),))`. The element expressions can be any of the following:

- An Int specifying a single element label; for example, `1`.
- A String specifying a single element label; for example, `'7'`.
- A String specifying a sequence of element labels; for example, `'3:5'` and `'3:15:3'`.

nodeSets
A sequence of Strings specifying node sets or a String specifying a single node set.

nodeLabels
A sequence of expressions specifying node labels per part instance in the model. Each part instance node expression is a sequence of a String specifying the part instance name and a sequence of node expressions; for example, `(('partInstance1',(1,'7','3:15;3'),), ('partInstance2','8'),))`. The node expressions can be any of the following:

- An Int specifying a single node label; for example, `1`.
- A String specifying a single node label; for example, `'7'`.
- A String specifying a sequence of node labels; for example, `'3:5'` and `'3:15:3'`.

numericForm
A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_VAL_AT_ANGLE. The default value is REAL.

complexAngle
A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_VAL_AT_ANGLE. The default value is 0.

Return value

A list of XYData objects.

Exceptions

InvalidNameError and RangeError.

XYDataFromPath(...)

This method creates an XYData object from path information.

Path

session.XYDataFromPath

xyPlot.XYDataFromPath

Required arguments

path A Path object to use in *X–Y* data generation.

name A String specifying the repository key:

- for *session* 'name' is required argument and for *xyPlot* 'name' is optional argument.

includeIntersections A Boolean specifying whether to include *X–Y* data for the intersections between the path and element faces or edges. The default value is False.

shape A SymbolicConstant specifying the model shape to use. Possible values are UNDEFORMED and DEFORMED.

pathStyle A SymbolicConstant specifying the path style. Possible values are PATH_POINTS and UNIFORM_SPACING.

numIntervals An Int specifying the number of uniform-spacing intervals. The default value is 10.

labelType A SymbolicConstant specifying the *X*-label type to use. Possible values are NORM_DISTANCE, SEQ_ID, TRUE_DISTANCE, TRUE_DISTANCE_X, TRUE_DISTANCE_Y, TRUE_DISTANCE_Z, X_COORDINATE, Y_COORDINATE and Z_COORDINATE.

Optional arguments

viewport A String specifying the viewport name or an Int specifying the viewport id from which to obtain values. The default is the current viewport.

removeDuplicateXYPairs A Boolean specifying whether to remove duplicate XY values from the final result. The default value is True.

includeAllElements A Boolean specifying whether to include elements which do not lie in the direction of the path. The default value is False.

step An Int identifying the step from which to obtain values. The default value is the current step.

frame

An Int identifying the frame from which to obtain values. The default value is the current frame.

variable

A tuple of tuples containing the descriptions of variables for which to extract data along the path. The default value is the current variable. Each tuple specifies the following:

- Variable label: A String specifying the variable; for example, 'U'.
- Variable output position: A SymbolicConstant specifying the output position. Possible values are ELEMENT_CENTROID, ELEMENT_FACE, ELEMENT_NODAL, GENERAL_PARTICLE, INTEGRATION_POINT, NODAL, WHOLE_ELEMENT, WHOLE_MODEL, WHOLE_PART_INSTANCE, and WHOLE_REGION.
- Refinement: A tuple specifying the refinement. If the refinement tuple is omitted, data are written for all components and invariants (if applicable). This element is required if the location dictionary (the following element in the tuple) is included. The refinement tuple contains the following:
  - Type: A SymbolicConstant specifying the type of refinement. Possible values are INVARIANT and COMPONENT.
  - Label: A String specifying the invariant or the component; for example, 'Mises' or 'S22'.
- Location: An optional Dictionary specifying the location. The dictionary contains pairs of the following:
  - A String specifying the category selection label.
  - A String specifying the section point label.

For example,

```
variable= ('S',INTEGRATION_POINT, ( (COMPONENT, 'S22' ), ),
)
variable= (('S',INTEGRATION_POINT, ((COMPONENT, 'S11' ), ),
),
           ('U',NODAL,((COMPONENT, 'U1'),)),)
variable= (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises' ),
),
           {'shell < STEEL > < 3 section points >':'SNEG,
                                  (fraction = -1.0)', }),
)
```

deformedMag

A tuple of three Floats specifying the deformation magnitude in the *X*-, *Y*-, and *Z*-planes. The default value is (1, 1, 1).

numericForm

A SymbolicConstant specifying the numeric form in which to display results that contain complex numbers. Possible values are COMPLEX_MAGNITUDE, COMPLEX_PHASE, REAL, IMAGINARY, and COMPLEX_VAL_AT_ANGLE. The default value is REAL.

complexAngle

A Float specifying the angle (in degrees) at which to display results that contain complex numbers when *numericForm*=COMPLEX_VAL_AT_ANGLE. The default value is 0.

projectOntoMesh	A Boolean to specify whether to consider the data points that do not lie on or inside the mesh. The default value is False.
projectionTolerance	A Float specifying the tolerance value for the projected distance considered for the data extraction when *projectOntoMesh*= True. The default value is 0.

Return value

If *variable* specified has one fieldoutput: Returns an XYData object.

If *variable* specified has more than one fieldoutputs: Returns list of XYData objects.

Exceptions

If *path* is invalid:	ErrorPathNotFound: Path not found.
If *viewport* is invalid:	ErrorCurrentVPNotFound: Current viewport not found.
If *step* and/or *frame* are invalid:	ErrorInvalidUserStepAndFrame: The user step and frame specified have not been defined.
If the *variable* argument is empty:	ErrorNoVarInPathExtract: No variable selection for XY data extraction from path.
If the specified output variable is not available in the output database:	ErrorUnavailableSelectedVariable: The selected variable is not available for the current frame.
If the specified output variable cannot be used to obtain *X–Y* data:	ErrorUnusableVarInPathExtract: Specified variable cannot be used in XY data extraction from path.
If the SymbolicConstant specifying the refinement type is invalid:	ErrorUnsupportedRefinementType: Unsupported refinement type.
If the label specifying the refinement invariant or component is invalid:	ErrorInvalidRefinementSpecification: Invalid refinement specification.
If *deformedMag* does not contain three Floats:	ErrorDeformedMagTupleInPathExtract: Deformed magnification tuple must contain X, Y and Z values.

`save()`

This method saves a temporary XYData. The name of the XYData is changed to "XYData-#". If the XYData is already saved, nothing is done.

Arguments

None.

Return value
`None.`

Exceptions

None.

`setValues(...)`

This method modifies the XYData object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *XYData* method, except for the *name* argument.

Return value
`None.`

Exceptions

None.

Members

The XYData object has members with the same names and descriptions as the arguments to the *XYData* method.

In addition, the XYData object has the following members:

sourceType A SymbolicConstant specifying the source type of the XYData object. Possible values are FROM_ODB, FROM_KEYBOARD, FROM_ASCII_FILE, FROM_OPERATION, and FROM_USER_DEFINED.

fileName A String specifying the source file name of the XYData object.

description A String specifying the complete description of the XYData object.

Area object

The Area object is used to display a rectangular area in an XYPlot. The Area object has no constructor. Area objects are automatically created whenever a *XYPlot*, *Chart*, PlotTitle, or Legend objects are created.

Access

```
import visualization
session.charts[name].area
session.charts[name].gridArea
session.charts[name].legend.area
session.defaultChartOptions.gridArea
session.defaultChartOptions.legend.area
session.defaultPlot.area
session.defaultPlot.title.area
session.xyPlots[name].area
session.xyPlots[name].charts[name].area
session.xyPlots[name].charts[name].gridArea
session.xyPlots[name].charts[name].legend.area
session.xyPlots[name].title.area
```

`setValues(...)`

This method modifies the Area object.

Required arguments

None.

Optional arguments

area

An Area object from which attributes are to be copied.

style

An *AreaStyle* object.

border

A *LineStyle* object.

positionMethod

A SymbolicConstant specifying how the area is positioned. Possible values are AUTO_ALIGN and MANUAL. The default value is AUTO_ALIGN.

alignment

A SymbolicConstant specifying the relative position of the area in its parent when *positionMethod*=AUTO_ALIGN. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT

- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

sizeMethod
A SymbolicConstant specifying how the area size is defined. Possible values are AUTOMATIC and MANUAL. The default value is AUTOMATIC.

originOffset
A pair of Floats specifying the X- and Y-offsets of the origin as a fraction of the available area. The *originOffset* argument is ignored unless *positionMethod*=MANUAL. The default value is (-1, 0). The valid range for each float is (0, 1).

widthScale
A Float specifying the scale as a fraction of the width of the available area when the sizeMethod=MANUAL. The valid range is (0, 1). The default value is 1.0.

heightScale
A Float specifying the scale as a fraction of the height of the available area when the *sizeMethod*=MANUAL. The valid range is (0, 1). The default value is 1.0.

inset
A Boolean specifying whether the area is inset or occupies a reserved area. The default value is OFF.

pl
A Float specifying the left padding of the area in mm. The default value is 1.0.

pr
A Float specifying the right padding of the area in mm. The default value is 1.0.

pt
A Float specifying the top padding of the area in mm. The default value is 1.0.

pb
A Float specifying the bottom padding of the area in mm. The default value is 1.0.

Return value
`None.`

Exceptions
RangeError.

Members
The Area object can have the following members:

inset
A Boolean specifying whether the area is inset or occupies a reserved area. The default value is OFF.

positionMethod
A SymbolicConstant specifying how the area is positioned. Possible values are AUTO_ALIGN and MANUAL. The default value is AUTO_ALIGN.

alignment
A SymbolicConstant specifying the relative position of the area in its parent when *positionMethod*=AUTO_ALIGN. Possible values are:

- BOTTOM_LEFT
- BOTTOM_CENTER
- BOTTOM_RIGHT
- CENTER_LEFT
- CENTER
- CENTER_RIGHT
- TOP_LEFT
- TOP_CENTER
- TOP_RIGHT

The default value is BOTTOM_LEFT.

sizeMethod
A SymbolicConstant specifying how the area size is defined. Possible values are AUTOMATIC and MANUAL. The default value is AUTOMATIC.

width
A Float specifying the width of the area in mm. The default value is 1.0.

height
A Float specifying the height of the area in mm. The default value is 1.0.

widthScale
A Float specifying the scale as a fraction of the width of the available area when the sizeMethod=MANUAL. The valid range is (0, 1). The default value is 1.0.

heightScale
A Float specifying the scale as a fraction of the height of the available area when the *sizeMethod*=MANUAL. The valid range is (0, 1). The default value is 1.0.

pl
A Float specifying the left padding of the area in mm. The default value is 1.0.

pr
A Float specifying the right padding of the area in mm. The default value is 1.0.

pt
A Float specifying the top padding of the area in mm. The default value is 1.0.

pb
A Float specifying the bottom padding of the area in mm. The default value is 1.0.

style
An *AreaStyle* object specifying whether and how to fill the area.

border
A *LineStyle* object specifying whether and how to draw the border of the area.

origin
A pair of Floats specifying the X- and Y-offsets in millimeters from the lower-left corner of the XYPlot.

originOffset
A pair of Floats specifying the X- and Y-offsets of the origin as a fraction of the available area. The *originOffset* argument is ignored unless *positionMethod*=MANUAL. The default value is (-1, 0). The valid range for each float is (0, 1).

AreaStyle object

The AreaStyle object is used to define how areas are to be filled when drawing XY-plot objects.

AreaStyle objects are automatically created whenever an *Area* object is created. AreaStyle objects can be created using the methods described below.

Access

```
import visualization
session.charts[name].area.style
session.charts[name].gridArea.style
session.charts[name].legend.area.style
session.charts[name].tagAreaStyle
session.defaultChartOptions.areaStyle
session.defaultChartOptions.gridArea.style
session.defaultChartOptions.legend.area.style
session.defaultChartOptions.tagAreaStyle
session.defaultPlot.area.style
session.defaultPlot.title.area.style
session.xyPlots[name].area.style
session.xyPlots[name].charts[name].area.style
session.xyPlots[name].charts[name].gridArea.style
session.xyPlots[name].charts[name].legend.area.style
session.xyPlots[name].charts[name].tagAreaStyle
session.xyPlots[name].title.area.style
```

AreaStyle(...)

This method creates an AreaStyle.

Path

session.AreaStyle

xyPlot.AreaStyle

Required arguments

None.

Optional arguments

color A String specifying the color to be used when filling an area with this AreaStyle object. The default value is "White".

fill A Boolean specifying whether to fill the area when using this AreaStyle. The default value is ON.

style A SymbolicConstant specifying the area pattern style to be used when filling an area using this AreaStyle. The default value is SOLID.

Return value

An AreaStyle object.

Exceptions

ColorError

setValues(...)

This method modifies the AreaStyle object.

Required arguments

None.

Optional arguments

color A String specifying the color to be used when filling an area with this AreaStyle object. The default value is "White".

fill A Boolean specifying whether to fill the area when using this AreaStyle. The default value is ON.

style A SymbolicConstant specifying the area pattern style to be used when filling an area using this AreaStyle. The default value is SOLID.

Return value

`None.`

Exceptions

None.

Members

The AreaStyle object has members with the same names and descriptions as the arguments to the *setValues* method.

Axis object

The Axis object is used to store the display attributes of axes. Axes objects are automatically created when adding *XYCurve* objects to a *Chart* object.

Access

```
import visualization
session.charts[name].axes1[i]
session.charts[name].axes2[i]
session.defaultChartOptions.defaultAxis1Options
session.defaultChartOptions.defaultAxis2Options
session.xyPlots[name].charts[name].axes1[i]
session.xyPlots[name].charts[name].axes2[i]
```

`setValues(...)`

This method modifies the Axis object.

Required arguments

None.

Optional arguments

axis

An Axis object from which attributes are to be copied.

labelFrequency

An Int specifying the frequency of the labels with respect to the tick marks. The default value is 1.

labelPlacement

A SymbolicConstant specifying how labels are placed on the axis. Possible values are:

- NONE, specifying that no labels are displayed.
- INSIDE, specifying that the labels are placed on the inside of the axis.
- OUTSIDE, specifying that the labels are placed on the outside of the axis.

The default value is INSIDE.

labelStyle

A *TextStyle* object specifying the text properties to be used when displaying axis labels.

lineStyle

A *LineStyle* object specifying the line properties used to display the axis.

placement

A SymbolicConstant specifying the placement of the axis on the grid. Possible values are:

- MIN_EDGE, specifying that the axis is placed at the minimum edge - for an abscissa at the bottom, for an ordinate to the left.
- MAX_EDGE, specifying that the axis is placed at the maximum edge - for an abscissa at the top, for an ordinate at the right.
- MIN_MAX_EDGE, specifying that the axis is placed at the minimum edge - for an abscissa at the bottom, for an ordinate to the left - and repeated without labels and title at the maximum edge.

- CENTER, specifying that the axis is placed at the center of the grid.

The default value is MIN_MAX_EDGE.

tickLength
A Float specifying the length of the ticks in mm. The default value is 2.0.

tickPlacement
A SymbolicConstant specifying how tick marks are placed on the axis. Possible values are:

- NONE, specifying that no tick marks are displayed.
- INSIDE, specifying that the tick marcks are placed on the inside of the axis.
- OUTSIDE, specifying that the tick marcks are placed on the outside of the axis.
- ACROSS, specifying that the tick marcks are placed across the axis.

The default value is INSIDE.

tickStyle
A *LineStyle* object specifying the line properties to be used when displaying axis ticks.

titleStyle
A *TextStyle* object specifying the text properties to be used when displaying the axis title.

Return value
`None.`

Exceptions
None.

Members
The Axis object can have the following members:

labelFrequency
An Int specifying the frequency of the labels with respect to the tick marks. The default value is 1.

tickLength
A Float specifying the length of the ticks in mm. The default value is 2.0.

placement
A SymbolicConstant specifying the placement of the axis on the grid. Possible values are:

- MIN_EDGE, specifying that the axis is placed at the minimum edge - for an abscissa at the bottom, for an ordinate to the left.
- MAX_EDGE, specifying that the axis is placed at the maximum edge - for an abscissa at the top, for an ordinate at the right.
- MIN_MAX_EDGE, specifying that the axis is placed at the minimum edge - for an abscissa at the bottom, for an ordinate to the left - and repeated without labels and title at the maximum edge.
- CENTER, specifying that the axis is placed at the center of the grid.

The default value is MIN_MAX_EDGE.

tickPlacement
A SymbolicConstant specifying how tick marks are placed on the axis. Possible values are:

- NONE, specifying that no tick marks are displayed.
- INSIDE, specifying that the tick marcks are placed on the inside of the axis.
- OUTSIDE, specifying that the tick marcks are placed on the outside of the axis.
- ACROSS, specifying that the tick marcks are placed across the axis.

The default value is INSIDE.

labelPlacement A SymbolicConstant specifying how labels are placed on the axis. Possible values are:

- NONE, specifying that no labels are displayed.
- INSIDE, specifying that the labels are placed on the inside of the axis.
- OUTSIDE, specifying that the labels are placed on the outside of the axis.

The default value is INSIDE.

axisData An *AxisData* object specifying the numerical data of the axis.

lineStyle A *LineStyle* object specifying the line properties used to display the axis.

labelStyle A *TextStyle* object specifying the text properties to be used when displaying axis labels.

titleStyle A *TextStyle* object specifying the text properties to be used when displaying the axis title.

tickStyle A *LineStyle* object specifying the line properties to be used when displaying axis ticks.

AxisData object

The AxisData object is used to store the data attributes of axes. An AxisData object is automatically created when creating an *Axis* object.

Access

```
import visualization
session.charts[name].axes1[i].axisData
session.charts[name].axes2[i].axisData
session.defaultChartOptions.defaultAxis1Options.axisData
session.defaultChartOptions.defaultAxis2Options.axisData
session.xyPlots[name].charts[name].axes1[i].axisData
session.xyPlots[name].charts[name].axes2[i].axisData
```

`setValues(...)`

This method modifies the AxisData object.

Required arguments

None.

Optional arguments

axisData

An AxisData object from which attributes are to be copied.

labelFormat

A SymbolicConstant specifying how tick labels are formatted. Possible values are AUTOMATIC, DECIMAL, SCIENTIFIC, and ENGINEERING. The default value is AUTOMATIC.

labelNumDigits

An Int specifying the number of significant digits displayed for the labels. Possible values are 1 to 7. The default value is 2.

scale

A SymbolicConstant specifying the type of scale to use for the axis. Possible values are:

- LINEAR, specifying tickmarks and labels are linearly distributed.
- LOG, specifying tickmarks and labels are logarithmically distributed.
- DB, specifying tickmarks and labels are distributed on a decibel scale.
- DB2, specifying tickmarks and labels are distributed on a 2*decibel scale.

The default value is LINEAR.

dbReference

A Float specifying the reference value for decibel computation. The default value is 1.0.

minAutoCompute

A Boolean specifying whether or not to use the automatically computed minimum value for the axis. The default value is ON.

minValue

A Float specifying the minimum value when *minAutoCompute* is false. By default, *minValue* is set to *minAutoValue*.

maxAutoCompute
A Boolean specifying whether or not to use the automatically computed maximum value for the axis. The default value is ON.

maxValue
A Float specifying the maximum value when *maxAutoCompute* is false. By default, *maxValue* is set to *maxAutoValue*.

tickMode
A SymbolicConstant specifying the type of scale to use for the axis. Possible values are:

- AUTOCOMPUTE, specifying tickmarks and labels are automatically computed.
- INCREMENT, specifying tickmarks and labels are defined by a given increment.
- TOTAL_NUMBER, specifying tickmarks and labels are defined by the total number of ticks.

The default value is AUTOCOMPUTE.

tickIncrement
A Float specifying the increment of the major tick marks on the axis when *tickMode* = INCREMENT. Valid values are 0 <*tickIncrement*. The default value is computed based on the results of the automatic method and the range being plotted. When the *scale* is set to LOG, the tickIncrement is interpreted as a value per decade and should be between 0.05 and 1.

tickCount
An Int specifying the number of major tick marks on the axis when *tickMode* =TOTAL_NUMBER. Possible values are 0 ≤*tickCount*≤ 30. The default value is computed based on the range of the axis. When the *scale* is set to LOG, the tickCount is interpreted as the number of ticks per decade and acceptable values are 1, 4, 8, and 17.

minorTickCount
An Int specifying the number the number of minor tick marks between major ticks. Possible values are 0 ≤*minorTickCount*≤ 20. When the *scale* is set to LOG, the minorTickCount is interpreted as the number of ticks per decade and limited to 0, 1, 4, 8, and 17. The default value is 1.

title
A String specifying the title of the axis. By default, the title is set to the *systemTitle*.

useSystemTitle
A Boolean specifying whether the title to use for the axis title is system defined or user defined. The default value is ON.

Return value
`None.`

Exceptions
RangeError.

Members
The AxisData object can have the following members:

dbReference
A Float specifying the reference value for decibel computation. The default value is 1.0.

direction

A SymbolicConstant specifying the direction of the axis. Possible values are ABSCISSA and ORDINATE.

labelFormat

A SymbolicConstant specifying how tick labels are formatted. Possible values are AUTOMATIC, DECIMAL, SCIENTIFIC, and ENGINEERING. The default value is AUTOMATIC.

labelNumDigits

An Int specifying the number of significant digits displayed for the labels. Possible values are 1 to 7. The default value is 2.

maxAutoCompute

A Boolean specifying whether or not to use the automatically computed maximum value for the axis. The default value is ON.

maxAutoValue

A Float specifying the maximum value when *maxAutoCompute* is true.

maxValue

A Float specifying the maximum value when *maxAutoCompute* is false. By default, *maxValue* is set to *maxAutoValue*.

maxShownValue

A Float specifying the current maximum value displayed for this axis. This value is different from *maxAutoValue* or *maxValue* when the axis is being transformed (zoom or pan).

minAutoCompute

A Boolean specifying whether or not to use the automatically computed minimum value for the axis. The default value is ON.

minAutoValue

A Float specifying the minimum value when *minAutoCompute* is true.

minValue

A Float specifying the minimum value when *minAutoCompute* is false. By default, *minValue* is set to *minAutoValue*.

minShownValue

A Float specifying the current minimum value displayed for this axis. This value is different from *minAutoValue* or *minValue* when the axis is being transformed (zoom or pan).

minorTickCount

An Int specifying the number the number of minor tick marks between major ticks. Possible values are $0 \leq$ *minorTickCount* ≤ 20. When the *scale* is set to LOG, the minorTickCount is interpreted as the number of ticks per decade and limited to 0, 1, 4, 8, and 17. The default value is 1.

scale

A SymbolicConstant specifying the type of scale to use for the axis. Possible values are:

- LINEAR, specifying tickmarks and labels are linearly distributed.
- LOG, specifying tickmarks and labels are logarithmically distributed.
- DB, specifying tickmarks and labels are distributed on a decibel scale.
- DB2, specifying tickmarks and labels are distributed on a 2*decibel scale.

The default value is LINEAR.

tickMode

A SymbolicConstant specifying the type of scale to use for the axis. Possible values are:

- AUTOCOMPUTE, specifying tickmarks and labels are automatically computed.

- INCREMENT, specifying tickmarks and labels are defined by a given increment.
- TOTAL_NUMBER, specifying tickmarks and labels are defined by the total number of ticks.

The default value is AUTOCOMPUTE.

tickCount

An Int specifying the number of major tick marks on the axis when *tickMode* =TOTAL_NUMBER. Possible values are $0 \leq tickCount \leq 30$. The default value is computed based on the range of the axis. When the *scale* is set to LOG, the tickCount is interpreted as the number of ticks per decade and acceptable values are 1, 4, 8, and 17.

tickCountShown

An Int specifying the number of major ticks effectively shown. This value takes zoom, pan and rounding into account.

tickIncrement

A Float specifying the increment of the major tick marks on the axis when *tickMode* = INCREMENT. Valid values are $0 < tickIncrement$. The default value is computed based on the results of the automatic method and the range being plotted. When the *scale* is set to LOG, the tickIncrement is interpreted as a value per decade and should be between 0.05 and 1.

tickIncrementShown

A Float specifying the shown tick increment of the major ticks. This value takes zoom/pan into account.

useSystemTitle

A Boolean specifying whether the title to use for the axis title is system defined or user defined. The default value is ON.

curves

An *XYCurveArray* object specifying a read-only sequence of Curve objects associated to this axis.

quantityType

A *QuantityType* object specifying the quantity type: i.e. the physical dimension and associated label of the data represented by this axis.

tickValues

A tuple of Floats specifying the read-only major tick values shown.

tickLabels

A tuple of Strings specifying the read-only major tick labels shown.

systemTitle

A String specifying the system title. The system title is based on the *quantityType* of the axis and associated curves.

title

A String specifying the title of the axis. By default, the title is set to the *systemTitle*.

Chart object

The Chart object is used to display *XYCurve* objects. A Chart object is automatically created when creating an *XYPlot* object

Access

```
import visualization
session.charts[name]
session.xyPlots[name].charts[name]
```

autoColor(...)

This method distributes the colors on all curves displayed in the chart using the color palette defined by the xyColors object.

Required arguments

None.

Optional arguments

lines	A Boolean defining whether color distribution affects curve lines.
symbols	A Boolean defining whether color distribution affects curve symbols.

Return value

`None.`

Exceptions

None.

autoSymbol()

This method distributes the symbols on all curves displayed in the chart.

Arguments

None.

Return value

`None.`

Exceptions

None.

fitCurves()

This method resets the transform of the chart. It cancels any zoom or pan action.

Arguments

None.

Return value

`None.`

Exceptions

None.

getAxis1(...)

This method returns the *Axis* object used for displaying the Axis1 of the *XYCurve* specified by name or object or used for the given *QuantityType* object.

Required arguments

curve	The name or the *XYCurve* object associated to the *Axis* object.
quantityType	The *QuantityType* object associated to the *Axis* object.

Optional arguments

None.

Return value

An *Axis* object.

Exceptions

If the given *XYCurve* is not used in the Chart.

XypError: Curve not found:

If both arguments are specified.	TypeError: Specify curve or quantityType; too many arguments; expected 1, got 2.
If the given *QuantityType* is not used in the Chart.	ValueError: QuantityType not found

getAxis2(...)

This method returns the *Axis* object used for displaying the Axis2 of the *XYCurve* specified by name or object or used for the given *QuantityType* object.

Required arguments

curve	The name or the *XYCurve* object associated to the *Axis* object.
quantityType	The *QuantityType* object associated to the *Axis* object.

Optional arguments

None.

Return value

An *Axis* object.

Exceptions

If the given *XYCurve* is not used in the Chart.

XypError: Curve not found:

If both arguments are specified.	TypeError: Specify curve or quantityType; too many arguments; expected 1, got 2.
If the given *QuantityType* is not used in the Chart.	ValueError: QuantityType not found

moveAxisUp(...)

This method moves the relative position of the given *Axis* object up in the axis sequence of the Chart.

Required arguments

axis	The *Axis* object to be moved.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

moveAxisDown(...)

This method moves the relative position of the given *Axis* object down in the axis sequence of the Chart.

Required arguments

axis	The *Axis* object to be moved.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

removeCurve(...)

This method removes the given *XYCurve* from the Chart.

Required arguments

curve
The XYCurve name or the *XYCurve* object or a sequence of XYCurve names or *XYCurve* objects to be removed from the Chart.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the Chart object.

Required arguments

None.

Optional arguments

chart
A Chart object from which attributes are to be copied.

curvesToPlot
A sequence of Strings specifying the names of the curves to plot. In addition to this type, the argument can also be one of the following:

- A String specifying the name of the curve to plot.
- An *XYCurve* object specifying the curve to plot.
- A sequence of *XYCurve* objects specifying the curves to plot (as returned by the `curveSet` method).

aspectRatio
A Float specifying the aspect ratio of the grid area. A value of -1 specifies that the gridArea will take up all available space. The default value is −1.

transform
A sequence of Floats specifying a transformation matrix used to scale or pan along the axes of the Chart.

view A *View* object.

useQuantityType A Boolean specifying whether to use the *QuantityType* to associate curves with axes. The default value is ON.

Return value

`None.`

Exceptions

RangeError.

Members

The Chart object can have the following members:

name A String specifying the name of the Chart object.

useQuantityType A Boolean specifying whether to use the *QuantityType* to associate curves with axes. The default value is ON.

aspectRatio A Float specifying the aspect ratio of the grid area. A value of -1 specifies that the gridArea will take up all available space. The default value is −1.

curves A repository of *XYCurve* objects specifying a repository of *XYCurve* objects to display in the Chart.

axes1 An *AxisArray* object specifying a read-only sequence of axis objects displayed as axes1 - the abscissa for a Cartesian chart.

axes2 An *AxisArray* object specifying a read-only sequence of axis objects displayed as axes2 - the ordinate for a Cartesian chart.

area An *Area* object specifying position, padding, background and borders of the chart.

gridArea An *Area* object specifying how to display the grid area.

legend A *Legend* object specifying the attributes for the legend of the chart.

majorAxis1GridStyle A *LineStyle* object specifying the line properties to be used when drawing major gridlines along axis 1.

majorAxis2GridStyle A *LineStyle* object specifying the line properties to be used when drawing major gridlines along axis 2.

minorAxis1GridStyle A *LineStyle* object specifying the line properties to be used when drawing minor gridlines along axis 1.

minorAxis2GridStyle

A *LineStyle* object specifying the line properties to be used when drawing minor gridlines along axis 2.

tagTextStyle

A *TextStyle* object specifying the text properties to be used when creating tags.

tagAreaStyle

An *AreaStyle* object specifying the area properties to be used when creating tags.

tagBorder

A *LineStyle* object specifying the tag area border properties to be used when creating tags.

transform

A tuple of Floats specifying a transformation matrix used to scale or pan along the axes of the Chart.

CurveOptions object

The CurveOptions object is used to define the curve properties when drawing curves.

CurveOptions objects can be created using the methods described below.

Access

```
import visualization
session.charts[name].axes1[i].axisData.curves[i].curveOptions
session.charts[name].axes2[i].axisData.curves[i].curveOptions
session.charts[name].curves[name].curveOptions
session.curves[name].curveOptions
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.curveOptions
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.curveOptions
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i]\
.curveOptions
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i]\
.curveOptions
session.xyPlots[name].charts[name].curves[name].curveOptions
session.xyPlots[name].curves[name].curveOptions
```

CurveOptions(...)

This method creates a CurveOptions object.

Path

session.CurveOptions

xyPlot.CurveOptions.

Required arguments

None.

Optional arguments

symbolColor

A String specifying the color to be used when drawing a marker with this CurveOptions object. The default value is "White."

showSymbol

A Boolean specifying whether to draw the marker when using this CurveOptions object. The default value is ON.

symbolMarker

A SymbolicConstant specifying the marker type be used when drawing symbols using this CurveOptions object. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI

- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER
- POINT

The default value is FILLED_CIRCLE.

symbolSize A Float specifying the marker size to be used when drawing markers using this CurveOptions object. The default value is 2.0.

lineColor A String specifying the color to be used when drawing a line with this CurveOptions object. The default value is "White."

showLine A Boolean specifying whether to draw the line when using this CurveOptions object. The default value is ON.

lineStyle A SymbolicConstant specifying the line style to be used when drawing lines using this CurveOptions object. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

lineThickness A Float specifying the line thickness in millimeters to be used when drawing lines using this CurveOptions object. The default value is 0.2.

Return value

A CurveOptions object.

Exceptions

ColorError.

`setValues(...)`

This method modifies the SymbolStyle object.

Required arguments

None.

Optional arguments

symbolColor A String specifying the color to be used when drawing a marker with this CurveOptions object. The default value is "White."

showSymbol A Boolean specifying whether to draw the marker when using this CurveOptions object. The default value is ON.

symbolMarker

A SymbolicConstant specifying the marker type be used when drawing symbols using this CurveOptions object. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER
- POINT

The default value is FILLED_CIRCLE.

symbolSize

A Float specifying the marker size to be used when drawing markers using this CurveOptions object. The default value is 2.0.

lineColor

A String specifying the color to be used when drawing a line with this CurveOptions object. The default value is "White."

showLine

A Boolean specifying whether to draw the line when using this CurveOptions object. The default value is ON.

lineStyle

A SymbolicConstant specifying the line style to be used when drawing lines using this CurveOptions object. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

lineThickness

A Float specifying the line thickness in millimeters to be used when drawing lines using this CurveOptions object. The default value is 0.2.

Return value

`None.`

Exceptions

None.

Members

The CurveOptions object has members with the same names and descriptions as the arguments to the *setValues* method.

DefaultChartOptions object

The DefaultChartOptions object is used to hold on default chart and axis attributes. The DefaultChartOptions object attributes are used whenever Chart or Axis are created. A DefaultChartOptions object is automatically created when opening a session.

Access

```
import visualization
session.defaultChartOptions
```

`setValues(...)`

This method modifies the DefaultChartOptions object.

Required arguments

None.

Optional arguments

areaStyle

An *AreaStyle* object specifying an *AreaStyle* used to hold on to the default display properties for the chart area.

aspectRatio

A Float specifying the default aspect ratio of the grid area. A value of -1 specifies that the gridArea will take up all available space. The default value is −1.

defaultAxis1Options

An *Axis* object specifying an *Axis* object used to hold on to the default properties for direction 1 axes—the abscissa for a Cartesian chart.

defaultAxis2Options

An *Axis* object specifying an *Axis* object used to hold on to the default properties for direction 2 axes—the ordinate for a Cartesian chart.

gridArea

An *Area* object specifying how to display the grid area by default.

legend

A *Legend* object specifying the default attributes for the legend of the chart.

majorAxis1GridStyle

A *LineStyle* object specifying the default line properties to be used when drawing major gridlines along axis 1.

majorAxis2GridStyle

A *LineStyle* object specifying the default line properties to be used when drawing major gridlines along axis 2.

minorAxis1GridStyle

A *LineStyle* object specifying the default line properties to be used when drawing minor gridlines along axis 1.

minorAxis2GridStyle

A *LineStyle* object specifying the default line properties to be used when drawing minor gridlines along axis 2.

tagAreaStyle An *AreaStyle* object specifying the default area properties to be used when creating tags.

tagBorder A *LineStyle* object specifying the default tag area border properties to be used when creating tags.

tagTextStyle A *TextStyle* object specifying the default text properties to be used when creating tags.

useQuantityType A Boolean specifying whether to use the *QuantityType* to associate curves with axes. The default value is ON.

Return value

`None.`

Exceptions

None.

Members

The DefaultChartOptions object has members with the same names and descriptions as the arguments to the `setValues` method.

DefaultPlot object

The DefaultPlot object is used to hold on default plot attributes. The DefaultPlot object attributes are used whenever an *XYPlot* object is created. A DefaultPlot object is automatically created when opening a session.

Access

```
import visualization
session.defaultPlot
```

Members

The DefaultPlot object can have the following members:

area

An *Area* object specifying an *Area* used to hold on to the default display properties for the plot area.

title

A *Title* object specifying a *Title* object used to hold on to the default properties of the XY-Plot title.

Legend object

The Legend object is used to store the display attributes of the chart legend. A legend object is automatically created when creating a *Chart* object.

Access

```
import visualization
session.charts[name].legend
session.defaultChartOptions.legend
session.xyPlots[name].charts[name].legend
```

`setValues(...)`

This method modifies the Legend object.

Required arguments

None.

Optional arguments

legend	A Legend object from which attributes are to be copied.
show	A Boolean specifying whether to show the legend. The default value is ON.
showMinMax	A Boolean specifying whether to display the minimum and maximum values. The default value is OFF.
title	A String specifying the title to appear on the legend. The default value is an empty string.
numberFormat	A SymbolicConstant specifying how the minimum and maximum values are formatted. Possible values are AUTOMATIC, DECIMAL, SCIENTIFIC, and ENGINEERING. The default value is AUTOMATIC.
numDigits	An Int specifying the number of significant digits displayed for the minimum and maximum values. Possible values are 1 to 7. The default value is 2.
textStyle	A *TextStyle* object specifying the text properties used to display the legend text.
titleStyle	A *TextStyle* object specifying the text properties used to display the legend title.

Return value

`None.`

Exceptions

None.

Members

The Legend object can have the following members:

numberFormat
A SymbolicConstant specifying how the minimum and maximum values are formatted. Possible values are AUTOMATIC, DECIMAL, SCIENTIFIC, and ENGINEERING. The default value is AUTOMATIC.

numDigits
An Int specifying the number of significant digits displayed for the minimum and maximum values. Possible values are 1 to 7. The default value is 2.

show
A Boolean specifying whether to show the legend. The default value is ON.

showMinMax
A Boolean specifying whether to display the minimum and maximum values. The default value is OFF.

area
An *Area* object specifying the area of the legend.

textStyle
A *TextStyle* object specifying the text properties used to display the legend text.

title
A String specifying the title to appear on the legend. The default value is an empty string.

titleStyle
A *TextStyle* object specifying the text properties used to display the legend title.

LineStyle object

The LineStyle object is used to define the line style to be used for drawing XY-Plot objects.

LineStyle objects can be created using the methods described below.

Access

```
import visualization
session.charts[name].area.border
session.charts[name].axes1[i].axisData.curves[i].lineStyle
session.charts[name].axes1[i].lineStyle
session.charts[name].axes1[i].tickStyle
session.charts[name].axes2[i].axisData.curves[i].lineStyle
session.charts[name].axes2[i].lineStyle
session.charts[name].axes2[i].tickStyle
session.charts[name].curves[name].lineStyle
session.charts[name].gridArea.border
session.charts[name].legend.area.border
session.charts[name].majorAxis1GridStyle
session.charts[name].majorAxis2GridStyle
session.charts[name].minorAxis1GridStyle
session.charts[name].minorAxis2GridStyle
session.charts[name].tagBorder
session.curves[name].lineStyle
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.lineStyle
session.defaultChartOptions.defaultAxis1Options.lineStyle
session.defaultChartOptions.defaultAxis1Options.tickStyle
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.lineStyle
session.defaultChartOptions.defaultAxis2Options.lineStyle
session.defaultChartOptions.defaultAxis2Options.tickStyle
session.defaultChartOptions.gridArea.border
session.defaultChartOptions.legend.area.border
session.defaultChartOptions.majorAxis1GridStyle
session.defaultChartOptions.majorAxis2GridStyle
session.defaultChartOptions.minorAxis1GridStyle
session.defaultChartOptions.minorAxis2GridStyle
session.defaultChartOptions.tagBorder
session.defaultPlot.area.border
session.defaultPlot.title.area.border
session.xyPlots[name].area.border
session.xyPlots[name].charts[name].area.border
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i]\
.lineStyle
session.xyPlots[name].charts[name].axes1[i].lineStyle
session.xyPlots[name].charts[name].axes1[i].tickStyle
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i]\
.lineStyle
session.xyPlots[name].charts[name].axes2[i].lineStyle
session.xyPlots[name].charts[name].axes2[i].tickStyle
session.xyPlots[name].charts[name].curves[name].lineStyle
session.xyPlots[name].charts[name].gridArea.border
session.xyPlots[name].charts[name].legend.area.border
session.xyPlots[name].charts[name].majorAxis1GridStyle
```

```
session.xyPlots[name].charts[name].majorAxis2GridStyle
session.xyPlots[name].charts[name].minorAxis1GridStyle
session.xyPlots[name].charts[name].minorAxis2GridStyle
session.xyPlots[name].charts[name].tagBorder
session.xyPlots[name].curves[name].lineStyle
session.xyPlots[name].title.area.border
```

LineStyle(...)

This method creates a LineStyle.

Path

session.LineStyle

xyPlot.LineStyle

Required arguments

None.

Optional arguments

color A String specifying the color to be used when drawing a line with this LineStyle object. The default value is "White".

show A Boolean specifying whether to draw the line when using this LineStyle. The default value is ON.

style A SymbolicConstant specifying the line style to be used when drawing lines using this LineStyle. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

thickness A Float specifying the line thickness in mm to be used when drawing lines using this LineStyle. The default value is 0.2.

Return value

A LineStyle object.

Exceptions

ColorError

setValues(...)

This method modifies the LineStyle object.

Required arguments

None.

Optional arguments

color
A String specifying the color to be used when drawing a line with this LineStyle object. The default value is "White".

show
A Boolean specifying whether to draw the line when using this LineStyle. The default value is ON.

style
A SymbolicConstant specifying the line style to be used when drawing lines using this LineStyle. Possible values are SOLID, DASHED, DOTTED, and DOT_DASH. The default value is SOLID.

thickness
A Float specifying the line thickness in mm to be used when drawing lines using this LineStyle. The default value is 0.2.

Return value

`None.`

Exceptions

None.

Members

The LineStyle object has members with the same names and descriptions as the arguments to the *setValues* method.

QuantityType object

The QuantityType object is used to store attributes defining the physical dimension and label of the quantity type to be associated with an *XYData* or an *AxisData* object.

QuantityType objects are automatically created when creating *XYData* objects from the odb.

QuantityType objects can be created using the methods described below.

Access

```
import visualization
session.charts[name].axes1[i].axisData.curves[i].data\
.axis1QuantityType
session.charts[name].axes1[i].axisData.curves[i].data\
.axis2QuantityType
session.charts[name].axes1[i].axisData.quantityType
session.charts[name].axes2[i].axisData.curves[i].data\
.axis1QuantityType
session.charts[name].axes2[i].axisData.curves[i].data\
.axis2QuantityType
session.charts[name].axes2[i].axisData.quantityType
session.charts[name].curves[name].data.axis1QuantityType
session.charts[name].curves[name].data.axis2QuantityType
session.curves[name].data.axis1QuantityType
session.curves[name].data.axis2QuantityType
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.data.axis1QuantityType
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.data.axis2QuantityType
session.defaultChartOptions.defaultAxis1Options.axisData.quantityType
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.data.axis1QuantityType
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.data.axis2QuantityType
session.defaultChartOptions.defaultAxis2Options.axisData.quantityType
import odbAccess
session.odbs[name].userData.axis1QuantityType
session.odbs[name].userData.axis2QuantityType
import visualization
import xyPlot
session.odbs[name].userData.xyDataObjects[name].axis1QuantityType
session.odbs[name].userData.xyDataObjects[name].axis2QuantityType
session.xyDataObjects[name].axis1QuantityType
session.xyDataObjects[name].axis2QuantityType
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i].data\
.axis1QuantityType
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i].data\
.axis2QuantityType
session.xyPlots[name].charts[name].axes1[i].axisData.quantityType
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i].data\
.axis1QuantityType
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i].data\
.axis2QuantityType
session.xyPlots[name].charts[name].axes2[i].axisData.quantityType
session.xyPlots[name].charts[name].curves[name].data.axis1QuantityType
session.xyPlots[name].charts[name].curves[name].data.axis2QuantityType
```

```
session.xyPlots[name].curves[name].data.axis1QuantityType
session.xyPlots[name].curves[name].data.axis2QuantityType
```

QuantityType(...)

This method creates a QuantityType object.

Path

session.QuantityType

xyPlot.QuantityType

Required arguments

None.

Optional arguments

label A String specifying the label for this quantity type.

type A SymbolicConstant specifying the physical dimension of the axis. Possible values are:

- NONE.
- ACCELERATION.
- ACOUSTIC_INTENSITY.
- ANGLE.
- ANGULAR_MOMENTUM.
- ARC_LENGTH.
- AREA.
- AREA_VELOCITY_SQUARED, specifying "Velocity squared per area".
- BIMOMENT.
- CURVATURE.
- CORIOLIS_LOAD.
- DAMAGE.
- DAMAGE_CRITERION.
- DENSITY.
- DENSITY_ROTATIONAL_ACCELERATION, specifying "Density * Angular acceleration".
- DISPLACEMENT.
- ECURRENT_AREA_TIME, specifying "Time integrated electric current per area".
- ELECTRIC_CHARGE.
- ELECTRIC_CURRENT.
- ELECTRIC_CURRENT_AREA, specifying "Electric current per unit area".
- ELECTRIC_POTENTIAL.
- ENERGY.

- ENERGY_DENSITY.
- ENERGY_RELEASE_RATE.
- EPOTENTIAL_GRADIENT, specifying "Electric potential gradient".
- FREQUENCY.
- FORCE.
- FORCE_VOLUME, specifying "Force per volume".
- HEAT_FLUX.
- HEAT_FLUX_AREA, specifying "Heat flux per area".
- HEAT_FLUX_RATE.
- HEAT_FLUX_VOLUME, specifying "Heat flux per volume".
- LENGTH.
- LINEAR_PRESSURE.
- LUMIN, specifying "Luminous intensity".
- MASS.
- MASS_FLOW_AREA, specifying "Mass flow per area".
- MASS_FLOW_AREA_RATE, specifying "Mass flow rate per area".
- MASS_FLOW_RATE.
- MODE_NUMBER.
- MOMENT.
- NUMBER.
- PATH.
- PHASE.
- POSITION.
- PRESSURE.
- PRESSURE_GRADIENT.
- RATE.
- ROTARY_INERTIA.
- ROTATIONAL_ACCELERATION.
- ROTATIONAL_VELOCITY.
- STATUS.
- STRAIN.
- STRAIN_RATE.
- STRESS.
- STRESS_INTENS_FACTOR, specifying "Stress intensity factor".
- SUBSTANCE, specifying "Amount of substance".
- TEMPERATURE.
- THICKNESS.
- TIME.
- TIME_INCREMENT.

- TIME_HEAT_FLUX, specifying "Time integrated heat flux".
- TIME_HEAT_FLUX_AREA, specifying "Time integrated heat flux per area".
- TIME_VOLUME, specifying "Time integrated volume".
- TIME_VOLUME_FLUX, specifying "Time integrated volume flux per area".
- TWIST.
- VELOCITY.
- VELOCITY_SQUARED.
- VOLUME.
- VOLUME_FLUX.
- VOLUME_FLUX_AREA, specifying "Volume flux per area".
- VOLUME_FRACTION.

The default value is NONE

Return value

A QuantityType object.

Exceptions

None.

`setValues(...)`

This method modifies the QuantityType object.

Required arguments

None.

Optional arguments

The optional arguments to setValues are the same as the arguments to the *`QuantityType`* method.

Return value

`None.`

Exceptions

None.

Members

The QuantityType object has members with the same names and descriptions as the arguments to the *`setValues`* method.

SymbolStyle object

The SymbolStyle object is used to define the marker properties to be used when drawing curves.

SymbolStyle objects can be created using the methods described below.

Access

```
import visualization
session.charts[name].axes1[i].axisData.curves[i].symbolStyle
session.charts[name].axes2[i].axisData.curves[i].symbolStyle
session.charts[name].curves[name].symbolStyle
session.curves[name].symbolStyle
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]\
.symbolStyle
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]\
.symbolStyle
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i]\
.symbolStyle
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i]\
.symbolStyle
session.xyPlots[name].charts[name].curves[name].symbolStyle
session.xyPlots[name].curves[name].symbolStyle
```

SymbolStyle(...)

This method creates a SymbolStyle object.

Path

session.SymbolStyle

xyPlot.SymbolStyle

Required arguments

None.

Optional arguments

color A String specifying the color to be used when drawing a marker with this SymbolStyle object. The default value is "White".

show A Boolean specifying whether to draw the marker when using this SymbolStyle object. The default value is ON.

marker A SymbolicConstant specifying the marker type be used when drawing symbols using this SymbolStyle object. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI

- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI
- CROSS
- XMARKER
- POINT

The default value is FILLED_CIRCLE.

size

A Float specifying the marker size to be used when drawing markers using this SymbolStyle object. The default value is 2.0.

Return value

A SymbolStyle object.

Exceptions

ColorError

`setValues(...)`

This method modifies the SymbolStyle object.

Required arguments

None.

Optional arguments

color

A String specifying the color to be used when drawing a marker with this SymbolStyle object. The default value is "White".

show

A Boolean specifying whether to draw the marker when using this SymbolStyle object. The default value is ON.

marker

A SymbolicConstant specifying the marker type be used when drawing symbols using this SymbolStyle object. Possible values are:

- FILLED_CIRCLE
- FILLED_SQUARE
- FILLED_DIAMOND
- FILLED_TRI
- HOLLOW_CIRCLE
- HOLLOW_SQUARE
- HOLLOW_DIAMOND
- HOLLOW_TRI

- CROSS
- XMARKER
- POINT

The default value is FILLED_CIRCLE.

size

A Float specifying the marker size to be used when drawing markers using this SymbolStyle object. The default value is 2.0.

Return value

`None.`

Exceptions

None.

Members

The SymbolStyle object has members with the same names and descriptions as the arguments to the *`setValues`* method.

TextStyle object

The TextStyle object is used to store the text properties to be used for drawing XY-plot text objects.

TextStyle objects are automatically created when creating a chart or can be created with methods described below.

Access

```
import visualization
session.charts[name].axes1[i].labelStyle
session.charts[name].axes1[i].titleStyle
session.charts[name].axes2[i].labelStyle
session.charts[name].axes2[i].titleStyle
session.charts[name].legend.textStyle
session.charts[name].legend.titleStyle
session.charts[name].tagTextStyle
session.defaultChartOptions.defaultAxis1Options.labelStyle
session.defaultChartOptions.defaultAxis1Options.titleStyle
session.defaultChartOptions.defaultAxis2Options.labelStyle
session.defaultChartOptions.defaultAxis2Options.titleStyle
session.defaultChartOptions.legend.textStyle
session.defaultChartOptions.legend.titleStyle
session.defaultChartOptions.tagTextStyle
session.defaultPlot.title.titleStyle
session.xyPlots[name].charts[name].axes1[i].labelStyle
session.xyPlots[name].charts[name].axes1[i].titleStyle
session.xyPlots[name].charts[name].axes2[i].labelStyle
session.xyPlots[name].charts[name].axes2[i].titleStyle
session.xyPlots[name].charts[name].legend.textStyle
session.xyPlots[name].charts[name].legend.titleStyle
session.xyPlots[name].charts[name].tagTextStyle
session.xyPlots[name].title.titleStyle
```

TextStyle(...)

This method creates a TextStyle.

Path

session.TextStyle

xyPlot.TextStyle

Required arguments

None.

Optional arguments

color

A String specifying the color to be used when drawing text with this TextStyle object. The default value is "White".

show

A Boolean specifying whether to draw the text when using this TextStyle object. The default value is ON.

font A String specifying the name of the font to be used when drawing text with this TextStyle object. The default value is "-*-verdana-medium-r-normal-*-*-120-*-*-p-*-*-*".

rotationAngle A Float specifying the angle in degrees used for displaying the text. The default value is 0.0.

Return value

A TextStyle object.

Exceptions

ColorError

`setValues(...)`

This method modifies the TextStyle object.

Required arguments

None.

Optional arguments

color A String specifying the color to be used when drawing text with this TextStyle object. The default value is "White".

show A Boolean specifying whether to draw the text when using this TextStyle object. The default value is ON.

font A String specifying the name of the font to be used when drawing text with this TextStyle object. The default value is "-*-verdana-medium-r-normal-*-*-120-*-*-p-*-*-*".

rotationAngle A Float specifying the angle in degrees used for displaying the text. The default value is 0.0.

Return value

`None.`

Exceptions

None.

Members

The TextStyle object has members with the same names and descriptions as the arguments to the *`setValues`* method.

Title object

The Title object is used to store the display attributes of the XYPlot title.
A Title object is automatically created when creating a *XYPlot* object.

Access

```
import visualization
session.defaultPlot.title
session.xyPlots[name].title
```

`setValues(...)`

This method modifies the Title object.

Required arguments

None.

Optional arguments

title A Title object from which attributes are to be copied.

text A String specifying the text to appear as a title. By default the title is set to the *XYPlot* object name. The default value is an empty string.

area An *Area* object specifying the area of the title.

useDefault A Boolean specifying whether to show the default title. The default value is OFF.

style A *TextStyle* object specifying the text properties used to display the legend title.

Return value

`None.`

Exceptions

None.

Members

The Title object can have the following members:

useDefault A Boolean specifying whether to show the default title. The default value is OFF.

area An *Area* object specifying the area of the title.

text A String specifying the text to appear as a title. By default the title is set to the *XYPlot* object name. The default value is an empty string.

style A *TextStyle* object specifying the text properties used to display the legend title.

XYCurve object

The XYCurve object is used to plot *X*–*Y* data and to store its display attributes.

Access

```
import visualization
session.charts[name].axes1[i].axisData.curves[i]
session.charts[name].axes2[i].axisData.curves[i]
session.charts[name].curves[name]
session.curves[name]
session.defaultChartOptions.defaultAxis1Options.axisData.curves[i]
session.defaultChartOptions.defaultAxis2Options.axisData.curves[i]
session.xyPlots[name].charts[name].axes1[i].axisData.curves[i]
session.xyPlots[name].charts[name].axes2[i].axisData.curves[i]
session.xyPlots[name].charts[name].curves[name]
session.xyPlots[name].curves[name]
```

Curve(...)

This method creates an XYCurve object from an *XYData* object.

Path

session.Curve

Required arguments

name	A String specifying the repository key.
xyData	An *XYData* object specifying the data for the curve.

Optional arguments

None.

Return value

An XYCurve object.

Exceptions

InvalidNameError.

setValues(...)

This method modifies the XYCurve object.

Required arguments

None.

Optional arguments

displayTypes A sequence of SymbolicConstants that describe how curves are to be displayed. Possible values are LINE and SYMBOL. The default value is LINE.

legendLabel A String specifying the label to be displayed in the legend. By default, the label is system defined.

symbolFrequency An Int specifying the frequency of plotting the markers. Possible values are *symbolFrequency*> 0. If *symbolFrequency*=1, then markers are plotted at every point. The default value is 1.

useDefault A Boolean specifying whether to use the system-supplied legend label. The default value is ON.

Return value

`None.`

Exceptions

None.

Members

The XYCurve object can have the following members:

name A String specifying the repository key.

symbolFrequency An Int specifying the frequency of plotting the markers. Possible values are *symbolFrequency*> 0. If *symbolFrequency*=1, then markers are plotted at every point.

useDefault A Boolean specifying whether to use the system-supplied legend label.

xyData An *XYData* object specifying the data for the curve.

lineStyle A *LineStyle* object specifying the line properties used to display the curve.

curveOptions A *CurveOptions* object specifying the line and symbol properties used to display the curve.

areaStyle An *AreaStyle* object specifying the properties of the area to display the curve.

symbolStyle A *SymbolStyle* object specifying the symbol properties used to display the curve.

legendLabel A String specifying the label to be displayed in the legend. By default, the label is system defined.

displayTypes A tuple of SymbolicConstants that describe how curves are to be displayed. Possible values are LINE and SYMBOL.

XYPlot object

The XYPlot object is used to display *Chart* objects.

Access

```
import visualization
session.xyPlots[name]
```

XYPlot(...)

This method creates an empty XYPlot object.

Path

```
session.XYPlot
```

Required arguments

name	A String specifying the name of the XYPlot object.

Optional arguments

None.

Return value

An XYPlot object.

Exceptions

InvalidNameError.

autoColor(...)

This method distributes the colors on all curves displayed in the XYPlot using the color palette defined by the xyColors *AutoColors* object.

Required arguments

None.

Optional arguments

lines	A Boolean defining whether color distribution affects curve lines.
symbols	A Boolean defining whether color distribution affects curve symbols.

Return value

None.

Exceptions

None.

autoSymbol()

This method distributes the symbols on all curves displayed in the XYPlot.

Arguments

None.

Return value

`None.`

Exceptions

None.

fitCurves()

This method resets the transform of all the charts of the XYPlot object. It cancels any zoom or pan action.

Arguments

None.

Return value

`None.`

Exceptions

None.

next(...)

This method restores the *transform* member of the active *Chart* object to the next setting in the transform list. (There is a list of eight transforms stored for each chart.) If there is no next transform, no action is taken.

Required arguments

None.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This is typically only used when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

previous(...)

This method restores the *transform* member of the active *Chart* object to the previous setting in the transform list. (There is a list of eight transforms stored for each chart.) If there is no next transform, no action is taken.

Required arguments

None.

Optional arguments

drawImmediately A Boolean specifying the viewport should refresh immediately after the command is processed. This is typically only used when writing a script and it is desirable to show intermediate results before the script completes. The default value is False.

Return value

`None.`

Exceptions

None.

setValues(...)

This method modifies the XYPlot object.

Required arguments

None.

Optional arguments

title A *Title* object specifying the title of the XYPlot object.

transform A sequence of Floats specifying a transformation matrix used to scale or pan along the axes of the active *Chart* object of this XYPlot.

Return value

`None.`

Exceptions

None.

Members

The XYPlot object has members with the same names and descriptions as the arguments to the *XYPlot* method.

In addition, the XYPlot object can have the following members:

area

An *Area* object specifying position, padding, background and borders of the XYPlot object.

title

A *Title* object specifying the title of the XYPlot object.

charts

A repository of *Chart* objects.

curves

A repository of *XYCurve* objects.

transform

A tuple of Floats specifying a transformation matrix used to scale or pan along the axes of the active *Chart* object of this XYPlot.

XYReportOptions object

The XYReportOptions object stores settings used by the `writeXYReport` method when you write an *XYData* object to an ASCII file. The XYReportOptions object has no constructor. Abaqus creates the *xyReportOptions* member when you import the Visualization module.

Access

```
import visualization
session.defaultXYReportOptions
session.xyReportOptions
```

`setValues(...)`

This method modifies the XYReportOptions object.

Required arguments

None.

Optional arguments

pageWidth
An Int specifying the number of characters per line of the report file when *pageWidthLimited*=ON. Possible values are *pageWidth*> 0. The default value is 80.

numDigits
An Int specifying the number of significant digits to be included for each data value in the report file. Possible values are 0 ≤*numDigits*≤ 9. The default value is 6.

interpolation
A Boolean specifying whether to perform linear interpolation for missing data values. The default value is OFF.

xyData
A Boolean specifying whether to print the *X*- and *Y*-values of the selected XYData objects. (If *xyData*=OFF, *totals* and *minMax* can still be printed.) The default value is ON.

totals
A Boolean specifying whether to print the sum of the *Y*-values of the selected XYData objects. The default value is OFF.

minMax
A Boolean specifying whether to print the minimum and maximum *X*- and *Y*-values of the selected XYData objects. The default value is OFF.

pageWidthLimited
A Boolean specifying whether the page width is limited. The default value is OFF.

numberFormat
A SymbolicConstant specifying the number format to be used in reporting XYData objects. Possible values are AUTOMATIC, ENGINEERING, and SCIENTIFIC. The default value is ENGINEERING.

layout
A SymbolicConstant specifying the format used in reporting the XYData objects. Possible values are SINGLE_TABLE and SEPARATE_TABLES. The default value is SINGLE_TABLE.

Return value

`None.`

Exceptions

RangeError.

If *xyData*, *total*, and *minMax* are all OFF:	At least one of the data print methods must be selected

Members

The XYReportOptions object has members with the same names and descriptions as the arguments to the *`setValues`* method.

Session object

The following commands operate on Session objects. For more information about the Session object, see *Session object*.

Access

```
import visualization
```

`linearizeStress(...)`

This method is used to perform stress linearization along a defined stress line.

Required arguments

name

A String specifying the name of the stress line.

path

A*Path* object specifying the end points of the stress line.



Note:

You must provide either the *path* argument or the *startPoint* and *endPoint* arguments.

startPoint

A String specifying a part instance and a node belonging to that part instance, or a sequence of three Floats specifying the coordinates of a point.

endPoint

A String specifying a part instance and a node belonging to that part instance, or a sequence of three Floats specifying the coordinates of a point.

modelShape

A SymbolicConstant specifying the model shape to be used for obtaining coordinates of the intermediate points along the stress line. Possible values are UNDEFORMED and DEFORMED. When DEFORMED is selected, the deformation scale factor is always set to a uniform value of 1.0.

components

A sequence of Strings specifying the linearized stress components to be displayed in the xyPlot object. Possible values are "S11", "S22", "S33", "S12", "S23", and "S13".



Note:

You must provide at least one of the *components* , *xyMembraneComps* , and *xyBendingComps* arguments.

xyMembraneComps

A sequence of Strings specifying the linearized stress membrane components to be displayed in the xyPlot object. Possible values are "S11", "S22", "S33", "S12", "S23", and "S13".

xyBendingComps

A sequence of Strings specifying the linearized stress bending components to be displayed in the xyPlot object. Possible values are "S11", "S22", "S33", "S12", "S23", and "S13".

Optional arguments

invariantBendingComps

A sequence of Strings specifying the linearized stress bending components to be included in the computation of the linearized stress invariants. Possible values are "S11", "S22", "S33", "S12", "S23", and "S13". The default value of the sequence includes all the possible values.

intervals

An Int specifying the number of equally-spaced intervals into which the stress line is to be divided. The default value is 40.

radiusOfCurvature

A Float specifying the in-plane radius of curvature of the center section along the stress line for axi-symmetric models. The default value is ∞.

oopRadiusOfCurvature

A Float specifying the out-of-plane radius of curvature of the center section along the Stress Line for axi-symmetric models. The default value forces Abaqus to compute the radius of curvature.

curvatureCorrection

A Boolean specifying whether the out-of-plane curvature correction should be applied for nonaxisymmetric models when performing linearized stress computations. Curvature correction is always applied for axisymmetric models. The default value is OFF.

curvatureCsys

A String specifying the name of the user-defined local coordinate system to be used for computing the out-of-plane radius of curvature for nonaxisymmetric models when *curvatureCorrection* =ON . The default value uses the global coordinate system to compute the out-of-plane radius of curvature.

useCurvatureCsysForOrient

A Boolean specifying whether to use the user-defined local coordinate system to compute the local stress line orientation for nonaxisymmetric models when *curvatureCorrection* =ON and *curvatureCsys* is specified. The default value is OFF.

saveXy

A Boolean specifying whether to save the xyData objects created during the stress linearization operation to the session. The default value is OFF .

useMaxStressValue

A Boolean specifying whether to use the maximum stress value from available multiple values at a point. The default value is OFF .

writeReport

A Boolean specifying whether to write the output from the stress linearization operation to a text file. The default value is ON .

reportFile

A String specifying the report file name. The default value is “linearStress.rpt”.

appendToFile

A Boolean specifying whether to append output from the stress linearization to the text file specified by the *reportFile* argument. The default value is ON .

saveToPath

A Boolean specifying whether to create a *Path* object containing all the points used when performing the linearized stress computation. The default value is OFF . When this argument is set to ON , a *Path* object is created with the same name as that of the stress line and placed in the Path repository.

Return value

A list of xyData objects.

Exceptions

Text Error	The stress line passes through incompatible part instances. Results cannot be extracted.
Text Error	Specified variables cannot be used for extracting results along the stress line.
Text Error	Results for all the points along the stress line were not found. Selected end points and all intermediate sample points must lie within the current display group, and within solid continuum elements. Due to tolerance limitations, intermediate points lying on display group boundaries may fail to meet this criterion. Choose new end points, adjust the number of intervals along the line, or modify the current display group to obtain the stress linearization plot.

`setPathTolerance(...)`

This method is used to set the *tolerance* to be used when creating *XYData* objects by extracting results along *Path* objects. This command should be exercised with caution since setting a value too low or too high may result in getting no results or unpredictable results.

Required arguments

None.

Optional arguments

tolerance	A Double specifying the tolerance. The default value is 0.00001.

Return value

None

Exceptions

None.

`getPathTolerance()`

This method is used to get the *tolerance* used when creating *XYData* objects by extracting results along *Path* objects.

Arguments

None.

Return value

A Double specifying the tolerance.

Exceptions

None.

setLimitForXYDataPlots(...)

This method is used to set the *limit* for number of XY data objects while creating *XYData* from field output.

Required arguments

None.

Optional arguments

limit None or an Int specifying the limit for number of XY data objects. The default value is no limit

Return value

None

Exceptions

None.

getLimitForXYDataPlots()

This method is used to get the *limit* for number of XY data objects while creating *XYData* from field output.

Arguments

None.

Return value

Int specifying the limit for number of XY data objects.

Exceptions

None.

writeXYReport

This method writes an XYData object to a user-defined ASCII file.

writeXYReport

This method writes an XYData object to a user-defined ASCII file.

Path

session.writeXYReport

Required arguments

fileName	A String specifying the name of the file to which *X–Y* data will be written.
xyData	A sequence of XYData objects to be written to the output file.

Optional arguments

appendMode	A Boolean specifying whether to append the *X–Y* data to the existing file. The default value is ON.

Return value

`None.`

Exceptions

None.

C++ commands

C++ commands are used to read and write data from an output database.

In this section:

- *Amplitude commands*
- *Beam Section profile commands*
- *Connector commands*
- *Filter commands*
- *Material commands*
- *Odb commands*
- *Property and Section commands*
- *Infrastructure commands*

Amplitude commands

Amplitude commands are used to create arbitrary time or frequency variations of load, displacement, and some interaction attributes throughout a step using step time or throughout an analysis using total time.

In this section:

- *Amplitude object*
- *ActuatorAmplitude object*
- *BaselineCorrection object*
- *DecayAmplitude object*
- *EquallySpacedAmplitude object*
- *ModulatedAmplitude object*
- *PeriodicAmplitude object*
- *PsdDefinition object*
- *SmoothStepAmplitude object*
- *SolutionDependentAmplitude object*
- *SpectrumAmplitude object*
- *TabularAmplitude object*
- *UserAmplitude object*

Amplitude object

The Amplitude object is the abstract base type for other Amplitude objects. The Amplitude object has no explicit constructor. The methods and members of the Amplitude object are common to all objects derived from the Amplitude.

Access

```
amplitudeApi.amplitudes()[name]
```

Members

The Amplitude object has the following members:

Prototype

```
odb_String name() const;
odb_String timeSpan() const;
```

name

An odb_String specifying the repository key.

timeSpan

An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

ActuatorAmplitude object

The ActuatorAmplitude object defines an actuator amplitude curve.

The ActuatorAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

ActuatorAmplitude(...)

This method creates a ActuatorAmplitude object.

Path

```
amplitudeApi.ActuatorAmplitude
```

Prototype

```
odb_ActuatorAmplitude&
ActuatorAmplitude(const odb_String& name,
                  const odb_String& timeSpan);
```

Required arguments

name An odb_String specifying the repository key.

Optional arguments

timeSpan An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

An ActuatorAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the ActuatorAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ActuatorAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ActuatorAmplitude object has members with the same names and descriptions as the arguments to the *ActuatorAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

BaselineCorrection object

The BaselineCorrection object modifies an acceleration history to minimize the overall drift of the displacement obtained from the time integration of the given acceleration.

Access

```
amplitudeApi.amplitudes()[name].baselineCorrection()
```

BaselineCorrection(...)

This method creates a BaselineCorrection object.

Path

```
amplitudeApi.amplitudes()[name].BaselineCorrection
```

Prototype

```
odb_BaselineCorrection&
BaselineCorrection(const odb_SequenceDouble& intervals);
```

Required arguments

None.

Optional arguments

intervals

An odb_SequenceDouble specifying the correction time interval end points. Possible values are positive and monotonically increasing Floats. The default value is an empty sequence.

Return value

A BaselineCorrection object.

Exceptions

RangeError.

setValues(...)

This method modifies the BaselineCorrection object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *BaselineCorrection* method.

Return value

`None.`

Exceptions

RangeError.

Members

The BaselineCorrection object has members with the same names and descriptions as the arguments to the *BaselineCorrection* method.

Corresponding analysis keywords

- **BASELINE CORRECTION*

DecayAmplitude object

The DecayAmplitude object defines an amplitude curve using an exponential decay.

The DecayAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

DecayAmplitude(...)

This method creates a DecayAmplitude object.

Path

```
amplitudeApi.DecayAmplitude
```

Prototype

```
odb_DecayAmplitude&
DecayAmplitude(const odb_String& name,
               double initial,
               double maximum,
               double start,
               double decayTime,
               const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

initial
A Double specifying the constant A_0.

maximum
A Double specifying the coefficient A.

start
A Double specifying the starting time t_0. Possible values are non-negative numbers.

decayTime
A Double specifying the decay time t_d. Possible values are non-negative numbers.

Optional arguments

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A DecayAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the DecayAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DecayAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The DecayAmplitude object has members with the same names and descriptions as the arguments to the *DecayAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

EquallySpacedAmplitude object

The EquallySpacedAmplitude object defines a list of amplitude values at fixed time intervals beginning at a specified value of time.

The EquallySpacedAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

EquallySpacedAmplitude(...)

This method creates an EquallySpacedAmplitude object.

Path

```
amplitudeApi.EquallySpacedAmplitude
```

Prototype

```
odb_EquallySpacedAmplitude&
EquallySpacedAmplitude(const odb_String& name,
                       double fixedInterval,
                       const odb_SequenceDouble& data,
                       double begin,
                       odb_Union smooth,
                       const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

fixedInterval
A Double specifying the fixed time interval at which the amplitude data are given. Possible values are positive numbers.

data
An odb_SequenceDouble specifying the amplitude values.

Optional arguments

begin
A Double specifying the time at which the first amplitude data are given. Possible values are non-negative numbers. The default value is 0.0.

smooth
The string "SOLVER_DEFAULT" or a Double specifying the degree of smoothing. Possible float values are $0 \leq smoothing \leq 0.5$. If *smooth*="SOLVER_DEFAULT", the default degree of smoothing will be determined by the solver. The default value is "SOLVER_DEFAULT".

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

An EquallySpacedAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the EquallySpacedAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *EquallySpacedAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The EquallySpacedAmplitude object has members with the same names and descriptions as the arguments to the *EquallySpacedAmplitude* method.

In addition, the EquallySpacedAmplitude object can have the following member:

Prototype

```
odb_BaselineCorrection baselineCorrection() const;
```

baselineCorrection A *BaselineCorrection* object.

Corresponding analysis keywords

- **AMPLITUDE*

ModulatedAmplitude object

The ModulatedAmplitude object defines a modulated amplitude curve.

The ModulatedAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

ModulatedAmplitude(...)

This method creates a ModulatedAmplitude object.

Path

```
amplitudeApi.ModulatedAmplitude
```

Prototype

```
odb_ModulatedAmplitude&
ModulatedAmplitude(const odb_String& name,
                   double initial,
                   double magnitude,
                   double start,
                   double frequency1,
                   double frequency2,
                   const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

initial
A Double specifying the constant A_0.

magnitude
A Double specifying the coefficient A.

start
A Double specifying the starting time t_0. Possible values are non-negative numbers.

frequency1
A Double specifying the circular frequency 1 (ω_1). Possible values are positive numbers.

frequency2
A Double specifying the circular frequency 2 (ω_2). Possible values are positive numbers.

Optional arguments

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A ModulatedAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the ModulatedAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ModulatedAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ModulatedAmplitude object has members with the same names and descriptions as the arguments to the *ModulatedAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

PeriodicAmplitude object

The PeriodicAmplitude object defines an amplitude curve using a Fourier series.

The PeriodicAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

PeriodicAmplitude(...)

This method creates a PeriodicAmplitude object.

Path

```
amplitudeApi.PeriodicAmplitude
```

Prototype

```
odb_PeriodicAmplitude&
PeriodicAmplitude(const odb_String& name,
                  double frequency,
                  double start,
                  double a_0,
                  const odb_SequenceSequenceDouble& data,
                  const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

frequency
A Double specifying the circular frequency ω. Possible values are positive numbers.

start
A Double specifying the starting time t_0. Possible values are positive numbers.

a_0
A Double specifying the constant A_0.

data
An odb_SequenceSequenceDouble specifying A_i and B_i pairs.

Optional arguments

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A PeriodicAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the PeriodicAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PeriodicAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PeriodicAmplitude object has members with the same names and descriptions as the arguments to the *PeriodicAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

PsdDefinition object

The PsdDefinition object defines the cross-spectral density frequency function for random response loading.

The PsdDefinition object is derived from the *Amplitude* object.

Access

```
Api.amplitudes()[name]
```

PsdDefinition(...)

This method creates a PsdDefinition object.

Path

```
Api.PsdDefinition
```

Prototype

```
odb_PsdDefinition&
PsdDefinition(const odb_String& name,
              const odb_SequenceSequenceDouble& data,
              const odb_String& unitType,
              double referenceGravityAcceleration,
              double referenecePower,
              bool user,
              const odb_String& timeSpan,
              const odb_String& amplitude);
```

Required arguments

name

An odb_String specifying the repository key.

data

An odb_SequenceSequenceDouble specifying the real part of the frequency function, the imaginary part of the frequency function, and the frequency or frequency band number values, depending on the value of *unitType*.

Optional arguments

unitType

An odb_String specifying the type of units for specifying the frequency function. "FORCE" implies power units. "BASE" implies gravity used to define base motion. "DB" implies decibel units. Possible values are "FORCE", "BASE", and "DB". The default value is "FORCE".

referenceGravityAcceleration

A Double specifying the reference gravity acceleration. This argument applies when *unitType* = "BASE". The default value is 1.0.

referenecePower

A Double specifying the reference power value, in load units squared. This argument applies when *unitType* = "DB". The default value is 0.0.

user
A Boolean specifying whether the frequency function is defined in user subroutine UPSD. If specified, then *data* is not applicable, and the *unitType* value must not be "DB". The default value is false.

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

amplitude
An odb_String specifying the name of the amplitude that describes the dynamic event used to define the cross-spectral density frequency function. The default value is an empty string.

Return value

A PsdDefinition object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the PsdDefinition object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *PsdDefinition* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The PsdDefinition object has members with the same names and descriptions as the arguments to the *PsdDefinition* method.

Corresponding analysis keywords

- **PSD-DEFINITION*

SmoothStepAmplitude object

The SmoothStepAmplitude object defines an amplitude that ramps up or down smoothly from one data point to another.

The SmoothStepAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

SmoothStepAmplitude(...)

This method creates a SmoothStepAmplitude object.

Path

```
amplitudeApi.SmoothStepAmplitude
```

Prototype

```
odb_SmoothStepAmplitude&
SmoothStepAmplitude(const odb_String& name,
                    const odb_SequenceSequenceDouble& data,
                    const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

data
An odb_SequenceSequenceDouble specifying time/frequency and amplitude pairs. Possible values for time/frequency are positive numbers.

Optional arguments

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A SmoothStepAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the SmoothStepAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SmoothStepAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SmoothStepAmplitude object has members with the same names and descriptions as the arguments to the *SmoothStepAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

SolutionDependentAmplitude object

The SolutionDependentAmplitude object defines a solution-dependent amplitude for superplastic forming analysis.

The SolutionDependentAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

SolutionDependentAmplitude(...)

This method creates a SolutionDependentAmplitude object.

Path

```
amplitudeApi.SolutionDependentAmplitude
```

Prototype

```
odb_SolutionDependentAmplitude&
SolutionDependentAmplitude(const odb_String& name,
                           double initial,
                           double minimum,
                           double maximum,
                           const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

Optional arguments

initial
A Double specifying the initial amplitude value. Possible values are those between *minimum* and *maximum*. The default value is 1.0.

minimum
A Double specifying the minimum amplitude value. Possible values are those smaller than *maximum* and *initial*. The default value is 0.1.

maximum
A Double specifying the maximum amplitude value. Possible values are those larger than *minimum* and *initial*. The default value is 1000.0.

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A SolutionDependentAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the SolutionDependentAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SolutionDependentAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SolutionDependentAmplitude object has members with the same names and descriptions as the arguments to the *SolutionDependentAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

SpectrumAmplitude object

The SpectrumAmplitude object defines the spectrum of responses for displacement, velocity, or acceleration to be used in a response spectrum analysis.

The SpectrumAmplitude object is derived from the *Amplitude* object.

Access

```
Api.amplitudes()[name]
```

SpectrumAmplitude(...)

This method creates a SpectrumAmplitude object.

Path

```
Api.SpectrumAmplitude
```

Prototype

```
odb_SpectrumAmplitude&
SpectrumAmplitude(const odb_String& name,
                  const odb_String& method,
                  const odb_SequenceSequenceDouble& data,
                  const odb_String& specificationUnits,
                  const odb_String& eventUnits,
                  const odb_String& solution,
                  double timeIncrement,
                  double gravity,
                  bool criticalDamping,
                  const odb_String& timeSpan,
                  const odb_String& amplitude);
```

Required arguments

name
An odb_String specifying the repository key.

method
An odb_String specifying the method for specifying the spectrum. Possible values are "DEFINE" and "CALCULATE".

data
An odb_SequenceSequenceDouble specifying the magnitude, frequency, and damping values.

Optional arguments

specificationUnits
An odb_String specifying the units used for specifying the spectrum. Possible values are "DISPLACEMENT", "VELOCITY", "ACCELERATION", and "GRAVITY". The default value is "ACCELERATION".

eventUnits
An odb_String specifying the units used to describe the dynamic event in the amplitude used for the calculation. Possible values are "EVENT_DISPLACEMENT", "EVENT_VELOCITY", "EVENT_ACCELERATION", and "EVENT_GRAVITY". The default value is "EVENT_ACCELERATION".

solution
An odb_String specifying the solution method for the dynamic equations. Possible values are "ABSOLUTE_VALUE" and "RELATIVE_VALUE". The default value is "ABSOLUTE_VALUE".

timeIncrement
A Double specifying the implicit time increment used to calculate the spectrum. This argument is required when the *method* = "CALCULATE". The default value is 0.0.

gravity
A Double specifying the acceleration due to gravity. This argument applies only when *specificationUnits* = "GRAVITY" or*eventUnits* = "GRAVITY". The default value is 1.0.

criticalDamping
A Boolean specifying whether to calculate the spectrum for only the specified range of critical damping values or a list of values. If *criticalDamping* = true, the spectrum is calculated only for the specified range of critical damping values. If *criticalDamping* = false, the spectrum is calculated for a list of damping values. The default value is false.

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

amplitude
An odb_String specifying the name of the amplitude that describes the dynamic event used to calculate the spectrum. The default value is an empty string.

Return value

A SpectrumAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the SpectrumAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *SpectrumAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The SpectrumAmplitude object has members with the same names and descriptions as the arguments to the *SpectrumAmplitude* method.

Corresponding analysis keywords

- **SPECTRUM*

TabularAmplitude object

The TabularAmplitude object defines an amplitude curve as a table of values at convenient points on the time scale.

The TabularAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

TabularAmplitude(...)

This method creates a TabularAmplitude object.

Path

```
amplitudeApi.TabularAmplitude
```

Prototype

```
odb_TabularAmplitude&
TabularAmplitude(const odb_String& name,
                 const odb_SequenceSequenceDouble& data,
                 odb_Union smooth,
                 const odb_String& timeSpan);
```

Required arguments

name
An odb_String specifying the repository key.

data
An odb_SequenceSequenceDouble specifying time/frequency and amplitude pairs. Possible values for time/frequency are positive numbers.

Optional arguments

smooth
The string "SOLVER_DEFAULT" or a Double specifying the degree of smoothing. Possible float values are between 0 and 0.5. If *smooth*="SOLVER_DEFAULT", the default degree of smoothing will be determined by the solver. The default value is "SOLVER_DEFAULT".

timeSpan
An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A TabularAmplitude object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the TabularAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TabularAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The TabularAmplitude object has members with the same names and descriptions as the arguments to the *TabularAmplitude* method.

In addition, the TabularAmplitude object can have the following member:

Prototype

```
odb_BaselineCorrection baselineCorrection() const;
```

baselineCorrection A *BaselineCorrection* object.

Corresponding analysis keywords

- **AMPLITUDE*

UserAmplitude object

The UserAmplitude object defines an amplitude curve using the *UAMP* or *VUAMP* user subroutine.

The UserAmplitude object is derived from the *Amplitude* object.

Access

```
amplitudeApi.amplitudes()[name]
```

UserAmplitude(...)

This method creates a UserAmplitude object.

Path

```
amplitudeApi.UserAmplitude
```

Prototype

```
odb_UserAmplitude&
UserAmplitude(const odb_String& name,
              int numVariables,
              const odb_String& timeSpan);
```

Required arguments

name An odb_String specifying the repository key.

numVariables An Int specifying the number of variables for the *UAMP* or *VUAMP* user subroutine.

Optional arguments

timeSpan An odb_String specifying the time span of the amplitude. Possible values are "STEP" and "TOTAL". The default value is "STEP".

Return value

A UserAmplitude object.

Exceptions

InvalidNameError and RangeError.

setValues(...)

This method modifies the UserAmplitude object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *UserAmplitude* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The UserAmplitude object has members with the same names and descriptions as the arguments to the *UserAmplitude* method.

Corresponding analysis keywords

- **AMPLITUDE*

Beam Section profile commands

The Beam Section profile commands are used to create profiles. A profile definition specifies the properties of a beam that are related to its cross-section geometry. When you define a beam section, you must include a reference to a profile in the section definition.

In this section:

- *Profile object*
- *ArbitraryProfile object*
- *BoxProfile object*
- *CircularProfile object*
- *GeneralizedProfile object*
- *HexagonalProfile object*
- *IProfile object*
- *LProfile object*
- *PipeProfile object*
- *RectangularProfile object*
- *TProfile object*
- *TrapezoidalProfile object*

Profile object

The Profile object defines the geometrical properties of a beam cross-section. Profile is an abstract base type.

Access

```
sectionApi.profiles()[name]
```

Members

The Profile object has the following member:

Prototype

```
odb_String name() const;
```

name An odb_String specifying the repository key.

ArbitraryProfile object

The ArbitraryProfile object defines the properties of an arbitrary profile.

The ArbitraryProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

ArbitraryProfile(...)

This method creates a ArbitraryProfile object.

Path

```
sectionApi.ArbitraryProfile
```

Prototype

```
odb_ArbitraryProfile&
ArbitraryProfile(const odb_String& name,
                 const odb_SequenceSequenceDouble& table);
```

Required arguments

name An odb_String specifying the repository key.

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

The first sequence in the table specifies the following:

- 1-coordinate of the first point defining the profile.
- 2-coordinate of the first point defining the profile.

All other sequences in the table specify the following:

- 1–coordinate of the next point defining the profile.
- 2–coordinate of the next point defining the profile.
- The thickness of the segment ending at that point.

Return value

An ArbitraryProfile object.

Exceptions

RangeError.

Members

The ArbitraryProfile object has members with the same names and descriptions as the arguments to the *ArbitraryProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=ARBITRARY

BoxProfile object

The BoxProfile object defines the properties of a box profile.

The BoxProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

BoxProfile(...)

This method creates a BoxProfile object.

Path

```
sectionApi.BoxProfile
```

Prototype

```
odb_BoxProfile&
BoxProfile(const odb_String& name,
           double a,
           double b,
           bool uniformThickness,
           double t1,
           double t2,
           double t3,
           double t4);
```

Required arguments

name
An odb_String specifying the repository key.

a
A Double specifying the *a* dimension of the box profile. For more information, see *Beam Cross-Section Library*.

b
A Double specifying the *b* dimension of the box profile.

uniformThickness
A Boolean specifying whether the thickness is uniform.

t1
A Double specifying the uniform wall thickness if *uniformThickness*=true, and the wall thickness of the first segment if *uniformThickness*=false.

Optional arguments

t2
A Double specifying the wall thickness of the second segment. *t2* is required only when *uniformThickness*=false. The default value is 0.0.

t3
A Double specifying the wall thickness of the third segment. *t3* is required only when *uniformThickness*=false. The default value is 0.0.

t4

A Double specifying the wall thickness of the fourth segment. *t4* is required only when *uniformThickness*=false. The default value is 0.0.

Return value

A BoxProfile object.

Exceptions

RangeError.

Members

The BoxProfile object has members with the same names and descriptions as the arguments to the *BoxProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=BOX

CircularProfile object

The CircularProfile object defines the properties of a solid circular profile.

The CircularProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

CircularProfile(...)

This method creates a CircularProfile object.

Path

```
sectionApi.CircularProfile
```

Prototype

```
odb_CircularProfile&
CircularProfile(const odb_String& name,
                double r);
```

Required arguments

name An odb_String specifying the repository key.

r A positive Double specifying the *r* dimension (outer radius) of the circular profile. For more information, see *Beam Cross-Section Library*.

Optional arguments

None.

Return value

A CircularProfile object.

Exceptions

RangeError.

Members

The CircularProfile object has members with the same names and descriptions as the arguments to the *CircularProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=CIRC

GeneralizedProfile object

The GeneralizedProfile object defines the properties of a profile via its area, moment of inertia, etc.

The GeneralizedProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

GeneralizedProfile(...)

This method creates a GeneralizedProfile object.

Path

```
sectionApi.GeneralizedProfile
```

Prototype

```
odb_GeneralizedProfile&
GeneralizedProfile(const odb_String& name,
                   double area,
                   double i11,
                   double i12,
                   double i22,
                   double j,
                   double gammaO,
                   double gammaW);
```

Required arguments

name
An odb_String specifying the repository key.

area
A Double specifying the cross-sectional area for the profile.

i11
A Double specifying the moment of inertia for bending about the 1-axis, I_{11}.

i12
A Double specifying the moment of inertia for cross bending, I_{12}.

i22
A Double specifying the moment of inertia for bending about the 2-axis, I_{22}.

j
A Double specifying the torsional constant, J.

gammaO
A Double specifying the sectorial moment, Γ_0.

gammaW
A Double specifying the warping constant, Γ_W.

Optional arguments

None.

Return value

A GeneralizedProfile object.

Exceptions

RangeError.

Members

The GeneralizedProfile object has members with the same names and descriptions as the arguments to the *GeneralizedProfile* method.

Corresponding analysis keywords

- **BEAM GENERAL SECTION*, SECTION=GENERAL or NONLINEAR GENERAL

HexagonalProfile object

The HexagonalProfile object defines the properties of a hexagonal profile.

The HexagonalProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

HexagonalProfile(...)

This method creates a HexagonalProfile object.

Path

```
sectionApi.HexagonalProfile
```

Prototype

```
odb_HexagonalProfile&
HexagonalProfile(const odb_String& name,
                 double r,
                 double t);
```

Required arguments

name An odb_String specifying the repository key.

r A positive Double specifying the *r* dimension (outer radius) of the hexagonal profile. For more information, see *Beam Cross-Section Library*.

t A positive Double specifying the *t* dimension (wall thickness) of the hexagonal profile, $t < (sqrt(3)/2)r$.

Optional arguments

None.

Return value

A HexagonalProfile object.

Exceptions

RangeError.

Members

The HexagonalProfile object has members with the same names and descriptions as the arguments to the *HexagonalProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=HEX

IProfile object

The IProfile object defines the properties of an I profile.

The IProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

IProfile(...)

This method creates an IProfile object.

Path

```
sectionApi.IProfile
```

Prototype

```
odb_IProfile&
IProfile(const odb_String& name,
         double l,
         double h,
         double b1,
         double b2,
         double t1,
         double t2,
         double t3);
```

Required arguments

name An odb_String specifying the repository key.

l A Double specifying the *l* dimension (offset of 1–axis from the bottom flange surface) of the I profile. For more information, see *Beam Cross-Section Library*.

h A Double specifying the *h* dimension (height) of the I profile.

b1 A Double specifying the *b1* dimension (bottom flange width) of the I profile.

b2 A Double specifying the *b2* dimension (top flange width) of the I profile.

t1 A Double specifying the *t1* dimension (bottom flange thickness) of the I profile.

t2 A Double specifying the *t2* dimension (top flange thickness) of the I profile.

t3 A Double specifying the *t3* dimension (web thickness) of the I profile.

Optional arguments

None.

Return value

An IProfile object.

Exceptions

RangeError.

Members

The IProfile object has members with the same names and descriptions as the arguments to the *IProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=I

LProfile object

The LProfile object defines the properties of a L profile.

The LProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

LProfile(...)

This method creates a LProfile object.

Path

```
sectionApi.LProfile
```

Prototype

```
odb_LProfile&
LProfile(const odb_String& name,
         double a,
         double b,
         double t1,
         double t2);
```

Required arguments

name	An odb_String specifying the repository key.
a	A positive Double specifying the *a* dimension (flange length) of the L profile. For more information, see *Beam Cross-Section Library*.
b	A positive Double specifying the *b* dimension (flange length) of the L profile.
t1	A positive Double specifying the *t1* dimension (flange thickness) of the L profile ($t1 < b$).
t2	A positive Double specifying the *t2* dimension (flange thickness) of the L profile ($t2 < a$).

Optional arguments

None.

Return value

A LProfile object.

Exceptions

RangeError.

Members

The LProfile object has members with the same names and descriptions as the arguments to the *LProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=L

PipeProfile object

The PipeProfile object defines the properties of a circular pipe profile.

The PipeProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

PipeProfile(...)

This method creates a PipeProfile object.

Path

```
sectionApi.PipeProfile
```

Prototype

```
odb_PipeProfile&
PipeProfile(const odb_String& name,
            double r,
            double t,
            const odb_String& formulation);
```

Required arguments

name An odb_String specifying the repository key.

r A Double specifying the outer radius of the pipe. For more information, see *Beam Cross-Section Library*.

t A Double specifying the wall thickness of the pipe.

Optional arguments

None.

Return value

A PipeProfile object.

Exceptions

RangeError.

Members

The PipeProfile object has members with the same names and descriptions as the arguments to the *PipeProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=PIPE
- **BEAM SECTION*, SECTION=THICK PIPE

RectangularProfile object

The RectangularProfile object defines the properties of a solid rectangular profile.

The RectangularProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

RectangularProfile(...)

This method creates a RectangularProfile object.

Path

```
sectionApi.RectangularProfile
```

Prototype

```
odb_RectangularProfile&
RectangularProfile(const odb_String& name,
                   double a,
                   double b);
```

Required arguments

name An odb_String specifying the repository key.

a A positive Double specifying the *a* dimension of the rectangular profile. For more information, see *Beam Cross-Section Library*.

b A positive Double specifying the *b* dimension of the rectangular profile.

Optional arguments

None.

Return value

A RectangularProfile object.

Exceptions

RangeError.

Members

The RectangularProfile object has members with the same names and descriptions as the arguments to the *RectangularProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=RECT

TProfile object

The TProfile object defines the properties of a T profile.

The TProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

TProfile(...)

This method creates a TProfile object.

Path

```
sectionApi.TProfile
```

Prototype

```
odb_TProfile&
TProfile(const odb_String& name,
         double b,
         double h,
         double l,
         double tf,
         double tw);
```

Required arguments

name An odb_String specifying the repository key.

b A positive Double specifying the *b* dimension (flange width) of the T profile. For more information, see *Beam Cross-Section Library*.

h A positive Double specifying the *h* dimension (height) of the T profile.

l A positive Double specifying the *l* dimension (offset of 1–axis from the edge of web) of the T profile.

tf A positive Double specifying the *tf* dimension (flange thickness) of the T profile ($tf < h$).

tw A positive Double specifying the *tw* dimension (web thickness) of the T profile ($tw < b$).

Optional arguments

None.

Return value

A TProfile object.

Exceptions

RangeError.

Members

The TProfile object has members with the same names and descriptions as the arguments to the *TProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=I

TrapezoidalProfile object

The TrapezoidalProfile object defines the properties of a trapezoidal profile.

The TrapezoidalProfile object is derived from the *Profile* object.

Access

```
sectionApi.profiles()[name]
```

TrapezoidalProfile(...)

This method creates a TrapezoidalProfile object.

Path

```
sectionApi.TrapezoidalProfile
```

Prototype

```
odb_TrapezoidalProfile&
TrapezoidalProfile(const odb_String& name,
                   double a,
                   double b,
                   double c,
                   double d);
```

Required arguments

name An odb_String specifying the repository key.

a A positive Double specifying the *a* dimension of the Trapezoidal profile. For more information, see *Beam Cross-Section Library*.

b A positive Double specifying the *b* dimension of the Trapezoidal profile.

c A positive Double specifying the *c* dimension of the Trapezoidal profile.

d A Double specifying the *d* dimension of the Trapezoidal profile.

Optional arguments

None.

Return value

A TrapezoidalProfile object.

Exceptions

RangeError.

Members

The TrapezoidalProfile object has members with the same names and descriptions as the arguments to the *TrapezoidalProfile* method.

Corresponding analysis keywords

- **BEAM SECTION*, SECTION=TRAPEZOID

Connector commands

A connector describes the relative motions between two points. A connector also describes the behavior associated with the relative motion.

In this section:

- *ConnectorBehaviorOption object*
- *CDCTerm object*
- *ConnectorDamage object*
- *ConnectorDamping object*
- *ConnectorElasticity object*
- *ConnectorFailure object*
- *ConnectorFriction object*
- *ConnectorLock object*
- *ConnectorOptions object*
- *ConnectorPlasticity object*
- *ConnectorPotential object*
- *ConnectorStop object*
- *DerivedComponent object*
- *TangentialBehavior object*

ConnectorBehaviorOption object

The ConnectorBehaviorOption object is the abstract base type for other ConnectorBehaviorOption objects. The ConnectorBehaviorOption object has no explicit constructor. The members of the ConnectorBehaviorOption object are common to all objects derived from the ConnectorBehaviorOption.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

Members

The ConnectorBehaviorOption object has no members.

CDCTerm object

The CDCTerm object is used to create contributing terms for a *DerivedComponent* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.derivedComponent().cdcTerms(i)
sectionApi.sections()[name].behaviorOptions(i).derivedComponent()\
.cdcTerms(i)
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.derivedComponent().cdcTerms(i)
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.derivedComponent().cdcTerms(i)
```

CDCTerm(...)

This method creates a CDCTerm object.

Path

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.derivedComponent().CDCTerm
sectionApi.sections()[name].behaviorOptions(i).derivedComponent()\
.CDCTerm
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.derivedComponent().CDCTerm
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.derivedComponent().CDCTerm
```

Prototype

```
odb_CDCTerm&
CDCTerm(const odb_SequenceInt& intrinsicComponents,
        const odb_SequenceSequenceDouble& table,
        const odb_String& termOperator,
        const odb_String& termSign,
        bool localDependency,
        const odb_String& indepCompType,
        const odb_SequenceInt& indepComponents,
        bool tempDependency,
        int fieldDependencies);
```

Required arguments

intrinsicComponents An odb_SequenceInt specifying the components of relative motion for which the contributing term is defined. Possible values are $1 \leq intrinsicComponents \leq 6$. Only available components can be specified if the DerivedComponent object is being referenced by a Potential object. This is not the case if the DerivedComponent object is referenced by a ConnectorFriction object directly. The default value is an empty sequence.

table An odb_SequenceSequenceDouble specifying components numbers and temperature and field values. Each sequence of the table data specifies:

- The first intrinsic component number.
- If applicable, the second intrinsic component number.
- Etc.
- If applicable, the first independent component number.
- If applicable, the second independent component number.
- Etc.
- If applicable, the temperature value.
- If applicable, the value of the first field variable.
- If applicable, the value of the second field variable.
- Etc.

The default value is an empty sequence.

Optional arguments

termOperator An odb_String specifying the method for combining contributing terms: square root of a sum of the squares, direct sum, or Macauley sum. Possible values are "RSS", "SUM", and "MACAULEY". The default value is "RSS".

termSign An odb_String specifying the overall sign for the contributing term. Possible values are "POSITIVE" and "NEGATIVE". The default value is "POSITIVE".

localDependency A Boolean specifying whether the table data depend on either components of relative position or components of constitutive relative motion. The default value is false.

indepCompType An odb_String specifying whether localDependency refers to components of relative position or components of constitutive relative motion. Possible values are "POSITION" and "MOTION". The default value is "POSITION".

The *indepCompType* argument applies only if *localDependency*=true.

indepComponents An odb_SequenceInt specifying the independent components included in the derived component definition. Possible values are $1 \leq$ *indepComponents* ≤ 6. Only available components can be specified. The *indepComponents* argument applies only if *localDependency*=true. The default value is an empty sequence.

tempDependency A Boolean specifying whether the table data depend on temperature. The default value is false.

fieldDependencies An Int specifying the number of field variable dependencies. The default value is 0.

Return value

A CDCTerm object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the CDCTerm object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *CDCTerm* method.

Return value

`None.`

Exceptions

ValueError.

Members

The CDCTerm object has members with the same names and descriptions as the arguments to the *CDCTerm* method.

In addition, the CDCTerm object can have the following member:

Prototype

```
odb_ConnectorOptions options() const;
```

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

ConnectorDamage object

The ConnectorDamage object defines damage behavior for one or more components of a connector's relative motion.

The ConnectorDamage object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorDamage(...)

This method creates a connector damage behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorDamage
```

Prototype

```
odb_ConnectorDamage&
ConnectorDamage(const odb_String& coupling,
  const odb_String& criterion,
  bool initiationTemperature,
  const odb_String& initiationPotentialOperator,
  double initiationPotentialExponent,
  int initiationDependencies,
  bool evolution,
  const odb_String& evolutionType,
  const odb_String& softening,
  bool useAffected,
  const odb_String& degradation,
  bool evolutionTemperature,
  int evolutionDependencies,
  const odb_String& evolutionPotentialOperator,
  double evolutionPotentialExponent,
  const odb_SequenceConnectorPotential& initiationPotentials,
  const odb_SequenceConnectorPotential& evolutionPotentials,
  const odb_SequenceSequenceDouble& initiationTable,
  const odb_SequenceSequenceDouble& evolutionTable,
  const odb_SequenceInt& affectedComponents,
  const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

coupling

An odb_String specifying whether or not the behavior is coupled. Possible values are "UNCOUPLED" and "COUPLED". The default value is "UNCOUPLED".

criterion

An odb_String specifying the damage initiation criterion to be used. Possible values are "FORCE", "MOTION", and "PLASTIC_MOTION". The default value is "FORCE".

initiationTemperature
A Boolean specifying whether the initiation data depend on temperature. The default value is false.

initiationPotentialOperator
An odb_String specifying the contribution operator for the initiation potential contributions. Possible values are "SUM" and "MAXIMUM". The default value is "SUM".

This argument is only if *coupling*="COUPLED" and if *criterion*="FORCE" or "MOTION".

initiationPotentialExponent
A Double specifying the number equal to the inverse of the overall exponent in the initiation potential definition. The default value is 2.0.

This argument is applicable only if *coupling*="COUPLED", when *initiationPotentialOperator*="SUM", and when *criterion*="FORCE" or "MOTION".

initiationDependencies
An Int specifying the number of field variable dependencies for the initiation data. The default value is 0.

evolution
A Boolean specifying whether damage evolution data will be used. The default value is true.

evolutionType
An odb_String specifying the type of damage evolution to be specified. Possible values are "MOTION_TYPE" and "ENERGY_TYPE". The default value is "MOTION_TYPE".

This argument is applicable only if *evolution*=true.

softening
An odb_String specifying the damage evolution law to be specified. Possible values are "LINEAR", "EXPONENTIAL", and "TABULAR". The default value is "LINEAR".

This argument is applicable only if *evolution*=true and when *evolutionType*="MOTION_TYPE".

useAffected
A Boolean specifying whether or not *affectedComponents* will be specified. If *useAffected*=false, then only the components of relative motion specified by *components* will undergo damage. The default value is false.

This argument is applicable only if *evolution*=true.

degradation
An odb_String specifying the contribution of each damage mechanism when more than one damage mechanism is defined. Possible values are "MAXIMUM" and "MULTIPLICATIVE". The default value is "MAXIMUM".

This argument is applicable if *evolution*=true.

evolutionTemperature
A Boolean specifying whether the evolution data depend on temperature. The default value is false.

This argument is applicable only if *evolution*=true.

evolutionDependencies
An Int specifying the number of field variable dependencies for the evolution data. The default value is 0.

This argument is applicable only if *evolution*=true.

evolutionPotentialOperator An odb_String specifying the contribution operator for the evolution potential contributions. Possible values are "SUM" and "MAXIMUM". The default value is "SUM".

This argument is applicable only if *coupling*="COUPLED", when *evolution*=true, when *evolutionType*="MOTION_TYPE", and when *criterion*="FORCE" or "MOTION".

evolutionPotentialExponent A Double specifying the number equal to the inverse of the overall exponent in the evolution potential definition. The default value is 2.0.

This argument is applicable only if *coupling*="COUPLED", when *evolution*=true, when *evolutionPotentialOperator*="SUM", when *evolutionType*="MOTION", and when *criterion*="FORCE" or "MOTION".

initiationPotentials A sequence of *ConnectorPotential* objects specifying one *ConnectorPotential* object for each initiation potential contribution. This member can be specified only if *coupling*="COUPLED" and if *criterion*="FORCE" or "MOTION".

evolutionPotentials A sequence of *ConnectorPotential* objects specifying one *ConnectorPotential* object for each evolution potential contribution). This member can be specified only if *coupling*="COUPLED", if *evolution*=true, if *evolutionType*="MOTION", and if *criterion*="FORCE" or "MOTION".

initiationTable An odb_SequenceSequenceDouble specifying the initiation properties. The default value is an empty sequence.

Items in the *initiationTable* data are described below.

evolutionTable An odb_SequenceSequenceDouble specifying the evolution properties. The default value is an empty sequence.

Items in the *evolutionTable* data are described below. This argument is only applicable if *evolution*=true.

affectedComponents An odb_SequenceInt specifying the components of relative motion that will be damaged. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument is applicable only if *evolution*=true and *useAffected*=true. The default value is an empty sequence.

components An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument can be specified only if *coupling*="UNCOUPLED". The default value is an empty sequence.

Table data

Table data for *initiationTable*:

If *criterion*=FORCE, then each sequence of the table data specifies the following:

- Lower (compression) limiting force or moment. Use -1.0E+36 to indicate an unspecified lower limit.
- Upper (tension) limiting force or moment. Use 1.0E+36 to indicate an unspecified upper limit. At least one limit, lower or upper, must be specified.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *criterion*=MOTION, then each sequence of the table data specifies the following:

- Lower (compression) limiting connector constitutive relative displacement or rotation. Use -1.0E+36 to indicate an unspecified lower limit.
- Upper (tension) limiting connector constitutive relative displacement or rotation. Use 1.0E+36 to indicate an unspecified upper limit. At least one limit, lower or upper, must be specified.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *criterion*=PLASTIC_MOTION, then each sequence of the table data specifies the following:

- Relative equivalent plastic displacement/rotation at which damage will be initiated.
- Mode mix ratio (only if *coupling*=COUPLED).
- Relative equivalent plastic displacement/rotation rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Table data for *evolutionTable*:

If *evolutionType*=MOTION and *softening*=LINEAR, then each sequence of the table data specifies the following:

- Post-initiation equivalent relative plastic motion at ultimate failure if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation) at ultimate failure.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=MOTION and *softening*=EXPONENTIAL, then each sequence of the table data specifies the following:

- Post-initiation equivalent relative plastic motion at ultimate failure if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation) at ultimate failure.
- Exponential law parameter.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *evolutionType*=MOTION and *softening*=TABULAR, then each sequence of the table data specifies the following:

- Damage variable (cannot be less than 0 or greater than 1).
- Post-initiation equivalent relative plastic motion if *criterion*=PLASTIC_MOTION. Otherwise, post-initiation constitutive relative motion (displacement/rotation).
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *evolutionType*=ENERGY, then each sequence of the table data specifies the following:

- Total energy dissipated by damage at ultimate failure.
- Mode mix ratio (only if *coupling*=COUPLED and *criterion*=PLASTIC_MOTION).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorDamage object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorDamage object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorDamage* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorDamage object has members with the same names and descriptions as the arguments to the *ConnectorDamage* method.

In addition, the ConnectorDamage object can have the following members:

Prototype

```
odb_ConnectorOptions initiationOptions() const;
odb_ConnectorOptions evolutionOptions() const;
```

initiationOptions

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the damage initiation table.

evolutionOptions

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the damage evolution table.

Corresponding analysis keywords

- **CONNECTOR DAMAGE INITIATION*, **CONNECTOR DAMAGE EVOLUTION*, **CONNECTOR POTENTIAL*

ConnectorDamping object

The ConnectorDamping object defines damping behavior for one or more components of a connector's relative motion.

The ConnectorDamping object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorDamping(...)

This method creates a connector damping behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorDamping
```

Prototype

```
odb_ConnectorDamping&
ConnectorDamping(const odb_String& behavior,
                 const odb_String& coupling,
                 int dependencies,
                 bool temperatureDependency,
                 bool frequencyDependency,
                 const odb_SequenceSequenceDouble& table,
                 const odb_SequenceInt& independentComponents,
                 const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

behavior

An odb_String specifying if the damping behavior is linear or nonlinear. Possible values are "LINEAR" and "NONLINEAR". The default value is "LINEAR".

coupling

An odb_String specifying whether the damping behavior is coupled between the connector's components of relative motion. If *behavior*="LINEAR", then possible values are "UNCOUPLED" and "COUPLED". If *behavior*="NONLINEAR", then possible values are "UNCOUPLED", "COUPLED_POSITION", and "COUPLED_MOTION". Possible values are "UNCOUPLED", "COUPLED", "COUPLED_POSITION", and "COUPLED_MOTION". The default value is "UNCOUPLED".

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

temperatureDependency

A Boolean specifying whether the behavior data depend on temperature. The default value is false.

frequencyDependency A Boolean specifying whether the behavior data depend on frequency. This value is applicable only if *behavior*= "LINEAR" and *coupling*="UNCOUPLED". The default value is false.

table An odb_SequenceSequenceDouble specifying damping properties. Items in the table data are described below. The default value is an empty sequence.

independentComponents An odb_SequenceInt specifying the list of independent components that are included in the definition of the connector damping data. This argument is applicable only if *behavior*="NONLINEAR" and *coupling*="COUPLED_POSITION" or "COUPLED_MOTION". When this argument is applicable, at least one value must be specified. Only available components can be specified. The default value is an empty sequence.

components An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Table data

If *behavior*=LINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Damping coefficient (force or moment per relative velocity).
- Frequency (cycles/time), if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=NONLINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Force or moment.
- Relative displacement or rotation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=LINEAR and *coupling*=COUPLED, the table data specify the symmetric portion of the damping matrix for the specified components followed by any temperature data and then any field data. For example, if components 2, 3, and 5 are specified, the table portion of the command is as follows:

```
table=( (C22, C23, C25, C33, C35, C55,), )
```

The following items should then be specified as comma-separated data:

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *behavior*=NONLINEAR and *coupling*=COUPLED_POSITION or COUPLED_MOTION, each sequence of the table data specifies the following:

- Force or moment for the directions in the *components* list.
- Relative velocity for the directions in the *components* list.
- Relative position or angle for the first *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the first *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Relative position or angle for the second *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the second *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Etc. up to the Nth *independentComponents* direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorDamping object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorDamping object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorDamping* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorDamping object has members with the same names and descriptions as the arguments to the *ConnectorDamping* method.

In addition, the ConnectorDamping object can have the following member:

Prototype

```
odb_ConnectorOptions options() const;
```

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords

- **CONNECTOR DAMPING*

ConnectorElasticity object

The ConnectorElasticity object defines elastic behavior for one or more components of a connector's relative motion.

The ConnectorElasticity object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorElasticity(...)

This method creates a connector elasticity behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorElasticity
```

Prototype

```
odb_ConnectorElasticity&
ConnectorElasticity(const odb_String& behavior,
                const odb_String& coupling,
                int dependencies,
                bool temperatureDependency,
                bool frequencyDependency,
                const odb_SequenceSequenceDouble& table,
                const odb_SequenceInt& independentComponents,
                const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

behavior

An odb_String specifying whether the elastic behavior is linear, nonlinear, or rigid. Possible values are "LINEAR", "NONLINEAR", and "RIGID". The default value is "LINEAR".

coupling

An odb_String specifying whether the elastic behavior is coupled between the connector's components of relative motion. If *behavior*="LINEAR", then possible values are "UNCOUPLED" and "COUPLED". If *behavior*="NONLINEAR", then possible values are "UNCOUPLED", "COUPLED_POSITION", and "COUPLED_MOTION". Possible values are "UNCOUPLED", "COUPLED", "COUPLED_POSITION", and "COUPLED_MOTION". The default value is "UNCOUPLED".

This argument is not applicable if *behavior*="RIGID".

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

This argument is not applicable if *behavior*="RIGID".

temperatureDependency

A Boolean specifying whether the behavior data depend on temperature. The default value is false.

This argument is not applicable if *behavior*="RIGID".

frequencyDependency

A Boolean specifying whether the behavior data depend on frequency. This value is applicable only if *behavior*="LINEAR" and *coupling*="UNCOUPLED". The default value is false.

This argument is not applicable if *behavior*="RIGID".

table

An odb_SequenceSequenceDouble specifying elasticity properties. Items in the table data are described below. This argument is not applicable if *behavior*="RIGID". The default value is an empty sequence.

independentComponents

An odb_SequenceInt specifying the list of independent components that are included in the definition of the connector elasticity data. This argument is applicable only if *behavior*="NONLINEAR" and *coupling*="COUPLED_POSITION" or "COUPLED_MOTION". If this argument is applicable, at least one value must be specified. Only available components can be specified. The default value is an empty sequence.

components

An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Table data

If *behavior*=LINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Elastic stiffness (per each specified force/moment component).
- Frequency (cycles/time), if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=NONLINEAR and *coupling*=UNCOUPLED, then each sequence of the table data specifies the following:

- Force or moment.
- Relative displacement or rotation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=LINEAR and *coupling*=COUPLED, the table data specify the symmetric portion of the elastic stiffness matrix for the specified components followed by any temperature data and then any field data. For example, if components 2, 3, and 5 are specified, the table portion of the command is as follows:

```
table=( (D22, D23, D25, D33, D35, D55,),
)
```

The following items should then be specified as comma-separated data:

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *behavior*=NONLINEAR and *coupling*=COUPLED_POSITION or COUPLED_MOTION, each sequence of the table data specifies the following:

- Force or moment for the directions in the *components* list.
- Relative position or angle for the first *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the first *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Relative position or angle for the second *independentComponents* direction if *coupling*=COUPLED_POSITION. Relative displacement or rotation for the second *independentComponents* direction if *coupling*=COUPLED_MOTION.
- Etc. up to the Nth *independentComponents* direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorElasticity object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorElasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorElasticity* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorElasticity object has members with the same names and descriptions as the arguments to the *ConnectorElasticity* method.

In addition, the ConnectorElasticity object can have the following member:

Prototype

```
odb_ConnectorOptions options() const;
```

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords

- **CONNECTOR ELASTICITY*

ConnectorFailure object

The ConnectorFailure object defines failure criteria for one or more components of a connector's relative motion.

The ConnectorFailure object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorFailure(...)

This method creates a connector failure behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorFailure
```

Prototype

```
odb_ConnectorFailure&
ConnectorFailure(odb_Union releaseComponent,
                 odb_Union minMotion,
                 odb_Union maxMotion,
                 odb_Union minForce,
                 odb_Union maxForce,
                 const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

releaseComponent The string "ALL" or an Int specifying the motion components that fail. If an Int is specified, only that motion component fails when the failure criteria are satisfied. If *releaseComponent*="ALL", all motion components fail. The default value is "ALL".

minMotion The string "NONE" or a Double specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is "NONE".

maxMotion The string "NONE" or a Double specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is "NONE".

minForce The string "NONE" or a Double specifying the lower bound of the force or moment in the directions of the specified components at which locking occurs, or no lower bound. The default value is "NONE".

maxForce The string "NONE" or a Double specifying the upper bound of the force or moment in the directions of the specified components at which locking occurs, or no upper bound. The default value is "NONE".

components An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorFailure object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorFailure object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorFailure* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorFailure object has members with the same names and descriptions as the arguments to the *ConnectorFailure* method.

Corresponding analysis keywords

- **CONNECTOR FAILURE*

ConnectorFriction object

The ConnectorFriction object defines Coulomb-like or hysteretic friction behavior for one or more components of a connector's relative motion.

The ConnectorFriction object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorFriction(...)

This method creates a connector friction behavior option for a *ConnectorSection* object. Depending upon the arguments provided, the friction behavior can be Coulomb-like or hysteretic in nature.

Path

```
sectionApi.sections()[name].ConnectorFriction
```

Prototype

```
odb_ConnectorFriction&
ConnectorFriction(const odb_String& frictionModel,
   const odb_String& slipStyle,
   odb_Union tangentDirection,
   odb_Union stickStiffness,
   const odb_String& componentType,
   bool slipDependency,
   bool temperatureDependency,
   int dependencies,
   bool useContactForceComponent,
   const odb_String& contactForceStyle,
   int contactForceComponent,
   const odb_String& forcePotentialOperator,
   double forcePotentialExponent,
   const odb_SequenceConnectorPotential& connectorPotentials,
   const odb_SequenceSequenceDouble& table,
   const odb_SequenceInt& independentComponents);
```

Required arguments

None.

Optional arguments

frictionModel

An odb_String specifying the desired frictional response model. Possible values are "PREDEFINED" and "USER_CUSTOMIZED". The default value is "PREDEFINED".

slipStyle

An odb_String specifying the method of indicating the slip direction: either specified or computed based upon the force potential data. Possible values are "SPECIFY" and "COMPUTE". The default value is "SPECIFY".

This argument is applicable only if *frictionModel*="USER_CUSTOMIZED".

tangentDirection

The string "NONE" or an Int specifying the direction for which the frictional behavior is specified. Possible values are $1 \leq$ *tangentDirection* ≤ 6, indicating an available component of relative motion. This argument applies only if *frictionModel*="USER_CUSTOMIZED" and if *slipStyle*="SPECIFY". The default value is "NONE".

stickStiffness

The string "NONE" or a Double specifying the stick stiffness associated with the frictional behavior in the direction specified by *tangentDirection*. If this argument is omitted, Abaqus computes an appropriate number for the stick stiffness. The default value is "NONE".

componentType

An odb_String specifying the type of the *independentComponents*. Possible values are "POSITION", "MOTION", and "NO_INDEPENDENT_COMPONENTS". The default value is "NO_INDEPENDENT_COMPONENTS".

slipDependency

A Boolean specifying whether the table data depend on accumulated slip. The default value is false.

This argument applies only if *frictionModel*="USER_CUSTOMIZED".

temperatureDependency

A Boolean specifying whether the table data depend on temperature. The default value is false.

This argument applies only if *frictionModel*="USER_CUSTOMIZED".

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

This argument applies only if *frictionModel*="USER_CUSTOMIZED".

useContactForceComponent

A Boolean specifying whether the contact force component will be defined. The default value is false.

This argument applies only if *frictionModel*="USER_CUSTOMIZED".

contactForceStyle

An odb_String specifying the method of indicating the contact force component direction: either specified or computed based on upon a DerivedComponent. Possible values are "COMPONENT_NUMBER" and "DERIVED_COMPONENT". The default value is "COMPONENT_NUMBER".

This argument is applicable only if *frictionModel*="USER_CUSTOMIZED" and if *useContactForceComponent*=true.

contactForceComponent

An Int specifying the contact force component direction. This argument applies only if *frictionModel*="USER_CUSTOMIZED", if *useContactForceComponent*=true, and if *contactForceStyle*="COMPONENT_NUMBER". The default value is 0.

forcePotentialOperator

An odb_String specifying the contribution operator for the force potential contributions. Possible values are "SUM" and "MAXIMUM". The default value is "SUM".

This argument is applicable only if *frictionModel*="USER_CUSTOMIZED" and if *slipStyle*="COMPUTE".

forcePotentialExponent

A Double specifying the number equal to the inverse of the overall exponent in the force potential definition. The default value is 2.0.

This argument is applicable only if *frictionModel*="USER_CUSTOMIZED", if *slipStyle*="COMPUTE", and if *forcePotentialOperator*="SUM".

connectorPotentials

A sequence of *ConnectorPotential* objects specifying one *ConnectorPotential* object for each force potential contribution. This member can be specified only if *frictionModel*="USER_CUSTOMIZED", and if *slipStyle*="COMPUTE".

table

An odb_SequenceSequenceDouble specifying friction properties. The default value is an empty sequence.

If *frictionModel*="PREDEFINED", each sequence of the table data specifies:

- If applicable, the first geometric scaling constant relevant to frictional interactions.
- Etc., up to as many geometric scaling constants as are associated with this connection type.
- Internal contact force/moment generating friction in the first predefined slip direction.
- If applicable, internal contact force/moment generating friction in the second predefined slip direction.
- Connector constitutive relative motion in the direction specified by *independentComponent*.
- Accumulated slip in the first predefined slip direction, if the data depend on accumulated slip.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *frictionModel*="USER_CUSTOMIZED", each sequence of the table data specifies:

- Effective radius of the cylindrical or spherical surface over which frictional slip occurs in the connector associated with frictional effects in the direction specified by *tangentDirection*. This radius is relevant only if the connection type includes an available rotational component of relative motion and *tangentDirection*="SLIP_DIRECTION".
- Internal contact force/moment generating friction in the direction specified by *tangentDirection*.
- Connector constitutive relative motion in the direction specified by *independentComponent*.
- Accumulated slip in the direction specified by *tangentDirection*, if the data depend on accumulated slip.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

independentComponents An odb_SequenceInt specifying the independent components. Possible values are 1 ≤*independentComponents*≤ 6. In addition, each independent component value must be unique. The *independentComponents* argument applies only if *frictionModel*="USER_CUSTOMIZED". Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorFriction object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorFriction object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorFriction* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorFriction object has members with the same names and descriptions as the arguments to the *ConnectorFriction* method.

In addition, the ConnectorFriction object can have the following members:

Prototype

```
odb_TangentialBehavior tangentialBehavior() const;
odb_DerivedComponent derivedComponent() const;
odb_ConnectorOptions options() const;
```

tangentialBehavior A *TangentialBehavior* object.

derivedComponent A *DerivedComponent* object specifying the *DerivedComponent* used to compute the contact force component direction. This argument applies only if *frictionModel*="USER_CUSTOMIZED", if *useContactForceComponent*=true, and if *contactForceStyle*="DERIVED_COMPONENT".

options

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for this *ConnectorBehaviorOption*.

Corresponding analysis keywords

- **CONNECTOR FRICTION*, **FRICTION*, **CONNECTOR POTENTIAL*, **CONNECTOR DERIVED COMPONENT*

ConnectorLock object

The ConnectorLock object defines locking criteria for one or more available components of a connector's relative motion.

The ConnectorLock object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorLock(...)

This method creates a connector lock behavior option for a *ConnectorSection*.

Path

```
sectionApi.sections()[name].ConnectorLock
```

Prototype

```
odb_ConnectorLock&
ConnectorLock(odb_Union lockingComponent,
              odb_Union minMotion,
              odb_Union maxMotion,
              odb_Union minForce,
              odb_Union maxForce,
              const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

lockingComponent

The string "ALL" or an Int specifying the motion components that are locked. If an Int is specified, only that motion component is locked when the locking criteria are satisfied. If *lockingComponent*=ALL, all motion components are locked. The default value is "ALL".

minMotion

The string "NONE" or a Double specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is "NONE".

maxMotion

The string "NONE" or a Double specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is "NONE".

minForce

The string "NONE" or a Double specifying the lower bound of the force or moment in the directions of the specified components at which locking occurs, or no lower bound. The default value is "NONE".

maxForce

The string "NONE" or a Double specifying the upper bound of the force or moment in the directions of the specified components at which locking occurs, or no upper bound. The default value is "NONE".

components An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorLock object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorLock object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorLock* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorLock object has members with the same names and descriptions as the arguments to the *ConnectorLock* method.

Corresponding analysis keywords

**CONNECTOR LOCK*

ConnectorOptions object

The ConnectorOptions object is used to define various options for connector behaviors. It can be used only in conjunction with *CDCTerm*, *ConnectorDamage*, *ConnectorDamping*, *ConnectorElasticity*, *ConnectorFriction*, and *ConnectorPlasticity* objects. Because the *ConnectorDamage* object contains two separate ConnectorOptions repositories (one for damage initiation and one for damage evolution), there are two ConnectorOptions constructors associated with that behavior—initiationOptions and evolutionOptions. The *ConnectorPlasticity* object also contains two separate ConnectorOptions repositories (one for isotropic hardening and one for kinematic hardening), so there are two ConnectorOptions constructors associated with that behavior—isotropicOptions and kinematicOptions.

Access

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.derivedComponent().cdcTerms(i).options()
sectionApi.sections()[name].behaviorOptions(i).derivedComponent()\
.cdcTerms(i).options()
sectionApi.sections()[name].behaviorOptions(i).evolutionOptions()
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.derivedComponent().cdcTerms(i).options()
sectionApi.sections()[name].behaviorOptions(i).initiationOptions()
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.derivedComponent().cdcTerms(i).options()
sectionApi.sections()[name].behaviorOptions(i).isotropicOptions()
sectionApi.sections()[name].behaviorOptions(i).kinematicOptions()
sectionApi.sections()[name].behaviorOptions(i).options()
```

ConnectorOptions(...)

This method creates a connector options object to be used in conjunction with an allowable connector behavior option, derived component term, or connector section.

Path

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.derivedComponent().cdcTerms(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).derivedComponent()\
.cdcTerms(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.derivedComponent().cdcTerms(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.derivedComponent().cdcTerms(i).ConnectorOptions
sectionApi.sections()[name].behaviorOptions(i).ConnectorOptions
```

Prototype

```
odb_ConnectorOptions&
ConnectorOptions(bool useBehRegSettings,
                 bool regularize,
                 bool defaultTolerance,
                 double regularization,
                 bool defaultRateFactor,
                 double rateFactor,
                 const odb_String& interpolation,
```

```
                    bool useBehExtSettings,
                    const odb_String& extrapolation);
```

Required arguments

None.

Optional arguments

useBehRegSettings
A Boolean specifying whether or not to use the behavior-level settings for regularization options. This argument is applicable only for an Abaqus/Explicit analysis. The default value is true.

regularize
A Boolean specifying whether or not the tabular data will be regularized. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=false. The default value is true.

defaultTolerance
A Boolean specifying whether or not the analysis default regularization tolerance will be used. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=false and *regularize*=true. The default value is true.

regularization
A Double specifying the regularization increment to be used. This argument is applicable only for an Abaqus/Explicit analysis and only if *useBehRegSettings*=false, *regularize*=true, and *defaultTolerance*=false. The default value is 0.03.

defaultRateFactor
A Boolean specifying whether or not the analysis default rate filter factor will be used. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic motion criteria. The default value is true.

rateFactor
A Double specifying the rate filter factor to be used. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic motion criteria. This argument is also applicable only if *defaultRateFactor*=false. The default value is 0.9.

interpolation
An odb_String specifying the type of interpolation increment to be used on rate-dependent tabular data. This argument is applicable only for an Abaqus/Explicit analysis that includes isotropic hardening with tabular definition or damage initiation with plastic motion criteria. Possible values are "LINEAR" and "LOGARITHMIC". The default value is "LINEAR".

useBehExtSettings
A Boolean specifying whether or not to use the behavior-level settings for extrapolation options. The default value is true.

extrapolation
An odb_String specifying the extrapolation technique to be used. This argument is applicable only if *useBehExtSettings*=false. Possible values are "CONSTANT" and "LINEAR". The default value is "CONSTANT".

Return value

A ConnectorOptions object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorOptions object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorOptions* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorOptions object has members with the same names and descriptions as the arguments to the *ConnectorOptions* method.

Corresponding analysis keywords

- **CONNECTOR BEHAVIOR*, **CONNECTOR DAMAGE INITIATION*, **CONNECTOR DAMAGE EVOLUTION*, **CONNECTOR DAMPING*, **CONNECTOR DERIVED COMPONENT*, **CONNECTOR ELASTICITY*, **CONNECTOR FRICTION*, and **CONNECTOR PLASTICITY*

ConnectorPlasticity object

The ConnectorPlasticity object defines plastic behavior for one or more components of a connector's relative motion.

The ConnectorPlasticity object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorPlasticity(...)

This method creates a connector plasticity behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorPlasticity
```

Prototype

```
odb_ConnectorPlasticity&
ConnectorPlasticity(const odb_String& coupling,
   bool isotropic,
   const odb_String& isotropicType,
   bool isotropicTemperature,
   int isotropicDependencies,
   bool kinematic,
   const odb_String& kinematicType,
   bool kinematicTemperature,
   int kinematicDependencies,
   const odb_String& forcePotentialOperator,
   double forcePotentialExponent,
   const odb_SequenceConnectorPotential& connectorPotentials,
   const odb_SequenceSequenceDouble& isotropicTable,
   const odb_SequenceSequenceDouble& kinematicTable,
   const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

coupling

An odb_String specifying whether or not the behavior is coupled. Possible values are "UNCOUPLED" and "COUPLED". The default value is "UNCOUPLED".

isotropic

A Boolean specifying whether isotropic hardening data will be used. The default value is true.

If *isotropic*=false, then *kinematic* must be specified as true.

isotropicType

An odb_String specifying the type of isotropic hardening to be specified. Possible values are "TABULAR" and "EXPONENTIAL_LAW". The default value is "TABULAR".

This argument is applicable only if *isotropic*=true.

isotropicTemperature

A Boolean specifying whether the isotropic data depend on temperature. The default value is false.

This argument is applicable only if *isotropic*=true.

isotropicDependencies

An Int specifying the number of field variable dependencies for the isotropic data. The default value is 0.

This argument is applicable only if *isotropic*=true.

kinematic

A Boolean specifying whether kinematic hardening data will be used. The default value is false.

If *kinematic*=false, then *isotropic* must be specified as true.

kinematicType

An odb_String specifying the type of kinematic hardening to be specified. Possible values are "HALF_CYCLE", "STABILIZED", and "PARAMETERS". The default value is "HALF_CYCLE".

This argument is applicable only if *kinematic*=true.

kinematicTemperature

A Boolean specifying whether the kinematic data depend on temperature. The default value is false.

This argument is applicable only if *kinematic*=true.

kinematicDependencies

An Int specifying the number of field variable dependencies for the kinematic data. The default value is 0.

This argument is applicable only if *kinematic*=true.

forcePotentialOperator

An odb_String specifying the contribution operator for the force potential contributions. Possible values are "SUM" and "MAXIMUM". The default value is "SUM".

This argument is applicable only if *coupling*="COUPLED".

forcePotentialExponent

A Double specifying the number equal to the inverse of the overall exponent in the force potential definition. The default value is 2.0.

This argument is applicable only if *coupling*="COUPLED" and if *forcePotentialOperator*="SUM".

connectorPotentials

A sequence of *ConnectorPotential* objects specifying one *ConnectorPotential* object for each force potential contribution. This member can be specified only if *coupling*="COUPLED".

isotropicTable

An odb_SequenceSequenceDouble specifying isotropic plasticity properties. Items in the *isotropicTable* data are described below. This argument is applicable only if *isotropic*=true. The default value is an empty sequence.

kinematicTable

An odb_SequenceSequenceDouble specifying kinematic plasticity properties. Items in the *kinematicTable* data are described below. This argument is applicable only if *kinematic*=true. The default value is an empty sequence.

components

An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. This argument can be specified only if *coupling*="UNCOUPLED". The default value is an empty sequence.

Table data

Table data for *isotropicTable*:

If *isotropicType*=TABULAR, then each sequence of the table data specifies the following:

- Equivalent yield force or moment defining the size of the elastic range.
- Equivalent relative plastic motion.
- Equivalent relative plastic motion rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *isotropicType*=EXPONENTIAL_LAW, then each sequence of the table data specifies the following:

- Equivalent force or moment defining the size of the elastic range at zero plastic motion.
- Isotropic hardening parameter $Q_{\inf}$.
- Isotropic hardening parameter b.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Table data for *kinematicTable*:

If *kinematicType*=HALF_CYCLE, then each sequence of the table data specifies the following:

- Yield force or moment.
- Connector relative plastic motion.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *kinematicType*=STABILIZED, then each sequence of the table data specifies the following:

- Yield force or moment.
- Connector relative plastic motion.
- Connector relative constitutive motion range.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *kinematicType*=PARAMETERS, then each sequence of the table data specifies the following:

- Yield force or moment at zero relative plastic motion.
- Kinematic hardening parameter C.
- Kinematic hardening parameter γ. Set γ=0 to specify linear Ziegler kinematic hardening.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConnectorPlasticity object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorPlasticity object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorPlasticity* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorPlasticity object has members with the same names and descriptions as the arguments to the *ConnectorPlasticity* method.

In addition, the ConnectorPlasticity object can have the following members:

Prototype

```
odb_ConnectorOptions isotropicOptions() const;
odb_ConnectorOptions kinematicOptions() const;
```

isotropicOptions

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the isotropic hardening table.

kinematicOptions

A *ConnectorOptions* object specifying the *ConnectorOptions* used to define tabular options for the kinematic hardening table.

Corresponding analysis keywords

- **CONNECTOR PLASTICITY*, **CONNECTOR HARDENING*, **CONNECTOR POTENTIAL*

ConnectorPotential object

The ConnectorPotential object is used to define a restricted set of mathematical functions to represent yield or limiting surfaces in the space spanned by connector available components. It can be used only in conjunction with *ConnectorDamage*, *ConnectorFriction*, and *ConnectorPlasticity* objects. Because the *ConnectorDamage* object contains two separate ConnectorPotential repositories (one for damage initiation and one for damage evolution), there are two ConnectorPotential constructors associated with that behavior—IniPotential and EvoPotential.

Access

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)
```

ConnectorPotential(...)

This method creates a connector potential object to be used in conjunction with an allowable connector behavior option.

Path

```
sectionApi.sections()[name].behaviorOptions(i).ConnectorPotential
```

Prototype

```
odb_ConnectorPotential&
ConnectorPotential(const odb_String& componentStyle,
                   int componentNumber,
                   const odb_String& sign,
                   double scaleFactor,
                   double positiveExponent,
                   double shiftFactor,
                   const odb_String& hFunction);
```

Required arguments

None.

Optional arguments

componentStyle

An odb_String specifying whether a component number or the name of the *DerivedComponent* object will be used in the contribution. Possible values are "COMPONENT_NUMBER" and "DERIVED_COMPONENT". The default value is "COMPONENT_NUMBER".

componentNumber

An Int specifying the component number used in the contribution. This argument is applicable only if *componentStyle*="COMPONENT_NUMBER". Possible values are 1 $\leq$*componentNumber*$\leq$ 6. Only available components can be specified. The default value is 0.

sign

An odb_String specifying the sign of the contribution. Possible values are "POSITIVE" and "NEGATIVE". The default value is "POSITIVE".

scaleFactor

A Double specifying the scaling factor for the contribution. The default value is 1.0.

positiveExponent A Double specifying the positive exponent for the contribution. The default value is 2.0.

This argument is ignored if the potential operator of the invoking behavior option is set to "MAXIMUM".

shiftFactor A Double specifying the shift factor for the contribution. The default value is 0.0.

hFunction An odb_String specifying the H function of the contribution: either absolute value, Macauley bracket, or the identity function. Possible values are "ABS", "MACAULEY", and "IDENTITY". The default value is "ABS".

The value of "IDENTITY" can be used only if *positiveExponent*=1.0 and the potential exponent of the invoking behavior option is also 1.0 (i.e., the potential operator of the invoking behavior option must be "SUM").

Return value

A ConnectorPotential object.

Exceptions

ValueError and TextError.

`setValues(...)`

This method modifies the ConnectorPotential object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorPotential* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorPotential object has members with the same names and descriptions as the arguments to the *ConnectorPotential* method.

In addition, the ConnectorPotential object can have the following member:

Prototype

```
odb_DerivedComponent derivedComponent() const;
```

derivedComponent A *DerivedComponent* object specifying the *DerivedComponent* used in the contribution. This argument is applicable only if *componentStyle*="DERIVED_COMPONENT".

Corresponding analysis keywords

- **CONNECTOR POTENTIAL*

ConnectorStop object

The ConnectorStop object defines connector stops for one or more components of a connector's relative motion.

The ConnectorStop object is derived from the *ConnectorBehaviorOption* object.

Access

```
sectionApi.sections()[name].behaviorOptions(i)
```

ConnectorStop(...)

This method creates a connector stop behavior option for a *ConnectorSection* object.

Path

```
sectionApi.sections()[name].ConnectorStop
```

Prototype

```
odb_ConnectorStop&
ConnectorStop(odb_Union minMotion,
              odb_Union maxMotion,
              const odb_SequenceInt& components);
```

Required arguments

None.

Optional arguments

minMotion

The string "NONE" or a Double specifying the lower bound for the connector's relative position for all specified components, or no lower bound. The default value is "NONE".

maxMotion

The string "NONE" or a Double specifying the upper bound for the connector's relative position for all specified components, or no upper bound. The default value is "NONE".

components

An odb_SequenceInt specifying the components of relative motion for which the behavior is defined. Possible values are $1 \leq components \leq 6$. Only available components can be specified. The default value is an empty sequence.

Return value

A ConnectorStop object.

Exceptions

ValueError and TextError.

setValues(...)

This method modifies the ConnectorStop object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ConnectorStop* method.

Return value

`None.`

Exceptions

ValueError.

Members

The ConnectorStop object has members with the same names and descriptions as the arguments to the *ConnectorStop* method.

Corresponding analysis keywords

- **CONNECTOR STOP*

DerivedComponent object

A DerivedComponent object describes user-customized components for use in defining *ConnectorFriction* and *Potential* objects.

Access

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.derivedComponent()
sectionApi.sections()[name].behaviorOptions(i).derivedComponent()
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.derivedComponent()
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.derivedComponent()
```

DerivedComponent()

This method creates a DerivedComponent object.

Path

```
sectionApi.sections()[name].behaviorOptions(i).connectorPotentials(i)\
.DerivedComponent
sectionApi.sections()[name].behaviorOptions(i).DerivedComponent
sectionApi.sections()[name].behaviorOptions(i).evolutionPotentials(i)\
.DerivedComponent
sectionApi.sections()[name].behaviorOptions(i).initiationPotentials(i)\
.DerivedComponent
```

Prototype

```
odb_DerivedComponent&
DerivedComponent();
```

Arguments

None.

Return value

A DerivedComponent object.

Exceptions

ValueError and TextError.

setValues(...)

This method modifies the DerivedComponent object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *DerivedComponent* method.

Return value

`None.`

Exceptions

ValueError.

Members

The DerivedComponent object can have the following member:

Prototype

```
odb_SequenceCDCTerm cdcTerms() const;
odb_CDCTerm cdcTerms(int index) const;
```

cdcTerms
A sequence of *CDCTerm* objects.

Corresponding analysis keywords

- **CONNECTOR DERIVED COMPONENT*

TangentialBehavior object

The TangentialBehavior object specifies tangential behavior for a connector friction behavior option.

Access

```
sectionApi.sections()[name].behaviorOptions(i).tangentialBehavior()
```

TangentialBehavior(...)

This method creates a TangentialBehavior object.

Path

```
sectionApi.sections()[name].behaviorOptions(i).TangentialBehavior
```

Prototype

```
odb_TangentialBehavior&
TangentialBehavior(const odb_String& formulation,
                bool slipRateDependency,
                bool pressureDependency,
                bool temperatureDependency,
                int dependencies,
                const odb_String& exponentialDecayDefinition,
                odb_Union shearStressLimit,
                const odb_String& maximumElasticSlip,
                double fraction,
                odb_Union absoluteDistance,
                const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

formulation An odb_String specifying the friction coefficient formulation. Possible values are "PENALTY" and "EXPONENTIAL_DECAY". The default value is "PENALTY".

slipRateDependency A Boolean specifying whether the data depend on slip rate. The default value is false.

pressureDependency A Boolean specifying whether the data depend on contact pressure. The default value is false.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variables for the data. The default value is 0.

exponentialDecayDefinition An odb_String specifying the exponential decay definition for the data. Possible values are "COEFFICIENTS" and "TEST_DATA". The default value is "COEFFICIENTS".

shearStressLimit

The string "NONE" or a Double specifying no upper limit or the friction coefficient shear stress limit. The default value is "NONE".

maximumElasticSlip

An odb_String specifying the method for modifying the allowable elastic slip. Possible values are "FRACTION" and "ABSOLUTE_DISTANCE". The default value is "FRACTION".

This argument applies only to Abaqus/Standard analyses.

fraction

A Double specifying the ratio of the allowable maximum elastic slip to a characteristic model dimension. The default value is 10^{-4}.

This argument applies only to Abaqus/Standard analyses.

absoluteDistance

The string "NONE" or a Double specifying the absolute magnitude of the allowable elastic slip. The default value is "NONE".

This argument applies only to Abaqus/Standard analyses.

table

An odb_SequenceSequenceDouble specifying the tangential properties. Items in the table data are described below. The default value is an empty sequence.

Table data

If *formulation*=PENALTY, the table data specify the following:

- Friction coefficient in the slip direction, μ.
- Slip rate, if the data depend on slip rate.
- Contact pressure, if the data depend on contact pressure.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=COEFFICIENTS, the table data specify the following:

- Static friction coefficient, μ_s.
- Kinetic friction coefficient, μ_k.
- Decay coefficient, d_c.

If *formulation*=EXPONENTIAL_DECAY and *exponentialDecayDefinition*=TEST_DATA, the table data specify the following:

- Static coefficient of friction.
- Dynamic friction coefficient measured at the reference slip rate, $\dot{\gamma}_2$.
- Reference slip rate, $\dot{\gamma}_2$, used to measure the dynamic friction coefficient.

- Kinetic friction coefficient, μ_∞. This value corresponds to the asymptotic value of the friction coefficient at infinite slip rate, $\dot{\gamma}_\infty$.

Return value

A TangentialBehavior object.

Exceptions

None.

`setValues(...)`

This method modifies the TangentialBehavior object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *TangentialBehavior* method.

Return value

`None.`

Exceptions

None.

Members

The TangentialBehavior object has members with the same names and descriptions as the arguments to the *TangentialBehavior* method.

Corresponding analysis keywords

- **FRICTION*

Filter commands

Filter commands are used to create real-time filters of output request data.

In this section:

- *Filter object*
- *ButterworthFilter object*
- *Chebyshev1Filter object*
- *Chebyshev2Filter object*
- *OperatorFilter object*

Filter object

The Filter object is the abstract base type for other Filter objects. The Filter object has no explicit constructor. The methods and members of the Filter object are common to all objects derived from the Filter.

Access

```
filterApi.filters()[name]
```

Members

The Filter object has the following members:

Prototype

```
odb_String name() const;
double cutoffFrequency() const;
int order() const;
odb_String operation() const;
bool halt() const;
odb_Union limit() const;
odb_String invariant() const;
```

name

An odb_String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency

A Double specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

order

An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation

An odb_String specifying the filter operator. Possible values are "NONE", "MIN", "MAX", and "ABS". The default value is "NONE".

halt

A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is false.

limit

The string "NONE" or a Double specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is "NONE".

invariant

An odb_String specifying the invariant to which filtering is applied. Possible values are "NONE", "FIRST", and "SECOND". The default value is "NONE".

ButterworthFilter object

The ButterworthFilter object defines a Butterworth type filter.

The ButterworthFilter object is derived from the *Filter* object.

Access

```
filterApi.filters()[name]
```

ButterworthFilter(...)

This method creates a ButterworthFilter object.

Path

```
filterApi.ButterworthFilter
```

Prototype

```
odb_ButterworthFilter&
ButterworthFilter(const odb_String& name,
                  double cutoffFrequency,
                  int order,
                  const odb_String& operation,
                  bool halt,
                  odb_Union limit,
                  const odb_String& invariant);
```

Required arguments

name

An odb_String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency

A Double specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

order

An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation

An odb_String specifying the filter operator. Possible values are "NONE", "MIN", "MAX", and "ABS". The default value is "NONE".

halt

A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is false.

limit

The string "NONE" or a Double specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is "NONE".

invariant An odb_String specifying the invariant to which filtering is applied. Possible values are "NONE", "FIRST", and "SECOND". The default value is "NONE".

Return value

A ButterworthFilter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the ButterworthFilter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *ButterworthFilter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The ButterworthFilter object has members with the same names and descriptions as the arguments to the *ButterworthFilter* method.

Corresponding analysis keywords

- **FILTER*

Chebyshev1Filter object

The Chebyshev1Filter object defines a Chebyshev type 1 filter.

The Chebyshev1Filter object is derived from the *Filter* object.

Access

```
filterApi.filters()[name]
```

Chebyshev1Filter(...)

This method creates a Chebyshev1Filter object.

Path

```
filterApi.Chebyshev1Filter
```

Prototype

```
odb_Chebyshev1Filter&
Chebyshev1Filter(const odb_String& name,
                 double cutoffFrequency,
                 double rippleFactor,
                 int order,
                 const odb_String& operation,
                 bool halt,
                 odb_Union limit,
                 const odb_String& invariant);
```

Required arguments

name

An odb_String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency

A Double specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

rippleFactor

A Double specifying the amount of allowable ripple in the filter. Possible values are non-negative numbers. The default value is 0.225.

order

An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation

An odb_String specifying the filter operator. Possible values are "NONE", "MIN", "MAX", and "ABS". The default value is "NONE".

halt

A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is false.

limit The string "NONE" or a Double specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is "NONE".

invariant An odb_String specifying the invariant to which filtering is applied. Possible values are "NONE", "FIRST", and "SECOND". The default value is "NONE".

Return value

A Chebyshev1Filter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the Chebyshev1Filter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Chebyshev1Filter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The Chebyshev1Filter object has members with the same names and descriptions as the arguments to the *Chebyshev1Filter* method.

Corresponding analysis keywords

- **FILTER*

Chebyshev2Filter object

The Chebyshev2Filter object defines a Chebyshev type 2 filter.

The Chebyshev2Filter object is derived from the *Filter* object.

Access

```
filterApi.filters()[name]
```

Chebyshev2Filter(...)

This method creates a Chebyshev2Filter object.

Path

```
filterApi.Chebyshev2Filter
```

Prototype

```
odb_Chebyshev2Filter&
Chebyshev2Filter(const odb_String& name,
                 double cutoffFrequency,
                 double rippleFactor,
                 int order,
                 const odb_String& operation,
                 bool halt,
                 odb_Union limit,
                 const odb_String& invariant);
```

Required arguments

name

An odb_String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency

A Double specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

rippleFactor

A Double specifying the amount of allowable ripple in the filter. Possible values are non-negative numbers less than 1. The default value is 0.025.

order

An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation

An odb_String specifying the filter operator. Possible values are "NONE", "MIN", "MAX", and "ABS". The default value is "NONE".

halt

A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is false.

limit The string "NONE" or a Double specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is "NONE".

invariant An odb_String specifying the invariant to which filtering is applied. Possible values are "NONE", "FIRST", and "SECOND". The default value is "NONE".

Return value

A Chebyshev2Filter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the Chebyshev2Filter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *Chebyshev2Filter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The Chebyshev2Filter object has members with the same names and descriptions as the arguments to the *Chebyshev2Filter* method.

Corresponding analysis keywords

- **FILTER*

OperatorFilter object

The OperatorFilter object defines an operator filter.

The OperatorFilter object is derived from the *Filter* object.

Access

```
filterApi.filters()[name]
```

OperatorFilter(...)

This method creates an OperatorFilter object.

Path

```
filterApi.OperatorFilter
```

Prototype

```
odb_OperatorFilter&
OperatorFilter(const odb_String& name,
               double cutoffFrequency,
               int order,
               const odb_String& operation,
               bool halt,
               odb_Union limit,
               const odb_String& invariant);
```

Required arguments

name

An odb_String specifying the repository key. This name ANTIALIASING is reserved for filters generated internally by the program.

cutoffFrequency

A Double specifying the attenuation point of the filter. Possible values are non-negative numbers. Order is not available for OperatorFilter.

Optional arguments

order

An Int specifying the highest power of the filter transfer function. Possible values are non-negative numbers less than or equal to 20. Order is not available for OperatorFilter. The default value is 2.

operation

An odb_String specifying the filter operator. Possible values are "NONE", "MIN", "MAX", and "ABS". The default value is "NONE".

halt

A Boolean specifying whether to stop the analysis if the specified limit is reached. The default value is false.

limit

The string "NONE" or a Double specifying the threshold limit, an upper or lower bound for output values depending on the operation, or a bound for stopping the analysis when Halt is used. The default value is "NONE".

invariant An odb_String specifying the invariant to which filtering is applied. Possible values are "NONE", "FIRST", and "SECOND". The default value is "NONE".

Return value

An OperatorFilter object.

Exceptions

InvalidNameError and RangeError.

`setValues(...)`

This method modifies the OperatorFilter object.

Required arguments

None.

Optional arguments

The optional arguments to `setValues` are the same as the arguments to the *OperatorFilter* method, except for the *name* argument.

Return value

`None.`

Exceptions

RangeError.

Members

The OperatorFilter object has members with the same names and descriptions as the arguments to the *OperatorFilter* method.

Corresponding analysis keywords

- **FILTER*

Material commands

The Material commands are used to define the materials in an ODB.

In this section:

- *Material object*
- *AcousticMedium object*
- *AnnealTemperature object*
- *BiaxialTestData object*
- *BrittleCracking object*
- *BrittleFailure object*
- *BrittleShear object*
- *CapCreepCohesion object*
- *CapCreepConsolidation object*
- *CapHardening object*
- *CapPlasticity object*
- *CastIronCompressionHardening object*
- *CastIronPlasticity object*
- *CastIronTensionHardening object*
- *ClayHardening object*
- *ClayPlasticity object*
- *CombinedTestData object*
- *Concrete object*
- *ConcreteCompressionDamage object*
- *ConcreteCompressionHardening object*
- *ConcreteDamagedPlasticity object*
- *ConcreteTensionDamage object*
- *ConcreteTensionStiffening object*
- *Conductivity object*
- *ContactArea object*
- *Creep object*
- *CrushableFoam object*
- *CrushableFoamHardening object*
- *CycledPlastic object*
- *CyclicHardening object*
- *DamageEvolution object*
- *DamageInitiation object*
- *DamageStabilization object*
- *DamageStabilizationCohesive object*
- *Damping object*
- *DeformationPlasticity object*
- *Density object*
- *Depvar object*

- *DetonationPoint object*
- *Dielectric object*
- *Diffusivity object*
- *DruckerPrager object*
- *DruckerPragerCreep object*
- *DruckerPragerHardening object*
- *Elastic object*
- *ElectricalConductivity object*
- *Eos object*
- *EosCompaction object*
- *Expansion object*
- *FailStrain object*
- *FailStress object*
- *FailureRatios object*
- *FluidLeakoff object*
- *GapConductance object*
- *GapConvection object*
- *GapFlow object*
- *GapRadiation object*
- *GasketMembraneElastic object*
- *GasketThicknessBehavior object*
- *GasketTransverseShearElastic object*
- *Gel object*
- *HeatGeneration object*
- *Hyperelastic object*
- *Hyperfoam object*
- *Hypoelastic object*
- *Hysteresis object*
- *InelasticHeatFraction object*
- *JouleHeatFraction object*
- *LatentHeat object*
- *LowDensityFoam object*
- *MagneticPermeability object*
- *MohrCoulombHardening object*
- *MohrCoulombPlasticity object*
- *MoistureSwelling object*
- *MullinsEffect object*
- *Ornl object*
- *Permeability object*
- *Piezoelectric object*
- *PlanarTestData object*
- *Plastic object*
- *PoreFluidExpansion object*
- *PorousBulkModuli object*
- *PorousElastic object*
- *PorousFailureCriteria object*

- *PorousMetalPlasticity object*
- *Potential object*
- *PressureEffect object*
- *RateDependent object*
- *Ratios object*
- *Regularization object*
- *SaturationDependence object*
- *ShearRetention object*
- *ShearTestData object*
- *SimpleShearTestData object*
- *Solubility object*
- *SoretEffect object*
- *Sorption object*
- *SpecificHeat object*
- *SuperElasticHardening object*
- *SuperElasticHardeningModifications object*
- *SuperElasticity object*
- *Swelling object*
- *TensileFailure object*
- *TensionCutOff object*
- *TensionStiffening object*
- *TriaxialTestData object*
- *Trs object*
- *UniaxialTestData object*
- *UserDefinedField object*
- *UserMaterial object*
- *UserOutputVariables object*
- *VelocityDependence object*
- *Viscoelastic object*
- *Viscosity object*
- *Viscous object*
- *VoidNucleation object*
- *VolumetricTestData object*

Material object

A Material object is the object used to specify a material. The Material object stores the various settings that determine how a material behaves.

A material is created by combining one or more individual material options and sub options. A particular material option is associated with the Material object through a member. For example: the *acousticMedium* member may contain an AcousticMedium object. The alternative of having a MaterialOption abstract base class and a container of MaterialOptions was rejected because it would make it more difficult to enforce the fact that one Material object cannot contain two AcousticMedium objects, for example.

Access

```
materialApi.materials()[name]
```

Material(...)

This method creates a Material object.

Path

```
materialApi.Material
```

Prototype

```
odb_Material&
Material(const odb_String& name,
         const odb_String& description,
         const odb_String& materialIdentifier);
```

Required arguments

name An odb_String specifying the name of the new material.

Optional arguments

description An odb_String specifying user description of the material. The default value is an empty string.

materialIdentifier An odb_String specifying material identifier for customer use. The default value is an empty string.

Return value

A Material object.

Exceptions

InvalidNameError.

Members

The Material object has members with the same names and descriptions as the arguments to the *Material* method.

In addition, the Material object can have the following members:

Prototype

```
odb_AcousticMedium acousticMedium() const;
odb_BrittleCracking brittleCracking() const;
odb_CapPlasticity capPlasticity() const;
odb_CastIronPlasticity castIronPlasticity() const;
odb_ClayPlasticity clayPlasticity() const;
odb_Concrete concrete() const;
odb_ConcreteDamagedPlasticity concreteDamagedPlasticity() const;
odb_Conductivity conductivity() const;
odb_Creep creep() const;
odb_CrushableFoam crushableFoam() const;
odb_DamageInitiation ductileDamageInitiation() const;
odb_DamageInitiation fldDamageInitiation() const;
odb_DamageInitiation flsdDamageInitiation() const;
odb_DamageInitiation johnsonCookDamageInitiation() const;
odb_DamageInitiation maxeDamageInitiation() const;
odb_DamageInitiation maxsDamageInitiation() const;
odb_DamageInitiation maxpeDamageInitiation() const;
odb_DamageInitiation maxpsDamageInitiation() const;
odb_DamageInitiation mkDamageInitiation() const;
odb_DamageInitiation msfldDamageInitiation() const;
odb_DamageInitiation quadeDamageInitiation() const;
odb_DamageInitiation quadsDamageInitiation() const;
odb_DamageInitiation shearDamageInitiation() const;
odb_DamageInitiation hashinDamageInitiation() const;
odb_Damping damping() const;
odb_DeformationPlasticity deformationPlasticity() const;
odb_Density density() const;
odb_Depvar depvar() const;
odb_Dielectric dielectric() const;
odb_Diffusivity diffusivity() const;
odb_DruckerPrager druckerPrager() const;
odb_Elastic elastic() const;
odb_ElectricalConductivity electricalConductivity() const;
odb_Eos eos() const;
odb_Expansion expansion() const;
odb_FluidLeakoff fluidLeakoff() const;
odb_GapFlow gapFlow() const;
odb_GasketThicknessBehavior gasketThicknessBehavior() const;
odb_GasketTransverseShearElastic gasketTransverseShearElastic() const;
odb_GasketMembraneElastic gasketMembraneElastic() const;
odb_Gel gel() const;
odb_HeatGeneration heatGeneration() const;
odb_Hyperelastic hyperelastic() const;
odb_Hyperfoam hyperfoam() const;
odb_Hypoelastic hypoelastic() const;
odb_InelasticHeatFraction inelasticHeatFraction() const;
odb_JouleHeatFraction jouleHeatFraction() const;
odb_LatentHeat latentHeat() const;
odb_LowDensityFoam lowDensityFoam() const;
odb_MagneticPermeability magneticPermeability() const;
odb_MohrCoulombPlasticity mohrCoulombPlasticity() const;
odb_MoistureSwelling moistureSwelling() const;
odb_MullinsEffect mullinsEffect() const;
odb_Permeability permeability() const;
odb_Piezoelectric piezoelectric() const;
odb_Plastic plastic() const;
odb_PoreFluidExpansion poreFluidExpansion() const;
```

```
odb_PorousBulkModuli porousBulkModuli() const;
odb_PorousElastic porousElastic() const;
odb_PorousMetalPlasticity porousMetalPlasticity() const;
odb_Regularization regularization() const;
odb_Solubility solubility() const;
odb_Sorption sorption() const;
odb_SpecificHeat specificHeat() const;
odb_Swelling swelling() const;
odb_UserDefinedField userDefinedField() const;
odb_UserMaterial userMaterial() const;
odb_UserOutputVariables userOutputVariables() const;
odb_Viscoelastic viscoelastic() const;
odb_Viscosity viscosity() const;
odb_Viscous viscous() const;
```

acousticMedium
An *AcousticMedium* object.

brittleCracking
A *BrittleCracking* object.

capPlasticity
A *CapPlasticity* object.

castIronPlasticity
A *CastIronPlasticity* object.

clayPlasticity
A *ClayPlasticity* object.

concrete
A *Concrete* object.

concreteDamagedPlasticity
A *ConcreteDamagedPlasticity* object.

conductivity
A *Conductivity* object.

creep
A *Creep* object.

crushableFoam
A *CrushableFoam* object.

ductileDamageInitiation
A *DamageInitiation* object.

fldDamageInitiation
A *DamageInitiation* object.

flsdDamageInitiation
A *DamageInitiation* object.

johnsonCookDamageInitiation
A *DamageInitiation* object.

maxeDamageInitiation
A *DamageInitiation* object.

maxsDamageInitiation
A *DamageInitiation* object.

maxpeDamageInitiation
A *DamageInitiation* object.

maxpsDamageInitiation
A *DamageInitiation* object.

mkDamageInitiation A *DamageInitiation* object.

msfldDamageInitiation A *DamageInitiation* object.

quadeDamageInitiation A *DamageInitiation* object.

quadsDamageInitiation A *DamageInitiation* object.

shearDamageInitiation A *DamageInitiation* object.

hashinDamageInitiation A *DamageInitiation* object.

damping A *Damping* object.

deformationPlasticity A *DeformationPlasticity* object.

density A *Density* object.

depvar A *Depvar* object.

dielectric A *Dielectric* object.

diffusivity A *Diffusivity* object.

druckerPrager A *DruckerPrager* object.

elastic An *Elastic* object.

electricalConductivity An *ElectricalConductivity* object.

eos An *Eos* object.

expansion An *Expansion* object.

fluidLeakoff A *FluidLeakoff* object.

gapFlow A *GapFlow* object.

gasketThicknessBehavior A *GasketThicknessBehavior* object.

gasketTransverseShearElastic A *GasketTransverseShearElastic* object.

gasketMembraneElastic A *GasketMembraneElastic* object.

gel A *Gel* object.

heatGeneration A *HeatGeneration* object.

hyperelastic A *Hyperelastic* object.

hyperfoam A *Hyperfoam* object.

hypoelastic A *Hypoelastic* object.

inelasticHeatFraction An *InelasticHeatFraction* object.

jouleHeatFraction A *JouleHeatFraction* object.

latentHeat A *LatentHeat* object.

lowDensityFoam A *LowDensityFoam* object.

magneticPermeability A *MagneticPermeability* object.

mohrCoulombPlasticity A *MohrCoulombPlasticity* object.

moistureSwelling A *MoistureSwelling* object.

mullinsEffect A *MullinsEffect* object.

permeability A *Permeability* object.

piezoelectric A *Piezoelectric* object.

plastic A *Plastic* object.

poreFluidExpansion A *PoreFluidExpansion* object.

porousBulkModuli A *PorousBulkModuli* object.

porousElastic A *PorousElastic* object.

porousMetalPlasticity A *PorousMetalPlasticity* object.

regularization A *Regularization* object.

solubility A *Solubility* object.

sorption A *Sorption* object.

specificHeat A *SpecificHeat* object.

swelling A *Swelling* object.

userDefinedField A *UserDefinedField* object.

userMaterial	A *UserMaterial* object.
userOutputVariables	A *UserOutputVariables* object.
viscoelastic	A *Viscoelastic* object.
viscosity	A *Viscosity* object.
viscous	A *Viscous* object.

Corresponding analysis keywords

- **MATERIAL*

AcousticMedium object

The AcousticMedium object specifies the acoustic properties of a material.

Access

```
materialApi.materials()[name].acousticMedium()
```

AcousticMedium(...)

This method creates an AcousticMedium object.

Path

```
materialApi.materials()[name].AcousticMedium
```

Prototype

```
odb_AcousticMedium&
AcousticMedium(bool acousticVolumetricDrag,
           bool temperatureDependencyB,
           bool temperatureDependencyV,
           int dependenciesB,
           int dependenciesV,
           const odb_SequenceSequenceDouble& bulkTable,
           const odb_SequenceSequenceDouble& volumetricTable);
```

Required arguments

None.

Optional arguments

acousticVolumetricDrag

A Boolean specifying whether the volumetricTable data is specified. The default value is false.

temperatureDependencyB

A Boolean specifying whether the data in *bulkTable* depend on temperature. The default value is false.

temperatureDependencyV

A Boolean specifying whether the data in *volumetricTable* depend on temperature. The default value is false.

dependenciesB

An Int specifying the number of field variable dependencies for the data in *bulkTable*. The default value is 0.

dependenciesV

An Int specifying the number of field variable dependencies for the data in *volumetricTable*. The default value is 0.

bulkTable

An odb_SequenceSequenceDouble specifying the following:

- Bulk modulus.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

The default value is an empty sequence.

volumetricTable

An odb_SequenceSequenceDouble specifying the following:

- Volumetric drag.
- Frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

The default value is an empty sequence.

Return value

An AcousticMedium object.

Exceptions

RangeError.

Members

The AcousticMedium object has members with the same names and descriptions as the arguments to the *AcousticMedium* method.

Corresponding analysis keywords

- **ACOUSTIC MEDIUM*

AnnealTemperature object

The AnnealTemperature object specifies the material annealing temperature.

Access

```
materialApi.materials()[name].plastic().annealTemperature()
```

AnnealTemperature(...)

This method creates an AnnealTemperature object.

Path

```
materialApi.materials()[name].plastic().AnnealTemperature
```

Prototype

```
odb_AnnealTemperature&
AnnealTemperature(const odb_SequenceSequenceDouble& table,
                  int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The annealing temperature, θ.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

An AnnealTemperature object.

Exceptions

RangeError.

Members

The AnnealTemperature object has members with the same names and descriptions as the arguments to the *AnnealTemperature* method.

Corresponding analysis keywords

- **ANNEAL TEMPERATURE*

BiaxialTestData object

The BiaxialTestData object provides equibiaxial test data (compression and/or tension).

Access

```
materialApi.materials()[name].hyperelastic().biaxialTestData()
materialApi.materials()[name].hyperfoam().biaxialTestData()
materialApi.materials()[name].mullinsEffect().biaxialTests(i)
```

BiaxialTestData(...)

This method creates a BiaxialTestData object.

Path

```
materialApi.materials()[name].hyperelastic().BiaxialTestData
materialApi.materials()[name].hyperfoam().BiaxialTestData
materialApi.materials()[name].mullinsEffect().BiaxialTestData
```

Prototype

```
odb_BiaxialTestData&
BiaxialTestData(const odb_SequenceSequenceDouble& table,
                odb_Union smoothing,
                bool lateralNominalStrain,
                bool temperatureDependency,
                int dependencies);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the following:

- Nominal stress, $\boldsymbol{T_B}$.
- Nominal strain, $\boldsymbol{\epsilon_B}$.

Optional arguments

smoothing

The string "NONE" or an Int specifying the value for smoothing. If *smoothing*="NONE", no smoothing is employed. The default value is "NONE".

lateralNominalStrain

A Boolean specifying whether to include lateral nominal strain. The default value is false.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Return value

A BiaxialTestData object.

Exceptions

None.

Members

The BiaxialTestData object has members with the same names and descriptions as the arguments to the *BiaxialTestData* method.

Corresponding analysis keywords

- **BIAXIAL TEST DATA*

BrittleCracking object

The BrittleCracking object specifies cracking and postcracking properties for the brittle cracking material model.

Access

```
materialApi.materials()[name].brittleCracking()
```

BrittleCracking(...)

This method creates a BrittleCracking object.

Path

```
materialApi.materials()[name].BrittleCracking
```

Prototype

```
odb_BrittleCracking&
BrittleCracking(const odb_SequenceSequenceDouble& table,
                bool temperatureDependency,
                int dependencies,
                const odb_String& type);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

type
An odb_String specifying the type of postcracking behavior. Possible values are "STRAIN", "DISPLACEMENT", and "GFI". The default value is "STRAIN".

Table data

If *type*=STRAIN the table data specify the following:

- Remaining direct stress after cracking.
- Direct cracking strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT the table data specify the following:

- Remaining direct stress after cracking.
- Direct cracking displacement.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=GFI the table data specify the following:

- Failure stress.
- Mode I fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleCracking object.

Exceptions

None.

Members

The BrittleCracking object has members with the same names and descriptions as the arguments to the *BrittleCracking* method.

In addition, the BrittleCracking object can have the following members:

Prototype

```
odb_BrittleShear brittleShear() const;
odb_BrittleFailure brittleFailure() const;
```

brittleShear A *BrittleShear* object.

brittleFailure A *BrittleFailure* object.

Corresponding analysis keywords

- **BRITTLE CRACKING*

BrittleFailure object

The BrittleFailure object specifies the brittle failure of the material.

Access

```
materialApi.materials()[name].brittleCracking().brittleFailure()
```

BrittleFailure(...)

This method creates a BrittleFailure object.

Path

```
materialApi.materials()[name].brittleCracking().BrittleFailure
```

Prototype

```
odb_BrittleFailure&
BrittleFailure(const odb_SequenceSequenceDouble& table,
               bool temperatureDependency,
               int dependencies,
               const odb_String& failureCriteria);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

failureCriteria
An odb_String specifying the failure criteria. Possible values are "UNIDIRECTIONAL", "BIDIRECTIONAL", and "TRIDIRECTIONAL". The default value is "UNIDIRECTIONAL".

Table data

If parent *BrittleCracking* member *type*=STRAIN the table data specify the following:

- Direct cracking failure strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If parent *BrittleCracking* member *type*=DISPLACEMENT or *type*=GFI the table data specify the following:

- Direct cracking failure displacement.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleFailure object.

Exceptions

RangeError.

Members

The BrittleFailure object has members with the same names and descriptions as the arguments to the *BrittleFailure* method.

Corresponding analysis keywords

- **BRITTLE FAILURE*

BrittleShear object

The BrittleShear object specifies the postcracking shear behavior of a material used in a brittle cracking model.

Access

```
materialApi.materials()[name].brittleCracking().brittleShear()
```

BrittleShear(...)

This method creates a BrittleShear object.

Path

```
materialApi.materials()[name].brittleCracking().BrittleShear
```

Prototype

```
odb_BrittleShear&
BrittleShear(const odb_SequenceSequenceDouble& table,
             bool temperatureDependency,
             int dependencies,
             const odb_String& type);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

type
An odb_String specifying the type of postcracking shear behavior. Possible values are "RETENTION_FACTOR" and "POWER_LAW". The default value is "RETENTION_FACTOR".

Table data

If *type*=RETENTION_FACTOR the table data specify the following:

- Shear retention factor.
- Crack opening strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=POWER_LAW the table data specify the following:

- e.
- p.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BrittleShear object.

Exceptions

RangeError.

Members

The BrittleShear object has members with the same names and descriptions as the arguments to the *BrittleShear* method.

Corresponding analysis keywords

- **BRITTLE SHEAR*

CapCreepCohesion object

The CapCreepCohesion object specifies a cap creep model and material properties.

Access

```
materialApi.materials()[name].capPlasticity().capCreepCohesion()
```

CapCreepCohesion(...)

This method creates a CapCreepCohesion object.

Path

```
materialApi.materials()[name].capPlasticity().CapCreepCohesion
```

Prototype

```
odb_CapCreepCohesion&
CapCreepCohesion(const odb_SequenceSequenceDouble& table,
                 const odb_String& law,
                 bool temperatureDependency,
                 int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law
An odb_String specifying the cap creep law. Possible values are "STRAIN", "TIME", "SINGHM", and "USER". The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A.
- α.
- m.
- t_1.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapCreepCohesion object.

Exceptions

None.

Members

The CapCreepCohesion object has members with the same names and descriptions as the arguments to the *CapCreepCohesion* method.

Corresponding analysis keywords

- **CAP CREEP*

CapCreepConsolidation object

The CapCreepConsolidation object specifies a cap creep model and material properties.

Access

```
materialApi.materials()[name].capPlasticity().capCreepConsolidation()
```

CapCreepConsolidation(...)

This method creates a CapCreepConsolidation object.

Path

```
materialApi.materials()[name].capPlasticity().CapCreepConsolidation
```

Prototype

```
odb_CapCreepConsolidation&
CapCreepConsolidation(const odb_SequenceSequenceDouble& table,
                      const odb_String& law,
                      bool temperatureDependency,
                      int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law
An odb_String specifying the cap creep law. Possible values are "STRAIN", "TIME", "SINGHM", and "USER". The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A.
- α.
- m.
- t_1.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapCreepConsolidation object.

Exceptions

None.

Members

The CapCreepConsolidation object has members with the same names and descriptions as the arguments to the *CapCreepConsolidation* method.

Corresponding analysis keywords

- **CAP CREEP*

CapHardening object

The CapHardening object specifies Drucker-Prager/Cap plasticity hardening.

Access

```
materialApi.materials()[name].capPlasticity().capHardening()
```

CapHardening(...)

This method creates a CapHardening object.

Path

```
materialApi.materials()[name].capPlasticity().CapHardening
```

Prototype

```
odb_CapHardening&
CapHardening(const odb_SequenceSequenceDouble& table,
             bool temperatureDependency,
             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Hydrostatic pressure yield stress.
- Absolute value of the corresponding volumetric inelastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapHardening object.

Exceptions

RangeError.

Members

The CapHardening object has members with the same names and descriptions as the arguments to the *CapHardening* method.

Corresponding analysis keywords

- **CAP HARDENING*

CapPlasticity object

The CapPlasticity object specifies the modified Drucker-Prager/Cap plasticity model.

Access

```
materialApi.materials()[name].capPlasticity()
```

CapPlasticity(...)

This method creates a CapPlasticity object.

Path

```
materialApi.materials()[name].CapPlasticity
```

Prototype

```
odb_CapPlasticity&
CapPlasticity(const odb_SequenceSequenceDouble& table,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Material cohesion, d, in the p–t plane (Abaqus/Standard) or in the p–q plane (Abaqus/Explicit).
- Material angle of friction, β, in the p–t plane (Abaqus/Standard) or in the p–q plane (Abaqus/Explicit). Give the value in degrees.
- Cap eccentricity parameter, R. Its value must be greater than zero (typically $0.0 < R < 1.0$).
- Initial cap yield surface position, $\varepsilon_{vol}^{pl}\big|_0$.
- Transition surface radius parameter, α. The default value is 0.0 (i.e., no transition surface).
- (Not used in Abaqus/Explicit) K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. Possible values are $0.778 \le K \le 1.0$. If the default value of 0.0 is accepted, Abaqus/Standard assumes $K = 1.0$.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CapPlasticity object.

Exceptions

RangeError.

Members

The CapPlasticity object has members with the same names and descriptions as the arguments to the *CapPlasticity* method.

In addition, the CapPlasticity object can have the following members:

Prototype

```
odb_CapCreepCohesion capCreepCohesion() const;
odb_CapCreepConsolidation capCreepConsolidation() const;
odb_CapHardening capHardening() const;
```

capCreepCohesion
A *CapCreepCohesion* object.

capCreepConsolidation
A *CapCreepConsolidation* object.

capHardening
A *CapHardening* object.

Corresponding analysis keywords

- **CAP PLASTICITY*

CastIronCompressionHardening object

The CastIronCompressionHardening object specifies hardening for the Cast- Iron plasticity model.

Access

```
materialApi.materials()[name].castIronPlasticity()\
.castIronCompressionHardening()
```

CastIronCompressionHardening(...)

This method creates a CastIronCompressionHardening object.

Path

```
materialApi.materials()[name].castIronPlasticity()\
.CastIronCompressionHardening
```

Prototype

```
odb_CastIronCompressionHardening&
CastIronCompressionHardening(const odb_SequenceSequenceDouble& table,
                             bool temperatureDependency,
                             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in compression, σ_c.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronCompressionHardening object.

Exceptions

RangeError.

Members

The CastIronCompressionHardening object has members with the same names and descriptions as the arguments to the *CastIronCompressionHardening* method.

Corresponding analysis keywords

- **CAST IRON COMPRESSION HARDENING*

CastIronPlasticity object

The CastIronPlasticity object specifies the Cast Iron plasticity model.

Access

```
materialApi.materials()[name].castIronPlasticity()
```

CastIronPlasticity(...)

This method creates a CastIronPlasticity object.

Path

```
materialApi.materials()[name].CastIronPlasticity
```

Prototype

```
odb_CastIronPlasticity&
CastIronPlasticity(const odb_SequenceSequenceDouble& table,
                   bool temperatureDependency,
                   int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Plastic Poisson's ratio, ν_{pl} (dimensionless).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronPlasticity object.

Exceptions

RangeError.

Members

The CastIronPlasticity object has members with the same names and descriptions as the arguments to the *CastIronPlasticity* method.

In addition, the CastIronPlasticity object can have the following members:

Prototype

```
odb_CastIronTensionHardening castIronTensionHardening() const;
odb_CastIronCompressionHardening castIronCompressionHardening() const;
```

castIronTensionHardening

A *CastIronTensionHardening* object.

castIronCompressionHardening

A *CastIronCompressionHardening* object.

Corresponding analysis keywords

- **CAST IRON PLASTICITY*

CastIronTensionHardening object

The CastIronTensionHardening object specifies hardening for the Cast- Iron plasticity model.

Access

```
materialApi.materials()[name].castIronPlasticity()\
.castIronTensionHardening()
```

CastIronTensionHardening(...)

This method creates a CastIronTensionHardening object.

Path

```
materialApi.materials()[name].castIronPlasticity()\
.CastIronTensionHardening
```

Prototype

```
odb_CastIronTensionHardening&
CastIronTensionHardening(const odb_SequenceSequenceDouble& table,
                         bool temperatureDependency,
                         int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in uniaxial tension, σ_t.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CastIronTensionHardening object.

Exceptions

RangeError.

Members

The CastIronTensionHardening object has members with the same names and descriptions as the arguments to the *CastIronTensionHardening* method.

Corresponding analysis keywords

- **CAST IRON TENSION HARDENING*

ClayHardening object

The ClayHardening object specifies hardening for the clay plasticity model.

Access

```
materialApi.materials()[name].clayPlasticity().clayHardening()
```

ClayHardening(...)

This method creates a ClayHardening object.

Path

```
materialApi.materials()[name].clayPlasticity().ClayHardening
```

Prototype

```
odb_ClayHardening&
ClayHardening(const odb_SequenceSequenceDouble& table,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The hydrostatic pressure stress at yield, p_c.
- The absolute value of the corresponding volumetric plastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ClayHardening object.

Exceptions

RangeError.

Members

The ClayHardening object has members with the same names and descriptions as the arguments to the *ClayHardening* method.

Corresponding analysis keywords

- **CLAY HARDENING*

ClayPlasticity object

The ClayPlasticity object specifies the extended Cam-clay plasticity model.

Access

```
materialApi.materials()[name].clayPlasticity()
```

ClayPlasticity(...)

This method creates a ClayPlasticity object.

Path

```
materialApi.materials()[name].ClayPlasticity
```

Prototype

```
odb_ClayPlasticity&
ClayPlasticity(const odb_SequenceSequenceDouble& table,
               odb_Union intercept,
               const odb_String& hardening,
               bool temperatureDependency,
               int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

intercept
The string "NONE" or a Double specifying e_1, the intercept of the virgin consolidation line with the void ratio axis in a plot of void ratio versus the logarithm of pressure stress. The default value is "NONE".

This argument is valid only if *hardening*="EXPONENTIAL".

hardening
An odb_String specifying the type of hardening/softening definition. Possible values are "EXPONENTIAL" and "TABULAR". The default value is "EXPONENTIAL".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *hardening*=EXPONENTIAL, the table data specify the following:

- Logarithmic plastic bulk modulus, λ (dimensionless).
- Stress ratio at critical state, M.
- The initial yield surface size, α_0.

- β, the parameter defining the size of the yield surface on the “wet” side of critical state.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \le K \le 1.0$. If the default value of 0.0 is accepted, a value of 1.0 is assumed.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=TABULAR, the table data specify the following:

- Stress ratio at critical state, M.
- The initial volumetric plastic strain, $\varepsilon_{\text{vol}}^{pl}\big|_0$, corresponding to $p_c|_0$ according to the *ClayHardening* definition.
- β, the parameter defining the size of the yield surface on the “wet” side of critical state.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \le K \le 1.0$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ClayPlasticity object.

Exceptions

RangeError.

Members

The ClayPlasticity object has members with the same names and descriptions as the arguments to the *ClayPlasticity* method.

In addition, the ClayPlasticity object can have the following member:

Prototype

```
odb_ClayHardening clayHardening() const;
```

clayHardening A *ClayHardening* object.

Corresponding analysis keywords

- **CLAY PLASTICITY*

CombinedTestData object

The CombinedTestData object specifies simultaneously the normalized shear and bulk compliances or relaxation moduli as functions of time.

Access

```
materialApi.materials()[name].viscoelastic().combinedTestData()
```

CombinedTestData(...)

This method creates a CombinedTestData object.

Path

```
materialApi.materials()[name].viscoelastic().CombinedTestData
```

Prototype

```
odb_CombinedTestData&
CombinedTestData(const odb_SequenceSequenceDouble& table,
                 odb_Union volinf,
                 odb_Union shrinf);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below. The values of the table data depend on the value of the *time* member of the *Viscoelastic* object.

Optional arguments

volinf

The string "NONE" or a Double specifying a normalized volume. The value of *volinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is "NONE".

If *time*="RELAXATION_TEST_DATA", *volinf* specifies the value of the long-term normalized volumetric modulus, $k_R(\infty)$.

If *time*="CREEP_TEST_DATA", *volinf* specifies the value of the long-term normalized volumetric compliance, $j_K(\infty)$.

shrinf

The string "NONE" or a Double specifying a normalized shear. The value of *shrinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is "NONE".

If *time*="RELAXATION_TEST_DATA", *shrinf* specifies the value of the long-term normalized shear modulus, $g_R(\infty)$.

If *time*="CREEP_TEST_DATA", *shrinf* specifies the value of the long-term normalized shear compliance, $j_S(\infty)$.

Table data

If *time*=RELAXATION_TEST_DATA, the table data specify the following:

- Normalized shear modulus, $g_R\ (t) (0 \le g_R\ (t) \le 1)$.
- Normalized volumetric (bulk) modulus, $k_R\ (t) (0 \le k_R\ (t) \le 1)$.
- Time $t (t > 0)$.

If *time*=CREEP_TEST_DATA, the table data specify the following:

- Normalized shear compliance, $j_S\ (t) (j_S\ (t) \ge 1)$.
- Normalized volumetric (bulk) compliance, $j_K\ (t) (j_K\ (t) \ge 1)$.
- Time $t (t > 0)$.

Return value

A CombinedTestData object.

Exceptions

None.

Members

The CombinedTestData object has members with the same names and descriptions as the arguments to the *CombinedTestData* method.

Corresponding analysis keywords

- **COMBINED TEST DATA*

Concrete object

The Concrete object defines concrete properties beyond the elastic range.

Access

```
materialApi.materials()[name].concrete()
```

Concrete(...)

This method creates a Concrete object.

Path

```
materialApi.materials()[name].Concrete
```

Prototype

```
odb_Concrete&
Concrete(const odb_SequenceSequenceDouble& table,
         bool temperatureDependency,
         int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Absolute value of compressive stress.
- Absolute value of plastic strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Concrete object.

Exceptions

RangeError.

Members

The Concrete object has members with the same names and descriptions as the arguments to the *Concrete* method.

In addition, the Concrete object can have the following members:

Prototype

```
odb_FailureRatios failureRatios() const;
odb_ShearRetention shearRetention() const;
odb_TensionStiffening tensionStiffening() const;
```

failureRatios A *FailureRatios* object.

shearRetention A *ShearRetention* object.

tensionStiffening A *TensionStiffening* object.

Corresponding analysis keywords

- **CONCRETE*

ConcreteCompressionDamage object

The ConcreteCompressionDamage object specifies hardening for the concrete damaged plasticity model.

Access

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.concreteCompressionDamage()
```

ConcreteCompressionDamage(...)

This method creates a ConcreteCompressionDamage object.

Path

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.ConcreteCompressionDamage
```

Prototype

```
odb_ConcreteCompressionDamage&
ConcreteCompressionDamage(const odb_SequenceSequenceDouble& table,
                          double tensionRecovery,
                          bool temperatureDependency,
                          int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

tensionRecovery
A Double specifying the value of the stiffness recovery factor, w_t, that determines the amount of tension stiffness that is recovered as loading changes from compression to tension. The default value is 0.0.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Compressive damage variable, d_c.
- Inelastic (crushing) strain, ϵ_c^{in}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteCompressionDamage object.

Exceptions

RangeError.

Members

The ConcreteCompressionDamage object has members with the same names and descriptions as the arguments to the *ConcreteCompressionDamage* method.

Corresponding analysis keywords

- **CONCRETE COMPRESSION DAMAGE*

ConcreteCompressionHardening object

The ConcreteCompressionHardening object specifies hardening for the concrete damaged plasticity model.

Access

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.concreteCompressionHardening()
```

ConcreteCompressionHardening(...)

This method creates a ConcreteCompressionHardening object.

Path

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.ConcreteCompressionHardening
```

Prototype

```
odb_ConcreteCompressionHardening&
ConcreteCompressionHardening(const odb_SequenceSequenceDouble& table,
                             bool rate,
                             bool temperatureDependency,
                             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

rate
A Boolean specifying whether the data depend on rate. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress in compression, σ_c.
- Inelastic (crushing) strain, ϵ_c^{in}.
- Inelastic (crushing) strain rate, $\dot{\epsilon}_c^{in}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteCompressionHardening object.

Exceptions

RangeError.

Members

The ConcreteCompressionHardening object has members with the same names and descriptions as the arguments to the *ConcreteCompressionHardening* method.

Corresponding analysis keywords

- **CONCRETE COMPRESSION HARDENING*

ConcreteDamagedPlasticity object

The ConcreteDamagedPlasticity object specifies the concrete damaged plasticity model.

Access

```
materialApi.materials()[name].concreteDamagedPlasticity()
```

ConcreteDamagedPlasticity(...)

This method creates a ConcreteDamagedPlasticity object.

Path

```
materialApi.materials()[name].ConcreteDamagedPlasticity
```

Prototype

```
odb_ConcreteDamagedPlasticity&
ConcreteDamagedPlasticity(const odb_SequenceSequenceDouble& table,
                          bool temperatureDependency,
                          int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

The table data specify the following:

- Dilation angle, ψ (in degrees) in the p–q plane.
- Flow potential eccentricity, ϵ. The default value is 0.1.
- σ_{b0}/σ_{t0}, the ratio of initial equibiaxial compressive yield stress to initial uniaxial compressive yield stress. The default value is 1.16.
- K_c, the ratio of the second stress invariant on the tensile meridian, to that on the compressive meridian, at initial yield for any given value of the pressure invariant p such that the maximum principal stress is negative. The default value is 2/3.
- Viscosity parameter, μ, used for the viscoplastic regularization of the concrete constitutive equations in an Abaqus/Standard analysis. This parameter is ignored in an Abaqus/Explicit analysis. The default value is 0.0.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteDamagedPlasticity object.

Exceptions

RangeError.

Members

The ConcreteDamagedPlasticity object has members with the same names and descriptions as the arguments to the *ConcreteDamagedPlasticity* method.

In addition, the ConcreteDamagedPlasticity object can have the following members:

Prototype

```
odb_ConcreteCompressionHardening concreteCompressionHardening() const;
odb_ConcreteTensionStiffening concreteTensionStiffening() const;
odb_ConcreteCompressionDamage concreteCompressionDamage() const;
odb_ConcreteTensionDamage concreteTensionDamage() const;
```

concreteCompressionHardening	A *ConcreteCompressionHardening* object.
concreteTensionStiffening	A *ConcreteTensionStiffening* object.
concreteCompressionDamage	A *ConcreteCompressionDamage* object.
concreteTensionDamage	A *ConcreteTensionDamage* object.

Corresponding analysis keywords

- **CONCRETE DAMAGED PLASTICITY*

ConcreteTensionDamage object

The ConcreteTensionDamage object specifies hardening for the concrete damaged plasticity model.

Access

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.concreteTensionDamage()
```

ConcreteTensionDamage(...)

This method creates a ConcreteTensionDamage object.

Path

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.ConcreteTensionDamage
```

Prototype

```
odb_ConcreteTensionDamage&
ConcreteTensionDamage(const odb_SequenceSequenceDouble& table,
                      double compressionRecovery,
                      const odb_String& type,
                      bool temperatureDependency,
                      int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

compressionRecovery
A Double specifying the value of the stiffness recovery factor, w_c, that determines the amount of compression stiffness that is recovered as loading changes from tension to compression. The default value is 1.0.

type
An odb_String specifying the type of tensile damage data. Set *type*="STRAIN" to specify the tensile damage variable as a function of cracking strain. Set *type*="DISPLACEMENT" to specify the tensile damage variable as a function of cracking displacement. Possible values are "STRAIN" and "DISPLACEMENT". The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Tensile damage variable, d_t.

- Direct cracking strain, ϵ_t^{ck}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Tensile damage variable, d_t.
- Direct cracking displacement, u_t^{ck}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteTensionDamage object.

Exceptions

RangeError.

Members

The ConcreteTensionDamage object has members with the same names and descriptions as the arguments to the *ConcreteTensionDamage* method.

Corresponding analysis keywords

- **CONCRETE TENSION DAMAGE*

ConcreteTensionStiffening object

The ConcreteTensionStiffening object specifies hardening for the concrete damaged plasticity model.

Access

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.concreteTensionStiffening()
```

ConcreteTensionStiffening(...)

This method creates a ConcreteTensionStiffening object.

Path

```
materialApi.materials()[name].concreteDamagedPlasticity()\
.ConcreteTensionStiffening
```

Prototype

```
odb_ConcreteTensionStiffening&
ConcreteTensionStiffening(const odb_SequenceSequenceDouble& table,
                          bool rate,
                          const odb_String& type,
                          bool temperatureDependency,
                          int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

rate
A Boolean specifying whether the data depend on rate. The default value is false.

type
An odb_String specifying the type of postcracking behavior data. Possible values are:

- "STRAIN", specifying postfailure stress as a function of cracking strain.
- "DISPLACEMENT", specifying postfailure stress as a function of cracking displacement.
- "GFI", specifying failure stress as a function of the fracture energy.

The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Remaining direct stress after cracking, σ_t.

- Direct cracking strain, ϵ_t^{ck}.
- Direct cracking strain rate, $\dot{\epsilon}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Remaining direct stress after cracking, σ_t.
- Direct cracking displacement, u_t^{ck}.
- Direct cracking displacement rate, $\dot{u}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=GFI, the table data specify the following:

- Failure stress, σ_{t0}.
- Fracture energy, G_f.
- Direct cracking displacement rate, $\dot{u}_t^{ck}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ConcreteTensionStiffening object.

Exceptions

RangeError.

Members

The ConcreteTensionStiffening object has members with the same names and descriptions as the arguments to the *ConcreteTensionStiffening* method.

Corresponding analysis keywords

- **CONCRETE TENSION STIFFENING*

Conductivity object

The Conductivity object specifies thermal conductivity.

Access

```
materialApi.materials()[name].conductivity()
```

Conductivity(...)

This method creates a Conductivity object.

Path

```
materialApi.materials()[name].Conductivity
```

Prototype

```
odb_Conductivity&
Conductivity(const odb_SequenceSequenceDouble& table,
             const odb_String& type,
             bool temperatureDependency,
             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the type of conductivity. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Conductivity, k.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- k_{11}.

- k_{22}.
- k_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- k_{11}.
- k_{12}.
- k_{22}.
- k_{13}.
- k_{23}.
- k_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Conductivity object.

Exceptions

RangeError.

Members

The Conductivity object has members with the same names and descriptions as the arguments to the *Conductivity* method.

Corresponding analysis keywords

- **CONDUCTIVITY*

ContactArea object

A ContactArea object specifies a suboption of gasket thickness behavior when *variableUnits*=FORCE on the GasketThicknessBehavior object. The ContactArea object defines the contact area or contact width versus closure curves to output an average pressure through variable CS11.

Access

```
materialApi.materials()[name].gasketThicknessBehavior().contactArea()
```

ContactArea(...)

This method creates a ContactArea object.

Path

```
materialApi.materials()[name].gasketThicknessBehavior().ContactArea
```

Prototype

```
odb_ContactArea&
ContactArea(const odb_SequenceSequenceDouble& table,
            bool temperatureDependency,
            int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether contact area data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies included in the definition of the contact area data, in addition to temperature. The default value is 0.

Table data

- Contact area or width; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ContactArea object.

Exceptions

None.

Members

The ContactArea object has members with the same names and descriptions as the arguments to the *ContactArea* method.

Corresponding analysis keywords

- **GASKET CONTACT AREA*

Creep object

The Creep object defines a creep law.

Access

```
materialApi.materials()[name].creep()
```

Creep(...)

This method creates a Creep object.

Path

```
materialApi.materials()[name].Creep
```

Prototype

```
odb_Creep&
Creep(const odb_SequenceSequenceDouble& table,
      const odb_String& law,
      bool temperatureDependency,
      int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law An odb_String specifying the strain-hardening law. Possible values are "STRAIN", "TIME", "HYPERBOLIC_SINE", and "USER". The default value is "STRAIN".

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=HYPERBOLIC_SINE, the table data specify the following:

- A.
- B.
- n.
- ΔH, if the data depend on temperature.
- R.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Creep object.

Exceptions

RangeError.

Members

The Creep object has members with the same names and descriptions as the arguments to the *Creep* method.

In addition, the Creep object can have the following members:

Prototype

```
odb_Ornl ornl() const;
odb_Potential potential() const;
```

ornl

An *Ornl* object.

potential

A *Potential* object.

Corresponding analysis keywords

- **CREEP*

CrushableFoam object

The CrushableFoam object specifies the crushable foam plasticity model.

Access

```
materialApi.materials()[name].crushableFoam()
```

CrushableFoam(...)

This method creates a CrushableFoam object.

Path

```
materialApi.materials()[name].CrushableFoam
```

Prototype

```
odb_CrushableFoam&
CrushableFoam(const odb_SequenceSequenceDouble& table,
              const odb_String& hardening,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

hardening
An odb_String specifying the type of hardening/softening definition. Possible values are "VOLUMETRIC" and "ISOTROPIC". The default value is "VOLUMETRIC".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *hardening*=VOLUMETRIC, the table data specify the following:

- Ratio, k, of initial yield stress in uniaxial compression, σ_c^0, to initial yield stress in hydrostatic compression, p_c^0; $0.0 < k < 3.0$.
- Ratio, k_t, of yield stress in hydrostatic tension, p_t, to initial yield stress in hydrostatic compression, p_c^0. The default value is 1.0.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *hardening*=ISOTROPIC, the table data specify the following:

- Ratio, k, of initial yield stress in uniaxial compression, σ_c^0, to initial yield stress in hydrostatic compression, p_c^0; $0.0 \leq k \leq 3.0$.
- Plastic Poisson's ratio.ν_p; $-1 \leq \nu_p \leq 0.5$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CrushableFoam object.

Exceptions

RangeError.

Members

The CrushableFoam object has members with the same names and descriptions as the arguments to the *CrushableFoam* method.

In addition, the CrushableFoam object can have the following members:

Prototype

```
odb_CrushableFoamHardening crushableFoamHardening() const;
odb_RateDependent rateDependent() const;
```

crushableFoamHardening A *CrushableFoamHardening* object.

rateDependent A *RateDependent* object.

Corresponding analysis keywords

- **CRUSHABLE FOAM*

CrushableFoamHardening object

The CrushableFoamHardening object specifies hardening for the crushable foam plasticity model.

Access

```
materialApi.materials()[name].crushableFoam().crushableFoamHardening()
```

CrushableFoamHardening(...)

This method creates a CrushableFoamHardening object.

Path

```
materialApi.materials()[name].crushableFoam().CrushableFoamHardening
```

Prototype

```
odb_CrushableFoamHardening&
CrushableFoamHardening(const odb_SequenceSequenceDouble& table,
                       bool temperatureDependency,
                       int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- The yield stress in uniaxial compression, σ_c.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CrushableFoamHardening object.

Exceptions

RangeError.

Members

The CrushableFoamHardening object has members with the same names and descriptions as the arguments to the *CrushableFoamHardening* method.

Corresponding analysis keywords

- **CRUSHABLE FOAM HARDENING*

CycledPlastic object

The CycledPlastic object specifies cycled yield stress data for the ORNL constitutive model.

Access

```
materialApi.materials()[name].plastic().cycledPlastic()
```

CycledPlastic(...)

This method creates a CycledPlastic object.

Path

```
materialApi.materials()[name].plastic().CycledPlastic
```

Prototype

```
odb_CycledPlastic&
CycledPlastic(const odb_SequenceSequenceDouble& table,
              bool temperatureDependency);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

Table data

- Yield stress.
- Plastic strain.
- Temperature, if the data depend on temperature.

Return value

A CycledPlastic object.

Exceptions

None.

Members

The CycledPlastic object has members with the same names and descriptions as the arguments to the *CycledPlastic* method.

Corresponding analysis keywords

- **CYCLED PLASTIC*

CyclicHardening object

The CyclicHardening object defines the evolution of the elastic domain for the nonlinear isotropic/kinematic hardening model.

Access

```
materialApi.materials()[name].plastic().cyclicHardening()
```

CyclicHardening(...)

This method creates a CyclicHardening object.

Path

```
materialApi.materials()[name].plastic().CyclicHardening
```

Prototype

```
odb_CyclicHardening&
CyclicHardening(const odb_SequenceSequenceDouble& table,
                bool temperatureDependency,
                int dependencies,
                bool parameters);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

parameters A Boolean specifying whether material parameters are to be input directly. The default value is false.

Table data

- Equivalent stress.
- Q_∞ (only if *parameters*=ON).
- Hardening parameter (only if *parameters*=ON).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A CyclicHardening object.

Exceptions

None.

Members

The CyclicHardening object has members with the same names and descriptions as the arguments to the *CyclicHardening* method.

Corresponding analysis keywords

- **CYCLIC HARDENING*

DamageEvolution object

The DamageEvolution object specifies material properties to define the evolution of damage.

Access

```
materialApi.materials()[name].ductileDamageInitiation()\
.damageEvolution()
materialApi.materials()[name].fldDamageInitiation().damageEvolution()
materialApi.materials()[name].flsdDamageInitiation().damageEvolution()
materialApi.materials()[name].hashinDamageInitiation().damageEvolution()
materialApi.materials()[name].johnsonCookDamageInitiation()\
.damageEvolution()
materialApi.materials()[name].maxeDamageInitiation().damageEvolution()
materialApi.materials()[name].maxpeDamageInitiation().damageEvolution()
materialApi.materials()[name].maxpsDamageInitiation().damageEvolution()
materialApi.materials()[name].maxsDamageInitiation().damageEvolution()
materialApi.materials()[name].mkDamageInitiation().damageEvolution()
materialApi.materials()[name].msfldDamageInitiation().damageEvolution()
materialApi.materials()[name].quadeDamageInitiation().damageEvolution()
materialApi.materials()[name].quadsDamageInitiation().damageEvolution()
materialApi.materials()[name].shearDamageInitiation().damageEvolution()
```

DamageEvolution(...)

This method creates a DamageEvolution object.

Path

```
materialApi.materials()[name].ductileDamageInitiation().DamageEvolution
materialApi.materials()[name].fldDamageInitiation().DamageEvolution
materialApi.materials()[name].flsdDamageInitiation().DamageEvolution
materialApi.materials()[name].hashinDamageInitiation().DamageEvolution
materialApi.materials()[name].johnsonCookDamageInitiation()\
.DamageEvolution
materialApi.materials()[name].maxeDamageInitiation().DamageEvolution
materialApi.materials()[name].maxpeDamageInitiation().DamageEvolution
materialApi.materials()[name].maxpsDamageInitiation().DamageEvolution
materialApi.materials()[name].maxsDamageInitiation().DamageEvolution
materialApi.materials()[name].mkDamageInitiation().DamageEvolution
materialApi.materials()[name].msfldDamageInitiation().DamageEvolution
materialApi.materials()[name].quadeDamageInitiation().DamageEvolution
materialApi.materials()[name].quadsDamageInitiation().DamageEvolution
materialApi.materials()[name].shearDamageInitiation().DamageEvolution
```

Prototype

```
odb_DamageEvolution&
DamageEvolution(const odb_String& type,
                const odb_SequenceSequenceDouble& table,
                const odb_String& degradation,
                bool temperatureDependency,
                int dependencies,
                const odb_String& mixedModeBehavior,
                const odb_String& modeMixRatio,
```

```
                odb_Union power,
                const odb_String& softening);
```

Required arguments

type
An odb_String specifying the type of damage evolution. Possible values are "DISPLACEMENT" and "ENERGY".

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

degradation
An odb_String specifying the degradation. Possible values are "MAXIMUM" and "MULTIPLICATIVE". The default value is "MAXIMUM".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

mixedModeBehavior
An odb_String specifying the mixed mode behavior. Possible values are "MODE_INDEPENDENT", "TABULAR", "POWER_LAW", and "BK". The default value is "MODE_INDEPENDENT".

modeMixRatio
An odb_String specifying the mode mix ratio. Possible values are "ENERGY" and "TRACTION". The default value is "ENERGY".

power
The string "NONE" or a Double specifying the exponent in the power law or the Benzeggagh-Kenane criterion that defines the variation of fracture energy with mode mix for cohesive elements. The default value is "NONE".

softening
An odb_String specifying the softening. Possible values are "LINEAR", "EXPONENTIAL", and "TABULAR". The default value is "LINEAR".

Table data

If *type*=DISPLACEMENT, and *softening*=LINEAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Equivalent total or plastic displacement at failure, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Fracture energy.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=LINEAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Fracture energy.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Equivalent total or plastic displacement at failure, measured from the time of damage initiation.
- Exponential law parameter.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Total displacement at failure, measured from the time of damage initiation.

- Exponential law parameter.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=EXPONENTIAL, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Fracture energy.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=TABULAR, and *mixedModeBehavior*=MODE_INDEPENDENT, the table data specify the following:

- Damage variable.
- Equivalent total or plastic displacement, measured from the time of damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, and *softening*=TABULAR, and *mixedModeBehavior*=TABULAR, the table data specify the following:

- Damage variable.
- Equivalent total or plastic displacement, measured from the time of damage initiation.
- Appropriate mode mix ratio.
- Appropriate mode mix ratio (if relevant, for three-dimensional problems with anisotropic shear behavior).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, and *softening*=LINEAR or EXPONENTIAL, and *mixedModeBehavior*=POWER_LAW or BK, the table data specify the following:

- Normal mode fracture energy.
- Shear mode fracture energy for failure in the first shear direction.

- Shear mode fracture energy for failure in the second shear direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENERGY, *softening*=LINEAR and constructor for *DamageInitiation*=HashinDamageInitiation the table data specify the following:

- Fiber tensile fracture energy.
- Fiber compressive fracture energy.
- Matrix tensile fracture energy.
- Matrix compressive fracture energy.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DamageEvolution object.

Exceptions

RangeError.

Members

The DamageEvolution object has members with the same names and descriptions as the arguments to the *DamageEvolution* method.

Corresponding analysis keywords

- **DAMAGE EVOLUTION*

DamageInitiation object

The DamageInitiation object specifies material properties to define the initiation of damage.

Access

```
materialApi.materials()[name].ductileDamageInitiation()
materialApi.materials()[name].fldDamageInitiation()
materialApi.materials()[name].flsdDamageInitiation()
materialApi.materials()[name].hashinDamageInitiation()
materialApi.materials()[name].johnsonCookDamageInitiation()
materialApi.materials()[name].maxeDamageInitiation()
materialApi.materials()[name].maxpeDamageInitiation()
materialApi.materials()[name].maxpsDamageInitiation()
materialApi.materials()[name].maxsDamageInitiation()
materialApi.materials()[name].mkDamageInitiation()
materialApi.materials()[name].msfldDamageInitiation()
materialApi.materials()[name].quadeDamageInitiation()
materialApi.materials()[name].quadsDamageInitiation()
materialApi.materials()[name].shearDamageInitiation()
```

DuctileDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].DuctileDamageInitiation
```

Prototype

```
odb_DamageInitiation&
DuctileDamageInitiation(const odb_SequenceSequenceDouble& table,
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Table data

If constructor is DuctileDamageInitiation, the table data specify the following:

- Equivalent fracture strain at damage initiation.
- Stress triaxiality.
- Strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If constructor is FldDamageInitiation, the table data specify the following:

- Major principal strain at damage initiation.
- Minor principal strain.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor FlsdDamageInitiation, the table data specify the following:

- Major principal stress at damage initiation.
- Minor principal stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is JohnsonCookDamageInitiation, the table data specify the following:

- Johnson-Cook failure parameter D1.
- Johnson-Cook failure parameter D2.
- Johnson-Cook failure parameter D3.
- Johnson-Cook failure parameter D4.
- Johnson-Cook failure parameter D5.
- Melting temperature.
- Transition temperature.
- Reference strain rate.

If constructor MkDamageInitiation, the table data specify the following:

- Flaw size relative to nominal thickness of the section.
- Angle (in degrees) with respect to the 1-direction of the local material orientation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is MsfldDamageInitiation and *definition*=MSFLD, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Equivalent plastic strain at initiation of localized necking.
- Ratio of minor to major principal strains.
- Equivalent plastic strain rate.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is MsfldDamageInitiation and *definition*=FLD, the table data specify the following:

- Major principal strain at initiation of localized necking.
- Equivalent plastic strain at initiation of localized necking.
- Ratio of minor to major principal strains.
- Equivalent plastic strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is QuadeDamageInitiation or MaxeDamageInitiation, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the first shear direction.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the second shear direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is QuadsDamageInitiation or MaxsDamageInitiation, the table data specify the following:

- Nominal strain at damage initiation in a normal-only mode.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the first shear direction.
- Nominal strain at damage initiation in a shear-only mode that involves separation only along the second shear direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is MaxpeDamageInitiation, the table data specify the following:

- Maximum principal strain at damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is MaxpsDamageInitiation, the table data specify the following:

- Maximum principal stress at damage initiation.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is ShearDamageInitiation, the table data specify the following:

- Equivalent fracture strain at damage initiation.
- Shear stress ratio.
- Strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If constructor is HashinDamageInitiation, the table data specify the following:

- Fiber tensile strength.
- Fiber compressive strength.
- Matrix tensile strength.
- Matrix compressive strength.
- Longitudinal shear strength.
- Transverse shear strength.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DamageInitiation object.

Exceptions

RangeError.

FldDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].FldDamageInitiation
```

Prototype

```
odb_DamageInitiation&
FldDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                    const odb_String& definition,
                    double feq,
                    double fnn,
                    double fnt,
                    int frequency,
                    double ks,
                    int numberImperfections,
                    bool temperatureDependency,
                    int dependencies,
                    double alpha,
                    double omega,
                    double tolerance,
                    const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

FlsdDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].FlsdDamageInitiation
```

Prototype

```
odb_DamageInitiation&
FlsdDamageInitiation(const odb_SequenceSequenceDouble& table,
                     const odb_String& definition,
                     double feq,
                     double fnn,
                     double fnt,
                     int frequency,
                     double ks,
                     int numberImperfections,
                     bool temperatureDependency,
                     int dependencies,
                     double alpha,
                     double omega,
                     double tolerance,
                     const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

JohnsonCookDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].JohnsonCookDamageInitiation
```

Prototype

```
odb_DamageInitiation&
JohnsonCookDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                                const odb_String& definition,
                                double feq,
                                double fnn,
                                double fnt,
                                int frequency,
                                double ks,
                                int numberImperfections,
                                bool temperatureDependency,
                                int dependencies,
                                double alpha,
                                double omega,
                                double tolerance,
                                const odb_String& direction);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition
An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq
A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MaxeDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MaxeDamageInitiation(const odb_SequenceSequenceDouble& table,
                     const odb_String& definition,
                     double feq,
                     double fnn,
                     double fnt,
                     int frequency,
                     double ks,
                     int numberImperfections,
                     bool temperatureDependency,
                     int dependencies,
                     double alpha,
                     double omega,
                     double tolerance,
                     const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MaxsDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MaxsDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition
An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq
A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MkDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MkDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MkDamageInitiation(const odb_SequenceSequenceDouble& table,
                   const odb_String& definition,
                   double feq,
                   double fnn,
                   double fnt,
                   int frequency,
                   double ks,
                   int numberImperfections,
                   bool temperatureDependency,
                   int dependencies,
                   double alpha,
                   double omega,
                   double tolerance,
                   const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MsfldDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MsfldDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MsfldDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition
An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq
A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

QuadeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].QuadeDamageInitiation
```

Prototype

```
odb_DamageInitiation&
QuadeDamageInitiation(const odb_SequenceSequenceDouble& table,
                      const odb_String& definition,
                      double feq,
                      double fnn,
                      double fnt,
                      int frequency,
                      double ks,
                      int numberImperfections,
                      bool temperatureDependency,
                      int dependencies,
                      double alpha,
                      double omega,
                      double tolerance,
                      const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

QuadsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].QuadsDamageInitiation
```

Prototype

```
odb_DamageInitiation&
QuadsDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxpeDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MaxpeDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MaxpeDamageInitiation(const odb_SequenceSequenceDouble& table,
                      const odb_String& definition,
                      double feq,
                      double fnn,
                      double fnt,
                      int frequency,
                      double ks,
                      int numberImperfections,
                      bool temperatureDependency,
                      int dependencies,
                      double alpha,
                      double omega,
                      double tolerance,
                      const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

MaxpsDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].MaxpsDamageInitiation
```

Prototype

```
odb_DamageInitiation&
MaxpsDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

ShearDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].ShearDamageInitiation
```

Prototype

```
odb_DamageInitiation&
ShearDamageInitiation(const odb_SequenceSequenceDouble& table,
                      const odb_String& definition,
                      double feq,
                      double fnn,
                      double fnt,
                      int frequency,
                      double ks,
                      int numberImperfections,
                      bool temperatureDependency,
                      int dependencies,
                      double alpha,
                      double omega,
                      double tolerance,
                      const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn
A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt
A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency
An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

HashinDamageInitiation(...)

This method creates a DamageInitiation object.

Path

```
materialApi.materials()[name].HashinDamageInitiation
```

Prototype

```
odb_DamageInitiation&
HashinDamageInitiation(const odb_SequenceSequenceDouble& table,
```

```
                        const odb_String& definition,
                        double feq,
                        double fnn,
                        double fnt,
                        int frequency,
                        double ks,
                        int numberImperfections,
                        bool temperatureDependency,
                        int dependencies,
                        double alpha,
                        double omega,
                        double tolerance,
                        const odb_String& direction);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

Optional arguments

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks A Double specifying the value of Ks. The default value is 0.0.

numberImperfections An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

alpha A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

Return value

A DamageInitiation object.

Exceptions

RangeError.

Members

The DamageInitiation object can have the following members:

Prototype

```
odb_String definition() const;
double feq() const;
double fnn() const;
double fnt() const;
int frequency() const;
double ks() const;
int numberImperfections() const;
bool temperatureDependency() const;
int dependencies() const;
double alpha() const;
double omega() const;
double tolerance() const;
odb_String direction() const;
odb_SequenceSequenceDouble table() const;
odb_DamageEvolution damageEvolution() const;
odb_DamageStabilization damageStabilization() const;
odb_DamageStabilizationCohesive damageStabilizationCohesive() const;
```

definition An odb_String specifying the damage initiation definition. Possible values are "FLD" and "MSFLD". The default value is "MSFLD".

feq A Double specifying the critical value of the deformation severity index for equivalent plastic strains. The default value is 10.0.

fnn A Double specifying the critical value of the deformation severity index for strains normal to the groove direction. The default value is 10.0.

fnt A Double specifying the critical value of the deformation severity index for shear strains. The default value is 10.0.

frequency An Int specifying the frequency, in increments, at which the Marciniak-Kuczynski analysis is going to be performed. The default value is 1.

ks
A Double specifying the value of Ks. The default value is 0.0.

numberImperfections
An Int specifying the number of imperfections to be considered for the evaluation of the Marciniak-Kuczynski analysis. These imperfections are assumed to be equally spaced in the angular direction. The default value is 4.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

alpha
A Double specifying the value of the coefficient that will multiply the shear contribution to the Hashin's fiber initiation criterion. The default value is 0.0.

omega
A Double specifying the factor used for filtering the ratio of principal strain rates used for the evaluation of the MSFLD damage initiation criterion. The default value is 1.0.

tolerance
A Double specifying the tolerance within which the damage initiation criterion must be satisfied. The default value is 0.05.

direction
An odb_String specifying the damage initiation direction. Possible values are "NMORI" and "TMORI". The default value is "NMORI".

table
An odb_SequenceSequenceDouble specifying the items described in the "Table data" section.

damageEvolution
A *DamageEvolution* object.

damageStabilization
A *DamageStabilization* object.

damageStabilizationCohesive
A *DamageStabilizationCohesive* object.

Corresponding analysis keywords

- **DAMAGE INITIATION*

DamageStabilization object

The DamageStabilization object specifies the viscosity coefficients for the damage model for fiber-reinforced materials.

Access

```
materialApi.materials()[name].ductileDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].fldDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].flsdDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].hashinDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].johnsonCookDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].maxeDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].maxpeDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].maxpsDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].maxsDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].mkDamageInitiation().damageStabilization()
materialApi.materials()[name].msfldDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].quadeDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].quadsDamageInitiation()\
.damageStabilization()
materialApi.materials()[name].shearDamageInitiation()\
.damageStabilization()
```

DamageStabilization(...)

This method creates a DamageStabilization object.

Path

```
materialApi.materials()[name].ductileDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].fldDamageInitiation().DamageStabilization
materialApi.materials()[name].flsdDamageInitiation().DamageStabilization
materialApi.materials()[name].hashinDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].johnsonCookDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].maxeDamageInitiation().DamageStabilization
materialApi.materials()[name].maxpeDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].maxpsDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].maxsDamageInitiation().DamageStabilization
materialApi.materials()[name].mkDamageInitiation().DamageStabilization
```

```
materialApi.materials()[name].msfldDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].quadeDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].quadsDamageInitiation()\
.DamageStabilization
materialApi.materials()[name].shearDamageInitiation()\
.DamageStabilization
```

Prototype

```
odb_DamageStabilization&
DamageStabilization(double fiberTensileCoeff,
                    double fiberCompressiveCoeff,
                    double matrixTensileCoeff,
                    double matrixCompressiveCoeff);
```

Required arguments

fiberTensileCoeff	A Double specifying the viscosity coefficient for the fiber tensile failure mode.
fiberCompressiveCoeff	A Double specifying the viscosity coefficient for the fiber compressive failure mode.
matrixTensileCoeff	A Double specifying the viscosity coefficient for the matrix tensile failure mode.
matrixCompressiveCoeff	A Double specifying the viscosity coefficient for the matrix compressive failure mode.

Optional arguments

None.

Return value

A DamageStabilization object.

Exceptions

RangeError.

Members

The DamageStabilization object has members with the same names and descriptions as the arguments to the *DamageStabilization* method.

Corresponding analysis keywords

- **DAMAGE STABILIZATION*

DamageStabilizationCohesive object

The DamageStabilizationCohesive object specifies the viscosity coefficients for the damage model for surface-based cohesive behavior or enriched cohesive behavior.

Access

```
materialApi.materials()[name].ductileDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].fldDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].flsdDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].hashinDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].johnsonCookDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].maxeDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].maxpeDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].maxpsDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].maxsDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].mkDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].msfldDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].quadeDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].quadsDamageInitiation()\
.damageStabilizationCohesive()
materialApi.materials()[name].shearDamageInitiation()\
.damageStabilizationCohesive()
```

DamageStabilizationCohesive(...)

This method creates a DamageStabilizationCohesive object.

Path

```
materialApi.materials()[name].ductileDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].fldDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].flsdDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].hashinDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].johnsonCookDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].maxeDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].maxpeDamageInitiation()\
.DamageStabilizationCohesive
```

```
materialApi.materials()[name].maxpsDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].maxsDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].mkDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].msfldDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].quadeDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].quadsDamageInitiation()\
.DamageStabilizationCohesive
materialApi.materials()[name].shearDamageInitiation()\
.DamageStabilizationCohesive
```

Prototype

```
odb_DamageStabilizationCohesive&
DamageStabilizationCohesive(odb_Union cohesiveCoeff);
```

Required arguments

None.

Optional arguments

cohesiveCoeff

The string "NONE" or a Double specifying the viscosity coefficient. The default value is "NONE".

Return value

A DamageStabilizationCohesive object.

Exceptions

RangeError.

Members

The DamageStabilizationCohesive object has members with the same names and descriptions as the arguments to the *DamageStabilizationCohesive* method.

Corresponding analysis keywords

- **DAMAGE STABILIZATION*

Damping object

The Damping object specifies material damping.

Access

```
materialApi.materials()[name].damping()
```

Damping(...)

This method creates a Damping object.

Path

```
materialApi.materials()[name].Damping
```

Prototype

```
odb_Damping&
Damping(double alpha,
        double beta,
        double composite,
        double structural);
```

Required arguments

None.

Optional arguments

alpha

A Double specifying the α_R factor to create mass proportional damping in direct-integration and explicit dynamics. The default value is 0.0.

beta

A Double specifying the β_R factor to create stiffness proportional damping in direct-integration and explicit dynamics. The default value is 0.0.

composite

A Double specifying the fraction of critical damping to be used with this material in calculating composite damping factors for the modes (for use in modal dynamics). The default value is 0.0.

This argument applies only to Abaqus/Standard analyses.

structural

A Double specifying the structural factor to create material damping in direct-integration and explicit dynamics. The default value is 0.0.

Return value

A Damping object.

Exceptions

RangeError.

Members

The Damping object has members with the same names and descriptions as the arguments to the *Damping* method.

Corresponding analysis keywords

- **DAMPING*

DeformationPlasticity object

The DeformationPlasticity object specifies the deformation plasticity model.

Access

```
materialApi.materials()[name].deformationPlasticity()
```

DeformationPlasticity(...)

This method creates a DeformationPlasticity object.

Path

```
materialApi.materials()[name].DeformationPlasticity
```

Prototype

```
odb_DeformationPlasticity&
DeformationPlasticity(const odb_SequenceSequenceDouble& table,
                      bool temperatureDependency);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

Table data

- Young's modulus, E.
- Poisson's ratio, ν.
- Yield stress, σ^0.
- Exponent, n.
- Yield offset, α.
- Temperature, if the data depend on temperature.

Return value

A DeformationPlasticity object.

Exceptions

RangeError.

Members

The DeformationPlasticity object has members with the same names and descriptions as the arguments to the *DeformationPlasticity* method.

Corresponding analysis keywords

- **DEFORMATION PLASTICITY*

Density object

The Density object specifies the material density.

Access

```
materialApi.materials()[name].density()
```

Density(...)

This method creates a Density object.

Path

```
materialApi.materials()[name].Density
```

Prototype

```
odb_Density&
Density(const odb_SequenceSequenceDouble& table,
        bool temperatureDependency,
        int dependencies,
        const odb_String& distributionType,
        const odb_String& fieldName);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

distributionType
An odb_String specifying how the density is distributed spatially. Possible values are "UNIFORM", "ANALYTICAL_FIELD", and "DISCRETE_FIELD". The default value is "UNIFORM".

fieldName
An odb_String specifying the name of the *AnalyticalField object* or *DiscreteField object* object associated with this material option. The *fieldName* argument applies only when *distributionType*="ANALYTICAL_FIELD" or *distributionType*="DISCRETE_FIELD". The default value is an empty string.

Table data

- The mass density.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

Return value

A Density object.

Exceptions

RangeError.

Members

The Density object has members with the same names and descriptions as the arguments to the *Density* method.

Corresponding analysis keywords

- **DENSITY*

Depvar object

The Depvar object specifies solution-dependent state variables.

Access

```
materialApi.materials()[name].depvar()
```

Depvar(...)

This method creates a Depvar object.

Path

```
materialApi.materials()[name].Depvar
```

Prototype

```
odb_Depvar&
Depvar(int deleteVar,
       int n);
```

Required arguments

None.

Optional arguments

deleteVar
An Int specifying the state variable number controlling the element deletion flag. The default value is 0.

This argument applies only to Abaqus/Explicit analyses.

n
An Int specifying the number of solution-dependent state variables required at each integration point. The default value is 0.

Return value

A Depvar object.

Exceptions

RangeError.

Members

The Depvar object has members with the same names and descriptions as the arguments to the *Depvar* method.

Corresponding analysis keywords

- **DEPVAR*

DetonationPoint object

A DetonationPoint object specifies a suboption of the Eos object. The DetonationPoint object defines either isotropic linear elastic shear or linear viscous shear behavior for a hydrodynamic material.

Access

```
materialApi.materials()[name].eos().detonationPoint()
```

DetonationPoint(...)

This method creates a DetonationPoint object.

Path

```
materialApi.materials()[name].eos().DetonationPoint
```

Prototype

```
odb_DetonationPoint&
DetonationPoint(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- X value for coordinate of detonation point.
- Y value for coordinate of detonation point.
- Z value for coordinate of detonation point.
- Detonation delay time.

Return value

A DetonationPoint object.

Exceptions

None.

Members

The DetonationPoint object has members with the same names and descriptions as the arguments to the *DetonationPoint* method.

Corresponding analysis keywords

- **DETONATION POINT*

Dielectric object

The Dielectric object specifies dielectric material properties.

Access

```
materialApi.materials()[name].dielectric()
```

Dielectric(...)

This method creates a Dielectric object.

Path

```
materialApi.materials()[name].Dielectric
```

Prototype

```
odb_Dielectric&
Dielectric(const odb_SequenceSequenceDouble& table,
           const odb_String& type,
           bool frequencyDependency,
           bool temperatureDependency,
           int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the dielectric behavior. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".

frequencyDependency
A Boolean specifying whether the data depend on frequency. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Dielectric constant.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{11}^{φ}.
- D_{22}^{φ}.
- D_{33}^{φ}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{11}^{φ}.
- D_{12}^{φ}.
- D_{22}^{φ}.
- D_{13}^{φ}.
- D_{23}^{φ}.
- D_{33}^{φ}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Dielectric object.

Exceptions

None.

Members

The Dielectric object has members with the same names and descriptions as the arguments to the *Dielectric* method.

Corresponding analysis keywords

- **DIELECTRIC*

Diffusivity object

The Diffusivity object specifies mass diffusivity.

Access

```
materialApi.materials()[name].diffusivity()
```

Diffusivity(...)

This method creates a Diffusivity object.

Path

```
materialApi.materials()[name].Diffusivity
```

Prototype

```
odb_Diffusivity&
Diffusivity(const odb_SequenceSequenceDouble& table,
            const odb_String& type,
            const odb_String& law,
            bool temperatureDependency,
            int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the type of diffusivity. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".

law
An odb_String specifying the diffusion behavior. Possible values are "GENERAL" and "FICK". The default value is "GENERAL".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Diffusivity, $\boldsymbol{D}$.
- Concentration, $\boldsymbol{c}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{11}.
- D_{22}.
- D_{33}.
- Concentration, c.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{11}.
- D_{12}.
- D_{22}.
- D_{13}.
- D_{23}.
- D_{33}.
- Concentration, c.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Diffusivity object.

Exceptions

RangeError.

Members

The Diffusivity object has members with the same names and descriptions as the arguments to the *Diffusivity* method.

In addition, the Diffusivity object can have the following members:

Prototype

```
odb_PressureEffect pressureEffect() const;
odb_SoretEffect soretEffect() const;
```

pressureEffect	A *PressureEffect* object.
soretEffect	A *SoretEffect* object.

Corresponding analysis keywords

- **DIFFUSIVITY*

DruckerPrager object

The DruckerPrager object specifies the extended Drucker-Prager plasticity model.

Access

```
materialApi.materials()[name].druckerPrager()
```

DruckerPrager(...)

This method creates a DruckerPrager object.

Path

```
materialApi.materials()[name].DruckerPrager
```

Prototype

```
odb_DruckerPrager&
DruckerPrager(const odb_SequenceSequenceDouble& table,
              const odb_String& shearCriterion,
              double eccentricity,
              bool testData,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

shearCriterion
An odb_String specifying the yield criterion. Possible values are "LINEAR", "HYPERBOLIC", and "EXPONENTIAL". The default value is "LINEAR".

This argument applies only to Abaqus/Standard analyses. Only the linear Drucker-Prager model is available in Abaqus/Explicit analyses.

eccentricity
A Double specifying the flow potential eccentricity, ϵ, a small positive number that defines the rate at which the hyperbolic flow potential approaches its asymptote. The default value is 0.1.

This argument applies only to Abaqus/Standard analyses.

testData
A Boolean specifying whether the material constants for the exponent model are to be computed by Abaqus/Standard from triaxial test data at different levels of confining pressure. The default value is false.

This argument is valid only if *shearCriterion*="EXPONENTIAL".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *shearCriterion*=LINEAR (the only option allowed in an Abaqus/Explicit analysis), the table data specify the following:

- Material angle of friction, β, in the p–t plane. Give the value in degrees.
- K, the ratio of the flow stress in triaxial tension to the flow stress in triaxial compression. $0.778 \le K \le 1.0$. If the default value of 0.0 is accepted, a value of 1.0 is assumed.
- Dilation angle, ψ, in the p–t plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shearCriterion*=HYPERBOLIC, the table data specify the following:

- Material angle of friction, β, at high confining pressure in the p–q plane. Give the value in degrees.
- Initial hydrostatic tension strength, $p_t|_0$.
- Dilation angle, ψ, at high confining pressure in the p–q plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shearCriterion*=EXPONENTIAL, the table data specify the following:

- Dilation angle, ψ, at high confining pressure in the p–q plane. Give the value in degrees.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DruckerPrager object.

Exceptions

RangeError.

Members

The DruckerPrager object has members with the same names and descriptions as the arguments to the *DruckerPrager* method.

In addition, the DruckerPrager object can have the following members:

Prototype

```
odb_DruckerPragerCreep druckerPragerCreep() const;
odb_DruckerPragerHardening druckerPragerHardening() const;
odb_RateDependent rateDependent() const;
odb_TriaxialTestData triaxialTestData() const;
```

druckerPragerCreep
A *DruckerPragerCreep* object.

druckerPragerHardening
A *DruckerPragerHardening* object.

rateDependent
A *RateDependent* object.

triaxialTestData
A *TriaxialTestData* object.

Corresponding analysis keywords

- **DRUCKER PRAGER*

DruckerPragerCreep object

The DruckerPragerCreep object specifies creep for Drucker-Prager plasticity models.

Access

```
materialApi.materials()[name].druckerPrager().druckerPragerCreep()
```

DruckerPragerCreep(...)

This method creates a DruckerPragerCreep object.

Path

```
materialApi.materials()[name].druckerPrager().DruckerPragerCreep
```

Prototype

```
odb_DruckerPragerCreep&
DruckerPragerCreep(const odb_SequenceSequenceDouble& table,
                   const odb_String& law,
                   bool temperatureDependency,
                   int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law
An odb_String specifying the type of data defining the creep law. Possible values are:

- "STRAIN", specifying a strain-hardening power law.
- "TIME", specifying a time-hardening power law.
- "SINGHM", specifying a Singh-Mitchell type law.
- "USER", specifying the creep law is input from user subroutine *CREEP*.

The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=TIME or *law*=STRAIN, the table data specify the following:

- A. (Units of $F^{-n}L^{2n}T^{-1-m}$.)
- n.
- m.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *law*=SINGHM, the table data specify the following:

- A. (Units of T^{-1}.)
- α. (Units of $F^{-1}L^2$.)
- m.
- t_1. (Units of T.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A DruckerPragerCreep object.

Exceptions

RangeError.

Members

The DruckerPragerCreep object has members with the same names and descriptions as the arguments to the *DruckerPragerCreep* method.

Corresponding analysis keywords

- **DRUCKER PRAGER CREEP*

DruckerPragerHardening object

The DruckerPragerHardening object specifies hardening for Drucker-Prager plasticity models.

Access

```
materialApi.materials()[name].druckerPrager().druckerPragerHardening()
```

DruckerPragerHardening(...)

This method creates a DruckerPragerHardening object.

Path

```
materialApi.materials()[name].druckerPrager().DruckerPragerHardening
```

Prototype

```
odb_DruckerPragerHardening&
DruckerPragerHardening(const odb_SequenceSequenceDouble& table,
                       const odb_String& type,
                       bool rate,
                       bool temperatureDependency,
                       int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the type of data defining the hardening behavior. Possible values are "COMPRESSION", "TENSION", and "SHEAR". The default value is "COMPRESSION".

rate
A Boolean specifying whether the data depend on rate. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Yield stress.
- Absolute value of the corresponding plastic strain. (The first tabular value entered must always be zero.)
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$, for which this hardening curve applies.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

Return value

A DruckerPragerHardening object.

Exceptions

RangeError.

Members

The DruckerPragerHardening object has members with the same names and descriptions as the arguments to the *DruckerPragerHardening* method.

Corresponding analysis keywords

- **DRUCKER PRAGER HARDENING*

Elastic object

The Elastic object specifies elastic material properties.

Access

```
materialApi.materials()[name].elastic()
```

Elastic(...)

This method creates an Elastic object.

Path

```
materialApi.materials()[name].Elastic
```

Prototype

```
odb_Elastic&
Elastic(const odb_SequenceSequenceDouble& table,
        const odb_String& type,
        bool noCompression,
        bool noTension,
        bool temperatureDependency,
        int dependencies,
        const odb_String& moduli);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type

An odb_String specifying the type of elasticity data provided. Possible values are:

- "ISOTROPIC"
- "ORTHOTROPIC"
- "ANISOTROPIC"
- "ENGINEERING_CONSTANTS"
- "LAMINA"
- "TRACTION"
- "COUPLED_TRACTION"
- "SHORT_FIBER"
- "SHEAR"
- "BILAMINA"

The default value is "ISOTROPIC".

noCompression

A Boolean specifying whether compressive stress is allowed. The default value is false.

noTension A Boolean specifying whether tensile stress is allowed. The default value is false.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

moduli An odb_String specifying the time-dependence of the elastic material constants. Possible values are "INSTANTANEOUS" and "LONG_TERM". The default value is "LONG_TERM".

Table data

If *type*=ISOTROPIC, the table data specify the following:

- The Young's modulus, E.
- The Poisson's ratio, ν.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=SHEAR, the table data specify the following:

- The shear modulus,G.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ENGINEERING_CONSTANTS, the table data specify the following:

- E_1.
- E_2.
- E_3.
- ν_{12}.
- ν_{13}.
- ν_{23}.
- G_{12}.
- G_{13}.
- G_{23}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *type*=LAMINA, the table data specify the following:

- E_1.
- E_2.
- ν_{12}.
- G_{12}.
- G_{13}. This shear modulus is needed to define transverse shear behavior in shells.
- G_{23}. This shear modulus is needed to define transverse shear behavior in shells.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- D_{1111}.
- D_{1122}.
- D_{2222}.
- D_{1133}.
- D_{2233}.
- D_{3333}.
- D_{1212}.
- D_{1313}.
- D_{2323}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- D_{1111}.
- D_{1122}.
- D_{2222}.
- D_{1133}.
- D_{2233}.
- D_{3333}.

- D_{1112}.
- D_{2212}.
- D_{3312}.
- D_{1212}.
- D_{1113}.
- D_{2213}.
- D_{3313}.
- D_{1213}.
- D_{1313}.
- D_{1123}.
- D_{2223}.
- D_{3323}.
- D_{1223}.
- D_{1323}.
- D_{2323}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=TRACTION, the table data specify the following:

- E for warping elements; E_{nn} for cohesive elements.
- G_1 for warping elements; E_{ss} for cohesive elements.
- G_2 for warping elements; E_{tt} for cohesive elements.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=COUPLED_TRACTION, the table data specify the following:

- E_{nn}.
- E_{ss}.
- E_{tt}.
- E_{ns}.
- E_{nt}.

- E_{st}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=BILAMINA, the table data specify the following:

- E_{1+}.
- E_{2+}.
- ν_{12+}.
- G_{12}.
- E_{1-}.
- E_{2-}.
- ν_{12-}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=SHORT_FIBER, there is no table data.

Return value

An Elastic object.

Exceptions

RangeError.

Members

The Elastic object has members with the same names and descriptions as the arguments to the *Elastic* method.

In addition, the Elastic object can have the following members:

Prototype

```
odb_FailStress failStress() const;
odb_FailStrain failStrain() const;
```

failStress A *FailStress* object.

failStrain A *FailStrain* object.

Corresponding analysis keywords

- **ELASTIC*

ElectricalConductivity object

The ElectricalConductivity object specifies electrical conductivity.

Access

```
materialApi.materials()[name].electricalConductivity()
```

ElectricalConductivity(...)

This method creates an ElectricalConductivity object.

Path

```
materialApi.materials()[name].ElectricalConductivity
```

Prototype

```
odb_ElectricalConductivity&
ElectricalConductivity(const odb_SequenceSequenceDouble& table,
                       const odb_String& type,
                       bool frequencyDependency,
                       bool temperatureDependency,
                       int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the type of electrical conductivity. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".

frequencyDependency
A Boolean specifying whether the data depend on frequency. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Electrical conductivity.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- σ_{11}^E.
- σ_{22}^E.
- σ_{33}^E.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- σ_{11}^E.
- σ_{12}^E.
- σ_{22}^E.
- σ_{13}^E.
- σ_{23}^E.
- σ_{33}^E.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

An ElectricalConductivity object.

Exceptions

RangeError.

Members

The ElectricalConductivity object has members with the same names and descriptions as the arguments to the *ElectricalConductivity* method.

Corresponding analysis keywords

- **ELECTRICAL CONDUCTIVITY*

Eos object

The Eos object specifies an equation of state model.

Access

```
materialApi.materials()[name].eos()
```

Eos(...)

This method creates an Eos object.

Path

```
materialApi.materials()[name].Eos
```

Prototype

```
odb_Eos&
Eos(const odb_String& type,
    bool temperatureDependency,
    int dependencies,
    double detonationEnergy,
    const odb_SequenceSequenceDouble& solidTable,
    const odb_SequenceSequenceDouble& gasTable,
    const odb_SequenceSequenceDouble& reactionTable,
    const odb_SequenceSequenceDouble& gasSpecificTable,
    const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

type

An odb_String specifying the equation of state. Possible values are "USUP", "JWL", "IDEALGAS", "TABULAR", and "IGNITIONANDGROWTH". The default value is "IDEALGAS".

temperatureDependency

A Boolean specifying whether the data in *gasSpecificTable* depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies for the data in *gasSpecificTable*. The default value is 0.

detonationEnergy

A Double specifying the detonation energy text field. The default value is 0.0.

solidTable

An odb_SequenceSequenceDouble specifying the following:

- A_{s}.
- B_{s}.
- ${\omega}_{s}$.

- R_{1s}.
- R_{2s}.

The default value is an empty sequence.

gasTable

An odb_SequenceSequenceDouble specifying the following:

- A_{g}.
- B_{g}.
- ${\omega}_{g}$.
- R_{1g}.
- R_{2g}.

The default value is an empty sequence.

reactionTable

An odb_SequenceSequenceDouble specifying the following:

- Initial Pressure, I.
- Product co-volume, a.
- Exponent on the unreacted fraction (ignition term), x.
- First burn rate coefficient, G_{1}
- Exponent on the unreacted fraction (growth term), c.
- Exponent on the reacted fraction (growth term), d.
- Pressure exponent (growth term), y.
- Second burn rate coefficient, G_{2}.
- Exponent on the unreacted fraction (completion term), e.
- Exponent on the reacted fraction (completion term), g.
- Pressure exponent (completion term), z.
- Initial reacted fraction, ${F^{max}}_{ig}$.
- Maximum reacted fraction for the growth term, ${F^{max}}_{G1}$.
- Minimum reacted fraction, ${F^{min}}_{G2}$.

The default value is an empty sequence.

gasSpecificTable

An odb_SequenceSequenceDouble specifying the following:

- Specific Heat per unit mass.
- Temperature dependent data.
- Value of first field variable.
- Value of second field variable.
- Etc.

The default value is an empty sequence.

table

An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

Table data

If *type*=IDEALGAS, the table data represents the following:

- Gas constant, R.
- The ambient pressure, p_A. If this field is left blank, a default of 0.0 is used.

If *type*=JWL, the table data represents the following:

- Detonation wave speed, C_d.
- A.
- B.
- ω. (Dimensionless.)
- R_1. (Dimensionless.)
- R_2. (Dimensionless.)
- Pre-detonation bulk modulus, K_{pd}.
- Detonation energy density, E_0.

If *type*=USUP, the table data represents the following:

- c_0.
- s. (Dimensionless.)
- Γ_0. (Dimensionless.)

If *type*=TABULAR, the table data represents the following:

- F_1.
- F_2.
- ε^c_{vol}. (Dimensionless.)

Return value

An Eos object.

Exceptions

None.

Members

The Eos object has members with the same names and descriptions as the arguments to the *Eos* method.

In addition, the Eos object can have the following members:

Prototype

```
odb_DetonationPoint detonationPoint() const;
odb_EosCompaction eosCompaction() const;
```

detonationPoint	A *DetonationPoint* object.
eosCompaction	An *EosCompaction* object.
tensileFailure	A *tensileFailure* object.

Corresponding analysis keywords

- **EOS*

EosCompaction object

The EosCompaction object specifies material eos compaction.

Access

```
materialApi.materials()[name].eos().eosCompaction()
```

EosCompaction(...)

This method creates a EosCompaction object.

Path

```
materialApi.materials()[name].eos().EosCompaction
```

Prototype

```
odb_EosCompaction&
EosCompaction(double soundSpeed,
              double porosity,
              double pressure,
              double compactionPressure);
```

Required arguments

soundSpeed	A Double specifying reference sound speed in the porous material.
porosity	A Double specifying value of the porosity of the unloaded material.
pressure	A Double specifying pressure required to initialize plastic behavior.
compactionPressure	A Double specifying compaction pressure at which all pores are crushed.

Optional arguments

None.

Return value

An EosCompaction object.

Exceptions

RangeError.

Members

The EosCompaction object has members with the same names and descriptions as the arguments to the *EosCompaction* method.

Corresponding analysis keywords

- **EOS COMPACTION*

Expansion object

The Expansion object specifies thermal expansion.

Access

```
materialApi.materials()[name].expansion()
```

Expansion(...)

This method creates an Expansion object.

Path

```
materialApi.materials()[name].Expansion
```

Prototype

```
odb_Expansion&
Expansion(const odb_String& type,
          bool userSubroutine,
          double zero,
          bool temperatureDependency,
          int dependencies,
          const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

type

An odb_String specifying the type of expansion. Possible values are "ISOTROPIC", "ORTHOTROPIC", "ANISOTROPIC", and "SHORT_FIBER". The default value is "ISOTROPIC".

userSubroutine

A Boolean specifying whether a user subroutine is used to define the increments of thermal strain. The default value is false.

zero

A Double specifying the value of θ^0 if the thermal expansion is temperature-dependent or field-variable-dependent. The default value is 0.0.

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

table

An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

This argument is required only if *type* is not "USER".

Table data

If *type*=ISOTROPIC, the table data specify the following:

- α in Abaqus/Standard or Abaqus/Explicit analyses.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- α_{11}.
- α_{22}.
- α_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- α_{11}.
- α_{22}.
- α_{33}. (Not used for plane stress case.)
- α_{12}.
- α_{13}.
- α_{23}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=SHORT_FIBER, there is no table data.

Return value

An Expansion object.

Exceptions

RangeError.

Members

The Expansion object has members with the same names and descriptions as the arguments to the *Expansion* method.

Corresponding analysis keywords

- **EXPANSION*

FailStrain object

The FailStrain object defines parameters for strain-based failure measures.

Access

```
materialApi.materials()[name].elastic().failStrain()
```

FailStrain(...)

This method creates a FailStrain object.

Path

```
materialApi.materials()[name].elastic().FailStrain
```

Prototype

```
odb_FailStrain&
FailStrain(const odb_SequenceSequenceDouble& table,
           bool temperatureDependency,
           int dependencies);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tensile strain limit in fiber direction, $X_{\varepsilon t}$.
- Compressive strain limit in fiber direction, $X_{\varepsilon c}$.
- Tensile strain limit in transverse direction, $Y_{\varepsilon t}$.
- Compressive strain limit in transverse direction, $Y_{\varepsilon c}$.
- Shear strain limit in the X–Y plane, S_{ε}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FailStrain object.

Exceptions

RangeError.

Members

The FailStrain object has members with the same names and descriptions as the arguments to the *FailStrain* method.

Corresponding analysis keywords

- **FAIL STRAIN*

FailStress object

The FailStress object defines parameters for stress-based failure measures.

Access

```
materialApi.materials()[name].elastic().failStress()
```

FailStress(...)

This method creates a FailStress object.

Path

```
materialApi.materials()[name].elastic().FailStress
```

Prototype

```
odb_FailStress&
FailStress(const odb_SequenceSequenceDouble& table,
           bool temperatureDependency,
           int dependencies);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tensile stress limit in fiber direction, X_t.
- Compressive stress limit in fiber direction, X_c.
- Tensile stress limit in transverse direction, Y_t.
- Compressive stress limit in transverse direction, Y_c.
- Shear strength in the X–Y plane, S.
- Cross product term coefficient, $\overset{*}{f}$ ($-1.0 \leq \overset{*}{f} \leq 1.0$). The default value is zero.
- Biaxial stress limit, σ_{biax}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

Return value

A FailStress object.

Exceptions

RangeError.

Members

The FailStress object has members with the same names and descriptions as the arguments to the *FailStress* method.

Corresponding analysis keywords

- **FAIL STRESS*

FailureRatios object

The FailureRatios object specifies the shape of the failure surface for a *Concrete* model.

Access

```
materialApi.materials()[name].concrete().failureRatios()
```

FailureRatios(...)

This method creates a FailureRatios object.

Path

```
materialApi.materials()[name].concrete().FailureRatios
```

Prototype

```
odb_FailureRatios&
FailureRatios(const odb_SequenceSequenceDouble& table,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Ratio of the ultimate biaxial compressive stress to the uniaxial compressive ultimate stress. The default value is 1.16.
- Absolute value of the ratio of the uniaxial tensile stress at failure to the uniaxial compressive stress at failure. The default value is 0.09.
- Ratio of the magnitude of a principal component of plastic strain at ultimate stress in biaxial compression to the plastic strain at ultimate stress in uniaxial compression. The default value is 1.28.
- Ratio of the tensile principal stress value at shear in plane stress, when the other nonzero principal stress component is at the ultimate compressive stress value, to the tensile cracking stress under uniaxial tension. The default value is 1/3.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

Return value

A FailureRatios object.

Exceptions

RangeError.

Members

The FailureRatios object has members with the same names and descriptions as the arguments to the *FailureRatios* method.

Corresponding analysis keywords

- **FAILURE RATIOS*

FluidLeakoff object

The FluidLeakoff object specifies leak-off coefficients for pore pressure cohesive elements.

Access

```
materialApi.materials()[name].fluidLeakoff()
```

FluidLeakoff(...)

This method creates a FluidLeakoff object.

Path

```
materialApi.materials()[name].FluidLeakoff
```

Prototype

```
odb_FluidLeakoff&
FluidLeakoff(bool temperatureDependency,
             int dependencies,
             const odb_String& type,
             const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

type An odb_String specifying the type of fluid leak-off. Possible values are "COEFFICIENTS" and "USER". The default value is "COEFFICIENTS".

table An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

Table data

The table data specify the following:

- Fluid leak-off coefficient at top element surface.
- Fluid leak-off coefficient at bottom element surface.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A FluidLeakoff object.

Exceptions

None.

Members

The FluidLeakoff object has members with the same names and descriptions as the arguments to the *FluidLeakoff* method.

Corresponding analysis keywords

- **FLUID LEAKOFF*

GapConductance object

The GapConductance object specifies conductive heat transfer between closely adjacent (or contacting) surfaces.

Access

```
materialApi.materials()[name].gapConductance()
```

GapConductance(...)

This method creates a GapConductance object.

Path

```
materialApi.materials()[name].GapConductance
```

Prototype

```
odb_GapConductance&
GapConductance(const odb_SequenceSequenceDouble& table,
        bool pressureDependency,
        int dependencies);
```

Optional arguments

pressureDependency	A Boolean specifying whether the data depend on pressure. The default value is false.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.
table	An odb_SequenceSequenceDouble specifying the items described below.

Table data

- Gap Conductance or Cohesive Separation.
- Gap Clearance, Gap Pressure (if the optional parameter pressureDependency is used), or Closure, c (for coupled temperature-displacement gasket elements).
- Average Temperature if the data depend on temperature.
- Mass Flow Rate per unit area if the data depend on the average mass flow rate.
- Value of the first field variable if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapConductance object.

Exceptions

None.

Members

The GapConductance object has members with the same names and descriptions as the arguments to the *GapConductance* method.

Corresponding analysis keywords

- **GAP CONDUCTANCE*

GapConvection object

The GapConvection object specifies the Nusselt number (**Nu**) to calculate the convective coefficient for heat transfer between the gap flow and both the top and bottom surfaces of a coupled temperature-pore pressure cohesive element.

Access

```
materialApi.materials()[name].gapConvection()
```

GapConvection(...)

This method creates a GapConvection object.

Path

```
materialApi.materials()[name].Gapconvection
```

Prototype

```
odb_GapConvection&
GapConvection(const odb_SequenceSequenceDouble& table,
        bool temperatureDependency,
        int dependencies,
        const odb_String& type);
```

Required arguments

type
An odb_String specifying the type of gap convection. Possible values are FLUX, TEMPERATURE, and TABULAR. The default value is FLUX.

Optional arguments

If *type*=TABULAR the following optional arguments can be used:

table
An odb_SequenceSequenceDouble specifying the items described below.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=TABULAR the table data specify the following:

- Nusselt number (**Nu**)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapConvection object.

Exceptions

None.

Members

The GapConvection object has members with the same names and descriptions as the arguments to the *GapConvection* method.

Corresponding analysis keywords

- **GAP CONVECTION*

GapFlow object

The GapFlow object specifies tangential flow constitutive parameters for pore pressure cohesive elements.

Access

```
materialApi.materials()[name].gapFlow()
```

GapFlow(...)

This method creates a GapFlow object.

Path

```
materialApi.materials()[name].GapFlow
```

Prototype

```
odb_GapFlow&
GapFlow(const odb_SequenceSequenceDouble& table,
        odb_Union kmax,
        bool temperatureDependency,
        int dependencies,
        const odb_String& type);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

kmax
The string "NONE" or a Double specifying the maximum permeability value that should be used. If *kmax*="NONE", Abaqus assumes that the permeability is not bounded. This value is meaningful only when *type*="NEWTONIAN". The default value is "NONE".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

type
An odb_String specifying the type of gap flow. Possible values are "NEWTONIAN", "POWER_LAW", "BINGHAM_PLASTIC", and "HERSCHEL-BULKLEY". The default value is "NEWTONIAN".

Table data

If *type*=NEWTONIAN the table data specify the following:

- Pore viscosity.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *type*=POWER_LAW the table data specify the following:

- Consistency.
- Exponent.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=BINGHAM_PLASTIC the table data specify the following:

- Consistency.
- Yield stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=HERSCHEL-BULKLEY the table data specify the following:

- Consistency.
- Exponent.
- Yield stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GapFlow object.

Exceptions

None.

Members

The GapFlow object has members with the same names and descriptions as the arguments to the *GapFlow* method.

Corresponding analysis keywords

- **GAP FLOW*

GapRadiation object

The GapRadiation object specifies radiative heat transfer between closely adjacent surfaces.

Access

```
materialApi.materials()[name].gapRadiation()
```

GapRadiation(...)

This method creates a GapConductance object.

Path

```
materialApi.materials()[name].GapConductance
```

Prototype

```
odb_GapConductance&
GapConductance(const odb_SequenceSequenceDouble& table,
Float mainSurfaceEmissivity, Float secondarySurfaceEmissivity, );
```

Required arguments

mainSurfaceEmissivity	A Float specifying the Emissivity of main surface.
secondarySurfaceEmissivity	A Float specifying the Emissivity of the secondary surface.
table	An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

Table data

The table data specify the following:

- Effective View Factor.
- Gap Clearance.
- Repeat this data line as often as necessary to define the dependence of the view factor on gap clearance.

Return value

A GapRadiation object.

Exceptions

None.

Members

The GapRadiation object has members with the same names and descriptions as the arguments to the *GapRadiation* method.

Corresponding analysis keywords

- **GAP RADIATION*

GasketMembraneElastic object

The GasketMembraneElastic object defines the elastic parameters for the membrane shear behavior of a gasket.

Access

```
materialApi.materials()[name].gasketMembraneElastic()
```

GasketMembraneElastic(...)

This method creates a GasketMembraneElastic object.

Path

```
materialApi.materials()[name].GasketMembraneElastic
```

Prototype

```
odb_GasketMembraneElastic&
GasketMembraneElastic(const odb_SequenceSequenceDouble& table,
                      bool temperatureDependency,
                      int dependencies);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Young's modulus, E.
- Poisson's ratio, ν.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketMembraneElastic object.

Exceptions

RangeError.

Members

The GasketMembraneElastic object has members with the same names and descriptions as the arguments to the *GasketMembraneElastic* method.

Corresponding analysis keywords

- **GASKET ELASTICITY*

GasketThicknessBehavior object

The GasketThicknessBehavior object defines the behavior in the thickness direction for a gasket.

Access

```
materialApi.materials()[name].gasketThicknessBehavior()
```

GasketThicknessBehavior(...)

This method creates a GasketThicknessBehavior object.

Path

```
materialApi.materials()[name].GasketThicknessBehavior
```

Prototype

```
odb_GasketThicknessBehavior&
GasketThicknessBehavior(const odb_SequenceSequenceDouble& table,
            bool temperatureDependency,
            int dependencies,
            double tensileStiffnessFactor,
            const odb_String& type,
            int unloadingDependencies,
            bool unloadingTemperatureDependency,
            const odb_String& variableUnits,
            double yieldOnset,
            const odb_String& yieldOnsetMethod,
            const odb_SequenceSequenceDouble& unloadingTable);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying loading data. The first sequence must contain only 0. At least two sequences must be specified if *type*="DAMAGE", and at least 3 sequences must be specified if *type*="ELASTIC_PLASTIC". The items in the table data are described below.

Optional arguments

temperatureDependency

A Boolean specifying whether the loading data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies included in the definition of the loading data, in addition to temperature. The default value is 0.

tensileStiffnessFactor

A Double specifying the fraction of the initial compressive stiffness that defines the stiffness in tension. The default value is 10^{-3}.

type

An odb_String specifying a damage elasticity model or an elastic-plastic model for gasket thickness-direction behavior. Possible values are "ELASTIC_PLASTIC" and "DAMAGE". The default value is "ELASTIC_PLASTIC".

unloadingDependencies
An Int specifying the number of field variable dependencies included in the definition of the unloading data, in addition to temperature. The default value is 0.

unloadingTemperatureDependency
A Boolean specifying whether unloading data depends on temperature. The default value is false.

variableUnits
An odb_String specifying the behavior in terms of units of force (or force in unit length) versus closure or pressure versus closure. Possible values are "STRESS" and "FORCE". The default value is "STRESS".

yieldOnset
A Double specifying the closure value at which the onset of yield occurs or the relative drop in slope on the loading curve that defines the onset of plastic deformation (depending on the value of *yieldOnsetMethod*). The default value is 0.1.

yieldOnsetMethod
An odb_String specifying the method used to determine yield onset. Possible values are "RELATIVE_SLOPE_DROP" and "CLOSURE_VALUE". The default value is "RELATIVE_SLOPE_DROP".

unloadingTable
An odb_SequenceSequenceDouble specifying unloading data. The items in the table data are described below. The default value is an empty sequence.

Table data

If *variableUnits*=STRESS, the loading table data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE, the loading table data specify the following:

- Force or force per unit length; this value must be positive.
- Closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=STRESS and *type*=ELASTIC_PLASTIC, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Plastic closure; this value must be positive.

- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE and *type*=ELASTIC_PLASTIC, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Plastic closure; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=STRESS and *type*=DAMAGE, the *unloadingTable* data specify the following:

- Pressure; this value must be positive.
- Closure; this value must be positive.
- Maximum closure reached while loading the gasket; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *variableUnits*=FORCE and *type*=DAMAGE, the *unloadingTable* data specify the following:

- Force or force per unit length; this value must be positive.
- Closure; this value must be positive.
- Maximum closure reached while loading the gasket; this value must be positive.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketThicknessBehavior object.

Exceptions

RangeError.

Members

The GasketThicknessBehavior object has members with the same names and descriptions as the arguments to the *GasketThicknessBehavior* method.

In addition, the GasketThicknessBehavior object can have the following member:

Prototype

```
odb_ContactArea contactArea() const;
```

contactArea

A *ContactArea* object.

Corresponding analysis keywords

- **GASKET THICKNESS BEHAVIOR*

GasketTransverseShearElastic object

The GasketTransverseShearElastic object defines the elastic parameters for the transverse shear behavior of a gasket.

Access

```
materialApi.materials()[name].gasketTransverseShearElastic()
```

GasketTransverseShearElastic(...)

This method creates a GasketTransverseShearElastic object.

Path

```
materialApi.materials()[name].GasketTransverseShearElastic
```

Prototype

```
odb_GasketTransverseShearElastic&
GasketTransverseShearElastic(const odb_SequenceSequenceDouble& table,
                             const odb_String& variableUnits,
                             bool temperatureDependency,
                             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

variableUnits
An odb_String specifying the unit system in which the transverse shear behavior will be defined. Possible values are "STRESS" and "FORCE". The default value is "STRESS".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Shear stiffness. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A GasketTransverseShearElastic object.

Exceptions

RangeError.

Members

The GasketTransverseShearElastic object has members with the same names and descriptions as the arguments to the *GasketTransverseShearElastic* method.

Corresponding analysis keywords

- **GASKET ELASTICITY*

Gel object

The Gel object defines a swelling gel.

Access

```
materialApi.materials()[name].gel()
```

Gel(...)

This method creates a Gel object.

Path

```
materialApi.materials()[name].Gel
```

Prototype

```
odb_Gel&
Gel(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- Radius of gel particles when completely dry, r_a^{dry}.
- Fully swollen radius of gel particles, r_a^f.
- Number of gel particles per unit volume, k_a.
- Relaxation time constant for long-term swelling of gel particles, τ_1.

Return value

A Gel object.

Exceptions

None.

Members

The Gel object has members with the same names and descriptions as the arguments to the *Gel* method.

Corresponding analysis keywords

- **GEL*

HeatGeneration object

The HeatGeneration object includes volumetric heat generation in heat transfer analyses.

Access

```
materialApi.materials()[name].heatGeneration()
```

HeatGeneration()

This method creates a HeatGeneration object.

Path

```
materialApi.materials()[name].HeatGeneration
```

Prototype

```
odb_HeatGeneration&
HeatGeneration();
```

Arguments

None.

Return value

A HeatGeneration object.

Exceptions

None.

Members

The HeatGeneration object has no members.

Corresponding analysis keywords

- **HEAT GENERATION*

Hyperelastic object

The Hyperelastic object specifies elastic properties for approximately incompressible elastomers.

Access

```
materialApi.materials()[name].hyperelastic()
```

Hyperelastic(...)

This method creates a Hyperelastic object.

Path

```
materialApi.materials()[name].Hyperelastic
```

Prototype

```
odb_Hyperelastic&
Hyperelastic(const odb_SequenceSequenceDouble& table,
             const odb_String& type,
             const odb_String& moduliTimeScale,
             bool temperatureDependency,
             int n,
             odb_Union beta,
             bool testData,
             bool compressible,
             int properties,
             const odb_String& deviatoricResponse,
             const odb_String& volumetricResponse,
             double poissonRatio,
             const odb_String& materialType,
             const odb_String& anisotropicType,
             const odb_String& formulation,
             const odb_String& behaviorType,
             int dependencies,
             int localDirections);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying the items described below. This argument is valid only if *testData*=OFF.

Optional arguments

type

An odb_String specifying the type of strain energy potential. Possible values are:

- "ARRUDA_BOYCE"
- "MARLOW"
- "MOONEY_RIVLIN"
- "NEO_HOOKE"
- "OGDEN"

- "POLYNOMIAL"
- "REDUCED_POLYNOMIAL"
- "USER"
- "VAN_DER_WAALS"
- "YEOH"
- "UNKNOWN"

The default value is "UNKNOWN".

moduliTimeScale
An odb_String specifying the nature of the time response. Possible values are "INSTANTANEOUS" and "LONG_TERM". The default value is "LONG_TERM".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

n
An Int specifying the order of the strain energy potential. The default value is 1.

If *testData*=ON and *type*="POLYNOMIAL", *n* can take only the values 1 or 2.

If *testData*=ON and *type*="OGDEN" or if *testData*=OFF for either type, $1 \leq n \leq 6$.

If *type*="USER", this argument cannot be used.

beta
The string "FITTED_VALUE" or a Double specifying the invariant mixture parameter. This argument is valid only if *testData*=ON and *type*="VAN_DER_WAALS". The default value is "FITTED_VALUE".

testData
A Boolean specifying whether test data are supplied. The default value is true.

compressible
A Boolean specifying whether the hyperelastic material is compressible. This parameter is applicable only to user-defined hyperelastic materials. The default value is false.

properties
An Int specifying the number of property values needed as data for the user-defined hyperelastic material. The default value is 0.

deviatoricResponse
An odb_String specifying which test data to use. Possible values are "UNIAXIAL", "BIAXIAL", and "PLANAR". The default value is "UNIAXIAL".

volumetricResponse
An odb_String specifying the volumetric response. Possible values are "DEFAULT", "VOLUMETRIC_DATA", "POISSON_RATIO", and "LATERAL_NOMINAL_STRAIN". The default value is "DEFAULT".

poissonRatio
A Double specifying the poisson ratio. This argument is valid only if *volumetricResponse*="POISSON_RATIO". The default value is 0.0.

materialType
An odb_String specifying the type of material. Possible values are "ISOTROPIC" and "ANISOTROPIC". The default value is "ISOTROPIC".

anisotropicType
An odb_String specifying the type of strain energy potential. Possible values are "FUNG_ANISOTROPIC", "FUNG_ORTHOTROPIC", "HOLZAPFEL", and "USER_DEFINED". The default value is "FUNG_ANISOTROPIC".

formulation An odb_String specifying the type of formulation. Possible values are "STRAIN" and "INVARIANT". The default value is "STRAIN".

behaviorType An odb_String specifying the type of anisotropic hyperelastic material behavior. Possible values are "INCOMPRESSIBLE" and "COMPRESSIBLE". The default value is "INCOMPRESSIBLE".

dependencies An Int specifying the number of field variable dependencies for the data in*volumetricTable* . The default value is 0.

localDirections An Int specifying the number of local directions for the data in*volumetricTable* . The default value is 0.

Table data

If *type*=ARRUDA_BOYCE, the table data specify the following:

- μ.
- λ_m.
- D.
- Temperature, if the data depend on temperature.

If *type*=MOONEY_RIVLIN, the table data specify the following:

- C_{10}.
- C_{01}.
- D_1.
- Temperature, if the data depend on temperature.

If *type*=NEO_HOOKE, the table data specify the following:

- C_{10}.
- D_1.
- Temperature, if the data depend on temperature.

If *type*=OGDEN, the table data specify the following for values of n:

- μ_i and α_i for i from 1 to n.
- n coefficients D_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=POLYNOMIAL, the table data specify the following for values of n:

- C_{ij} for each value of $(i + j)$ from 1 to n with i decreasing from $(i + j)$ to zero and j increasing from zero to $(i + j)$.
- n coefficients D_i.

- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $3 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=REDUCED_POLYNOMIAL, the table data specify the following for values of n:

- C_{i0} for i from 1 to n.
- n coefficients D_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \leq n \leq 6$ in an Abaqus/Explicit analysis.

If *type*=VAN_DER_WAALS, the table data specify the following:

- μ.
- λ_m.
- a.
- β.
- D.
- Temperature, if the data depend on temperature.

If *type*=YEOH, the table data specify the following:

- C_{10}.
- C_{20}.
- C_{30}.
- D_1.
- D_2.
- D_3.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed in an Abaqus/Explicit analysis.

The `None` object is the default value if *testData*=ON.

Return value

A Hyperelastic object.

Exceptions

InvalidNameError and RangeError.

Members

The Hyperelastic object has members with the same names and descriptions as the arguments to the *Hyperelastic* method.

In addition, the Hyperelastic object can have the following members:

Prototype

```
odb_BiaxialTestData biaxialTestData() const;
odb_PlanarTestData planarTestData() const;
odb_UniaxialTestData uniaxialTestData() const;
odb_VolumetricTestData volumetricTestData() const;
odb_Hysteresis hysteresis() const;
```

biaxialTestData
A *BiaxialTestData* object.

planarTestData
A *PlanarTestData* object.

uniaxialTestData
A *UniaxialTestData* object.

volumetricTestData
A *VolumetricTestData* object.

hysteresis
A *Hysteresis* object.

Corresponding analysis keywords

- **HYPERELASTIC*

Hyperfoam object

The Hyperfoam object specifies elastic properties for a hyperelastic foam.

Access

```
materialApi.materials()[name].hyperfoam()
```

Hyperfoam(...)

This method creates a Hyperfoam object.

Path

```
materialApi.materials()[name].Hyperfoam
```

Prototype

```
odb_Hyperfoam&
Hyperfoam(bool testData,
          odb_Union poisson,
          int n,
          bool temperatureDependency,
          const odb_String& moduli,
          const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

testData
A Boolean specifying whether test data are supplied. The default value is false.

poisson
The string "NONE" or a Double specifying the effective Poisson's ratio, ν, of the material. This argument is valid only when *testData*=true. The default value is "NONE".

n
An Int specifying the order of the strain energy potential. Possible values are $1 \le n \le 6$. The default value is 1.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

moduli
An odb_String specifying the time-dependence of the material constants. Possible values are "INSTANTANEOUS" and "LONG_TERM". The default value is "LONG_TERM".

table
An odb_SequenceSequenceDouble specifying the items described below. This argument is valid only when *testData*=false. The default value is an empty sequence.

Table data

The items in the table data specify the following for values of n:

- μ_i and α_i for i from 1 to n.
- ν_i.
- Temperature, if the data depend on temperature. Temperature dependence is not allowed for $4 \leq n \leq 6$ in an Abaqus/Explicit analysis.

Return value

A Hyperfoam object.

Exceptions

RangeError.

Members

The Hyperfoam object has members with the same names and descriptions as the arguments to the *Hyperfoam* method.

In addition, the Hyperfoam object can have the following members:

Prototype

```
odb_BiaxialTestData biaxialTestData() const;
odb_VolumetricTestData volumetricTestData() const;
odb_PlanarTestData planarTestData() const;
odb_SimpleShearTestData simpleShearTestData() const;
odb_UniaxialTestData uniaxialTestData() const;
```

biaxialTestData	A *BiaxialTestData* object.
volumetricTestData	A *VolumetricTestData* object.
planarTestData	A *PlanarTestData* object.
simpleShearTestData	A *SimpleShearTestData* object.
uniaxialTestData	A *UniaxialTestData* object.

Corresponding analysis keywords

- **HYPERFOAM*

Hypoelastic object

The Hypoelastic object specifies hypoelastic material properties.

Access

```
materialApi.materials()[name].hypoelastic()
```

Hypoelastic(...)

This method creates a Hypoelastic object.

Path

```
materialApi.materials()[name].Hypoelastic
```

Prototype

```
odb_Hypoelastic&
Hypoelastic(const odb_SequenceSequenceDouble& table,
            bool user);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

user
A Boolean specifying that hypoelasticity is defined by user subroutine *UHYPEL*. The default value is false.

Table data

- Instantaneous Young's modulus, E.
- Instantaneous Poisson's ratio, ν.
- First strain invariant, I_1.
- Second strain invariant, I_2.
- Third strain invariant, I_3.

Return value

A Hypoelastic object.

Exceptions

None.

Members

The Hypoelastic object has members with the same names and descriptions as the arguments to the *Hypoelastic* method.

Corresponding analysis keywords

- **HYPOELASTIC*

Hysteresis object

The Hysteresis object specifies the creep part of the material model for the hysteretic behavior of elastomers.

Access

```
materialApi.materials()[name].hyperelastic().hysteresis()
```

Hysteresis(...)

This method creates a Hysteresis object.

Path

```
materialApi.materials()[name].hyperelastic().Hysteresis
```

Prototype

```
odb_Hysteresis&
Hysteresis(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- Stress scaling factor.
- Creep parameter.
- Effective stress exponent.
- Creep strain exponent.

Return value

A Hysteresis object.

Exceptions

RangeError.

Members

The Hysteresis object has members with the same names and descriptions as the arguments to the *Hysteresis* method.

Corresponding analysis keywords

- **HYSTERESIS*

InelasticHeatFraction object

The InelasticHeatFraction object defines the fraction of the rate of inelastic dissipation that appears as a heat source.

Access

```
materialApi.materials()[name].inelasticHeatFraction()
```

InelasticHeatFraction(...)

This method creates an InelasticHeatFraction object.

Path

```
materialApi.materials()[name].InelasticHeatFraction
```

Prototype

```
odb_InelasticHeatFraction&
InelasticHeatFraction(double fraction);
```

Required arguments

None.

Optional arguments

fraction

A Double specifying the fraction of inelastic dissipation rate that appears as a heat flux per unit volume. The fraction may include a unit conversion factor if required. Possible values are 0.0 $\leq$*fraction*$\leq$ 1.0. The default value is 0.9.

Return value

An InelasticHeatFraction object.

Exceptions

RangeError.

Members

The InelasticHeatFraction object has members with the same names and descriptions as the arguments to the *InelasticHeatFraction* method.

Corresponding analysis keywords

- **INELASTIC HEAT FRACTION*

JouleHeatFraction object

The JouleHeatFraction object defines the fraction of electric energy released as heat.

Access

```
materialApi.materials()[name].jouleHeatFraction()
```

JouleHeatFraction(...)

This method creates a JouleHeatFraction object.

Path

```
materialApi.materials()[name].JouleHeatFraction
```

Prototype

```
odb_JouleHeatFraction&
JouleHeatFraction(double fraction);
```

Required arguments

None.

Optional arguments

fraction

A Double specifying the fraction of electrical energy released as heat, including any unit conversion factor. The default value is 1.0.

Return value

A JouleHeatFraction object.

Exceptions

RangeError.

Members

The JouleHeatFraction object has members with the same names and descriptions as the arguments to the *JouleHeatFraction* method.

Corresponding analysis keywords

- **JOULE HEAT FRACTION*

LatentHeat object

The LatentHeat object specifies a material's latent heat.

Access

```
materialApi.materials()[name].latentHeat()
```

LatentHeat(...)

This method creates a LatentHeat object.

Path

```
materialApi.materials()[name].LatentHeat
```

Prototype

```
odb_LatentHeat&
LatentHeat(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- Latent heat per unit mass.
- Solidus temperature.
- Liquidus temperature.

Return value

A LatentHeat object.

Exceptions

RangeError.

Members

The LatentHeat object has members with the same names and descriptions as the arguments to the *LatentHeat* method.

Corresponding analysis keywords

- **LATENT HEAT*

LowDensityFoam object

The LowDensityFoam object specifies properties for low-density foam.

Access

```
materialApi.materials()[name].lowDensityFoam()
```

LowDensityFoam(...)

This method creates a LowDensityFoam object.

Path

```
materialApi.materials()[name].LowDensityFoam
```

Prototype

```
odb_LowDensityFoam&
LowDensityFoam(bool elementRemoval,
               odb_Union maxAllowablePrincipalStress,
               bool extrapolateStressStrainCurve,
               const odb_String& strainRateType,
               double mu0,
               double mu1,
               double alpha);
```

Required arguments

None.

Optional arguments

elementRemoval

A Boolean specifying whether elements are removed if exceeding maximum principal tensile stress. This argument is valid only when *maxAllowablePrincipalStress* is defined. The default value is false.

maxAllowablePrincipalStress

The string "NONE" or a Double specifying the maximum allowable principal tensile stress. The default value is "NONE".

extrapolateStressStrainCurve

A Boolean specifying whether the stress-strain curve is extrapolated if exceeding maximum strain rate. The default value is false.

strainRateType

An odb_String specifying strain rate measure used for constitutive calculations. Possible values are "PRINCIPAL" and "VOLUMETRIC". The default value is "VOLUMETRIC".

mu0

A Double specifying the relaxation coefficient μ_0. The default value is 10^{-4}.

mu1

A Double specifying the relaxation coefficient μ_1. The default value is 0.5×10^{-2}.

alpha

A Double specifying the relaxation coefficient α. The default value is 2.0.

Return value

A LowDensityFoam object.

Exceptions

RangeError.

Members

The LowDensityFoam object has members with the same names and descriptions as the arguments to the *LowDensityFoam* method.

In addition, the LowDensityFoam object can have the following members:

Prototype

```
odb_UniaxialTestData uniaxialTensionTestData() const;
odb_UniaxialTestData uniaxialCompressionTestData() const;
```

uniaxialTensionTestData

A *UniaxialTestData* object.

uniaxialCompressionTestData

A *UniaxialTestData* object.

Corresponding analysis keywords

- **LOW DENSITY FOAM*

MagneticPermeability object

The MagneticPermeability object specifies magnetic permeability.

Access

```
materialApi.materials()[name].magneticPermeability()
```

MagneticPermeability(...)

This method creates a MagneticPermeability object.

Path

```
materialApi.materials()[name].MagneticPermeability
```

Prototype

```
odb_MagneticPermeability&
MagneticPermeability(const odb_SequenceSequenceDouble& table,
                     const odb_SequenceSequenceDouble& table2,
                     const odb_SequenceSequenceDouble& table3,
                     const odb_String& type,
                     bool frequencyDependency,
                     bool temperatureDependency,
                     int dependencies,
                     bool nonlinearBH);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below in “Table data.” If *type*=ORTHOTROPIC and nonlinearBH=ON, the data specified in the *table* is for the first direction and *table2* and *table3* must be specified.

table2
An odb_SequenceSequenceDouble specifying the items described below in “Table data” in the second direction. *table2* must be specified only if *type*=ORTHOTROPIC and nonlinearBH=ON.

table3
An odb_SequenceSequenceDouble specifying the items described below in “Table data” in the third direction. *table3* must be specified only if *type*=ORTHOTROPIC and nonlinearBH=ON.

Optional arguments

type
An odb_String specifying the type of magnetic permeability. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".

frequencyDependency
A Boolean specifying whether the data depend on frequency. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

nonlinearBH
A Boolean specifying whether the magnetic behavior is nonlinear and available in tabular form of magnetic flux density versus magnetic field values. The default value is false.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- Magnetic permeability.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ISOTROPIC, and *nonlinearBH*=TRUE, the table data specify the following:

- Magntitude of the magnetic flux density vector.
- Magnitude of the magnetic field vector.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, the table data specify the following:

- μ_{11}^{E}.
- μ_{22}^{E}.
- μ_{33}^{E}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=ORTHOTROPIC, and *nonlinearBH*=TRUE, the table data specify the following:

- Magntitude of the magnetic flux density vector in the first direction.
- Magnitude of the magnetic field vector in the second direction.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *type*=ANISOTROPIC, the table data specify the following:

- μ_{11}^{E}.
- μ_{12}^{E}.
- μ_{22}^{E}.
- μ_{13}^{E}.
- μ_{23}^{E}.
- μ_{33}^{E}.
- Frequency, if the data depend on frequency.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MagneticPermeability object.

Exceptions

RangeError.

Members

The MagneticPermeability object has members with the same names and descriptions as the arguments to the *MagneticPermeability* method.

Corresponding analysis keywords

- **MAGNETIC PERMEABILITY*

MohrCoulombHardening object

The MohrCoulombHardening object specifies hardening for the Mohr-Coulomb plasticity model.

Access

```
materialApi.materials()[name].mohrCoulombPlasticity()\
.mohrCoulombHardening()
```

MohrCoulombHardening(...)

This method creates a MohrCoulombHardening object.

Path

```
materialApi.materials()[name].mohrCoulombPlasticity()\
.MohrCoulombHardening
```

Prototype

```
odb_MohrCoulombHardening&
MohrCoulombHardening(const odb_SequenceSequenceDouble& table,
                     bool temperatureDependency,
                     int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Cohesion yield stress.
- The absolute value of the corresponding plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MohrCoulombHardening object.

Exceptions

RangeError.

Members

The MohrCoulombHardening object has members with the same names and descriptions as the arguments to the *MohrCoulombHardening* method.

Corresponding analysis keywords

- **MOHR COULOMB HARDENING*

MohrCoulombPlasticity object

The MohrCoulombPlasticity object specifies the extended Mohr-Coulomb plasticity model.

Access

```
materialApi.materials()[name].mohrCoulombPlasticity()
```

MohrCoulombPlasticity(...)

This method creates a MohrCoulombPlasticity object.

Path

```
materialApi.materials()[name].MohrCoulombPlasticity
```

Prototype

```
odb_MohrCoulombPlasticity&
MohrCoulombPlasticity(const odb_SequenceSequenceDouble& table,
                      odb_Union deviatoricEccentricity,
                      double meridionalEccentricity,
                      bool temperatureDependency,
                      int dependencies,
                      bool useTensionCutoff);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

deviatoricEccentricity
The string "NONE" or a Double specifying the flow potential eccentricity in the deviatoric plane, e; $1/2 \leq e \leq 1.0$. If *deviatoricEccentricity*="NONE", Abaqus calculates the value using the specified Mohr-Coulomb angle of friction. The default value is "NONE".

meridionalEccentricity
A Double specifying the flow potential eccentricity in the meridional plane, ϵ. The default value is 0.1.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

useTensionCutoff
A Boolean specifying whether tension cutoff specification is needed. The default value is false.

Table data

The table data specify the following:

- Friction angle (given in degrees), ϕ, at high confining pressure in the p–$R_{mc}q$ plane.
- Dilation angle, ψ, at high confining pressure in the p–$R_{mw}q$ plane.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A MohrCoulombPlasticity object.

Exceptions

RangeError.

Members

The MohrCoulombPlasticity object has members with the same names and descriptions as the arguments to the *MohrCoulombPlasticity* method.

In addition, the MohrCoulombPlasticity object can have the following members:

Prototype

```
odb_MohrCoulombHardening mohrCoulombHardening() const;
odb_TensionCutOff tensionCutOff() const;
```

mohrCoulombHardening

A *MohrCoulombHardening* object.

tensionCutOff

A *TensionCutOff* object.

Corresponding analysis keywords

- **MOHR COULOMB*

MoistureSwelling object

The MoistureSwelling object defines moisture-driven swelling.

Access

```
materialApi.materials()[name].moistureSwelling()
```

MoistureSwelling(...)

This method creates a MoistureSwelling object.

Path

```
materialApi.materials()[name].MoistureSwelling
```

Prototype

```
odb_MoistureSwelling&
MoistureSwelling(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- Volumetric moisture swelling strain, ε^{ms}.
- Saturation, s. This value must lie in the range $0.0 \leq s \leq 1.0$.

Return value

A MoistureSwelling object.

Exceptions

None.

Members

The MoistureSwelling object has members with the same names and descriptions as the arguments to the *MoistureSwelling* method.

In addition, the MoistureSwelling object can have the following member:

Prototype

```
odb_Ratios ratios() const;
```

ratios A *Ratios* object.

Corresponding analysis keywords

- **MOISTURE SWELLING*

MullinsEffect object

The MullinsEffect specifies properties for mullins data.

Access

```
materialApi.materials()[name].mullinsEffect()
```

Members

The MullinsEffect object can have the following members:

Prototype

```
odb_String definition() const;
bool temperatureDependency() const;
int dependencies() const;
int properties() const;
odb_SequenceSequenceDouble table() const;
odb_SequenceUniaxialTestData uniaxialTests() const;
odb_UniaxialTestData uniaxialTests(int index) const;
odb_SequenceBiaxialTestData biaxialTests() const;
odb_BiaxialTestData biaxialTests(int index) const;
odb_SequencePlanarTestData planarTests() const;
odb_PlanarTestData planarTests(int index) const;
```

definition

An odb_String specifying the method of specifying the data. Possible values are "USER", "CONSTANTS", and "TEST_DATA". The default value is "CONSTANTS".

temperatureDependency

A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies

An Int specifying the number of field variable dependencies. The default value is 0.

properties

An Int specifying the number of property values needed as data for the user-defined hyperelastic material. The default value is 0.

table

An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

uniaxialTests

A sequence of *UniaxialTestData* objects.

biaxialTests

A sequence of *BiaxialTestData* objects.

planarTests

A sequence of *PlanarTestData* objects.

Ornl object

The Ornl object specifies the constitutive model developed by Oak Ridge National Laboratory.

Access

```
materialApi.materials()[name].creep().ornl()
materialApi.materials()[name].plastic().ornl()
```

Ornl(...)

This method creates an Ornl object.

Path

```
materialApi.materials()[name].creep().Ornl
materialApi.materials()[name].plastic().Ornl
```

Prototype

```
odb_Ornl&
Ornl(double a,
     odb_Union h,
     bool reset);
```

Required arguments

None.

Optional arguments

a

A Double specifying the saturation rates for kinematic shift caused by creep strain, as defined by Equation (15) of Section 4.3.3–3 of the Nuclear Standard. The default value corresponds to that section of the Standard. Set *a*=0.0 to use the 1986 revision of the Standard. The default value is 0.3.

h

The string "NONE" or a Double specifying the rate of kinematic shift with respect to creep strain [Equation (7) of Section 4.3.2–1 of the Nuclear Standard]. If *h*="NONE", the value of *h* is determined according to Section 4.3.3–3 of the 1981 revision of the Standard. Set *h*=0.0 to use the 1986 revision of the Standard. The default value is "NONE".

reset

A Boolean specifying whether to invoke the optional α reset procedure described in Section 4.3.5 of the Nuclear Standard. The default value is false.

Return value

An Ornl object.

Exceptions

RangeError.

Members

The Ornl object has members with the same names and descriptions as the arguments to the *Ornl* method.

Corresponding analysis keywords

- **ORNL*

Permeability object

The Permeability object defines permeability for pore fluid flow.

Access

```
materialApi.materials()[name].permeability()
```

Permeability(...)

This method creates a Permeability object.

Path

```
materialApi.materials()[name].Permeability
```

Prototype

```
odb_Permeability&
Permeability(double specificWeight,
             double inertialDragCoefficient,
             const odb_SequenceSequenceDouble& table,
             const odb_String& type,
             bool temperatureDependency,
             int dependencies);
```

Required arguments

specificWeight	A Double specifying the specific weight of the wetting liquid, γ_w.
inertialDragCoefficient	A Double specifying The inertial drag coefficient of the wetting liquid, γ_w.
table	An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type	An odb_String specifying the type of permeability. Possible values are "ISOTROPIC", "ORTHOTROPIC", and "ANISOTROPIC". The default value is "ISOTROPIC".
temperatureDependency	A Boolean specifying whether the data depend on temperature. The default value is false.
dependencies	An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=ISOTROPIC, the table data specify the following:

- k.
- Void ratio, e.
- Temperature, if the data depend on temperature.

If *type*=ORTHOTROPIC, the table data specify the following:

- k_{11}.
- k_{22}.
- k_{33}.
- Void ratio, e.
- Temperature, if the data depend on temperature.

If *type*=ANISOTROPIC, the table data specify the following:

- k_{11}.
- k_{12}.
- k_{22}.
- k_{13}.
- k_{23}.
- k_{33}.
- Void ratio, e.
- Temperature, if the data depend on temperature.

Return value

A Permeability object.

Exceptions

RangeError.

Members

The Permeability object has members with the same names and descriptions as the arguments to the *Permeability* method.

In addition, the Permeability object can have the following members:

Prototype

```
odb_SaturationDependence saturationDependence() const;
odb_VelocityDependence velocityDependence() const;
```

saturationDependence

A *SaturationDependence* object specifying the dependence of the permeability of a material on the saturation of the wetting liquid.

velocityDependence

A *VelocityDependence* object specifying the dependence of the permeability of a material on the velocity of fluid flow.

Corresponding analysis keywords

- **PERMEABILITY*

Piezoelectric object

The Piezoelectric object specifies piezoelectric material properties.

Access

```
materialApi.materials()[name].piezoelectric()
```

Piezoelectric(...)

This method creates a Piezoelectric object.

Path

```
materialApi.materials()[name].Piezoelectric
```

Prototype

```
odb_Piezoelectric&
Piezoelectric(const odb_SequenceSequenceDouble& table,
              const odb_String& type,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type An odb_String specifying the type of material coefficients for the piezoelectric property. Possible values are "STRAIN" and "STRESS". The default value is "STRESS".

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRESS, the table data specify the following:

- $e^{\varphi}_{1\,11}$.
- $e^{\varphi}_{1\,22}$.
- $e^{\varphi}_{1\,33}$.
- $e^{\varphi}_{1\,12}$.
- $e^{\varphi}_{1\,13}$.

- $e^{\varphi}_{1\,23}$.
- $e^{\varphi}_{2\,11}$.
- $e^{\varphi}_{2\,22}$.
- $e^{\varphi}_{2\,33}$.
- $e^{\varphi}_{2\,12}$.
- $e^{\varphi}_{2\,13}$.
- $e^{\varphi}_{2\,23}$.
- $e^{\varphi}_{3\,11}$.
- $e^{\varphi}_{3\,22}$.
- $e^{\varphi}_{3\,33}$.
- $e^{\varphi}_{3\,12}$.
- $e^{\varphi}_{3\,13}$.
- $e^{\varphi}_{3\,23}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=STRAIN, the table data specify the following:

- $d^{\varphi}_{1\,11}$.
- $d^{\varphi}_{1\,22}$.
- $d^{\varphi}_{1\,33}$.
- $d^{\varphi}_{1\,12}$.
- $d^{\varphi}_{1\,13}$.
- $d^{\varphi}_{1\,23}$.
- $d^{\varphi}_{2\,11}$.
- $d^{\varphi}_{2\,22}$.
- $d^{\varphi}_{2\,33}$.
- $d^{\varphi}_{2\,12}$.
- $d^{\varphi}_{2\,13}$.

- $d^{\varphi}_{2\,23}$.
- $d^{\varphi}_{3\,11}$.
- $d^{\varphi}_{3\,22}$.
- $d^{\varphi}_{3\,33}$.
- $d^{\varphi}_{3\,13}$.
- $d^{\varphi}_{3\,23}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Piezoelectric object.

Exceptions

None.

Members

The Piezoelectric object has members with the same names and descriptions as the arguments to the *Piezoelectric* method.

Corresponding analysis keywords

- **PIEZOELECTRIC*

PlanarTestData object

The PlanarTestData object specifies planar test (or pure shear) data (compression and/or tension).

Access

```
materialApi.materials()[name].hyperelastic().planarTestData()
materialApi.materials()[name].hyperfoam().planarTestData()
materialApi.materials()[name].mullinsEffect().planarTests(i)
```

PlanarTestData(...)

This method creates a PlanarTestData object.

Path

```
materialApi.materials()[name].hyperelastic().PlanarTestData
materialApi.materials()[name].hyperfoam().PlanarTestData
materialApi.materials()[name].mullinsEffect().PlanarTestData
```

Prototype

```
odb_PlanarTestData&
PlanarTestData(const odb_SequenceSequenceDouble& table,
               odb_Union smoothing,
               bool lateralNominalStrain,
               bool temperatureDependency,
               int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

smoothing
The string "NONE" or an Int specifying the value for smoothing. If *smoothing*="NONE", no smoothing is employed. The default value is "NONE".

lateralNominalStrain
A Boolean specifying whether to include lateral nominal strain. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic material model, the table data specify the following:

- Nominal stress, T_S.

- Nominal strain in the direction of loading, ϵ_S.

For a hyperfoam material model, the table data specify the following:

- Nominal stress, T_L.
- Nominal strain in the direction of loading, ϵ_p.
- Nominal transverse strain, ϵ_3. The default value is 0.

Return value

A PlanarTestData object.

Exceptions

None.

Members

The PlanarTestData object has members with the same names and descriptions as the arguments to the *PlanarTestData* method.

Corresponding analysis keywords

- **PLANAR TEST DATA*

Plastic object

The Plastic object specifies a metal plasticity model.

Access

```
materialApi.materials()[name].plastic()
```

Plastic(...)

This method creates a Plastic object.

Path

```
materialApi.materials()[name].Plastic
```

Prototype

```
odb_Plastic&
Plastic(const odb_SequenceSequenceDouble& table,
        const odb_String& hardening,
        bool rate,
        const odb_String& dataType,
        bool strainRangeDependency,
        int numBackstresses,
        bool temperatureDependency,
        int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

hardening
An odb_String specifying the type of hardening. Possible values are "ISOTROPIC", "KINEMATIC", "COMBINED", "JOHNSON_COOK", and "USER". The default value is "ISOTROPIC".

rate
A Boolean specifying whether the data depend on rate. The default value is false.

dataType
An odb_String specifying the type of combined hardening. This argument is only valid if *hardening*="COMBINED". Possible values are "HALF_CYCLE", "PARAMETERS", and "STABILIZED". The default value is "HALF_CYCLE".

strainRangeDependency
A Boolean specifying whether the data depend on strain range. This argument is only valid if *hardening*="COMBINED" and *dataType*="STABILIZED". The default value is false.

numBackstresses
An Int specifying the number of backstresses. This argument is only valid if *hardening*="COMBINED". The default value is 1.

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *hardening*=ISOTROPIC, or if *hardening*=COMBINED and *dataType*=HALF_CYCLE, the table data specify the following:

- Yield stress.
- Plastic strain.
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=COMBINED and *dataType*=STABILIZED, the table data specify the following:

- Yield stress.
- Plastic strain.
- Strain range, if the data depend on strain range.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=COMBINED and *dataType*=PARAMETERS, the table data specify the following:

- Yield stress at zero plastic strain.
- The first kinematic hardening parameter, C_1.
- The first kinematic hardening parameter, γ_1.
- If applicable, the second kinematic hardening parameter, C_2.
- If applicable, the second kinematic hardening parameter, γ_2.
- Etc.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *hardening*=KINEMATIC, the table data specify the following:

- Yield stress.
- Plastic strain.

- Temperature, if the data depend on temperature.

If *hardening*=JOHNSON_COOK, the table data specify the following:

- A.
- B.
- n.
- m.
- Melting temperature.
- Transition temperature.

If *hardening*=USER, the table data specify the following:

- Hardening properties.

Return value

A Plastic object.

Exceptions

RangeError.

Members

The Plastic object has members with the same names and descriptions as the arguments to the *Plastic* method.

In addition, the Plastic object can have the following members:

Prototype

```
odb_RateDependent rateDependent() const;
odb_Potential potential() const;
odb_CyclicHardening cyclicHardening() const;
odb_Ornl ornl() const;
odb_CycledPlastic cycledPlastic() const;
odb_AnnealTemperature annealTemperature() const;
```

rateDependent A *RateDependent* object.

potential A *Potential* object.

cyclicHardening A *CyclicHardening* object.

ornl An *Ornl* object.

cycledPlastic A *CycledPlastic* object.

annealTemperature An *AnnealTemperature* object.

tensileFailure A *tensileFailure* object.

Corresponding analysis keywords

- **PLASTIC*

PoreFluidExpansion object

The PoreFluidExpansion object specifies the thermal expansion coefficient for a hydraulic fluid.

Access

```
materialApi.materials()[name].poreFluidExpansion()
```

PoreFluidExpansion(...)

This method creates a PoreFluidExpansion object.

Path

```
materialApi.materials()[name].PoreFluidExpansion
```

Prototype

```
odb_PoreFluidExpansion&
PoreFluidExpansion(const odb_SequenceSequenceDouble& table,
                   double zero,
                   bool temperatureDependency,
                   int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

zero
A Double specifying the value of θ^0. The default value is 0.0.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Mean coefficient of thermal expansion, α.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PoreFluidExpansion object.

Exceptions

RangeError.

Members

The PoreFluidExpansion object has members with the same names and descriptions as the arguments to the *PoreFluidExpansion* method.

Corresponding analysis keywords

- **EXPANSION*

PorousBulkModuli object

The PorousBulkModuli object defines bulk moduli for soils and rocks.

Access

```
materialApi.materials()[name].porousBulkModuli()
```

PorousBulkModuli(...)

This method creates a PorousBulkModuli object.

Path

```
materialApi.materials()[name].PorousBulkModuli
```

Prototype

```
odb_PorousBulkModuli&
PorousBulkModuli(const odb_SequenceSequenceDouble& table,
                 bool temperatureDependency);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

Table data

- Bulk modulus of solid grains.
- Bulk modulus of permeating fluid.
- Temperature, if the data depend on temperature.

Return value

A PorousBulkModuli object.

Exceptions

None.

Members

The PorousBulkModuli object has members with the same names and descriptions as the arguments to the *PorousBulkModuli* method.

Corresponding analysis keywords

- **POROUS BULK MODULI*

PorousElastic object

The PorousElastic object specifies elastic material properties for porous materials.

Access

```
materialApi.materials()[name].porousElastic()
```

PorousElastic(...)

This method creates a PorousElastic object.

Path

```
materialApi.materials()[name].PorousElastic
```

Prototype

```
odb_PorousElastic&
PorousElastic(const odb_SequenceSequenceDouble& table,
              const odb_String& shear,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

shear
An odb_String specifying the shear definition form. Possible values are "G" and "POISSON". The default value is "POISSON".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *shear*=G, the table data specify the following:

- The logarithmic bulk modulus, κ. (Dimensionless.)
- The shear modulus, G.
- The elastic tensile limit, p_t^{el}. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *shear*=POISSON, the table data specify the following:

- The logarithmic bulk modulus, κ. (Dimensionless.)
- The Poisson's ratio, ν.
- The elastic tensile limit, p_t^{el}. (This value cannot be negative.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PorousElastic object.

Exceptions

RangeError.

Members

The PorousElastic object has members with the same names and descriptions as the arguments to the *PorousElastic* method.

Corresponding analysis keywords

- **POROUS ELASTIC*

PorousFailureCriteria object

The PorousFailureCriteria object specifies the material failure criteria for a porous metal.

Access

```
materialApi.materials()[name].porousMetalPlasticity()\
.porousFailureCriteria()
```

PorousFailureCriteria(...)

This method creates a PorousFailureCriteria object.

Path

```
materialApi.materials()[name].porousMetalPlasticity()\
.PorousFailureCriteria
```

Prototype

```
odb_PorousFailureCriteria&
PorousFailureCriteria(double fraction,
                      double criticalFraction);
```

Required arguments

None.

Optional arguments

fraction

A Double specifying the void volume fraction at total failure, $f_F > 0$. The default value is 1.0.

criticalFraction

A Double specifying the critical void volume fraction, $f_c \geq 0$. The default value is 1.0.

Return value

A PorousFailureCriteria object.

Exceptions

RangeError.

Members

The PorousFailureCriteria object has members with the same names and descriptions as the arguments to the *PorousFailureCriteria* method.

Corresponding analysis keywords

- **POROUS FAILURE CRITERIA*

PorousMetalPlasticity object

The PorousMetalPlasticity object specifies a porous metal plasticity model.

Access

```
materialApi.materials()[name].porousMetalPlasticity()
```

PorousMetalPlasticity(...)

This method creates a PorousMetalPlasticity object.

Path

```
materialApi.materials()[name].PorousMetalPlasticity
```

Prototype

```
odb_PorousMetalPlasticity&
PorousMetalPlasticity(const odb_SequenceSequenceDouble& table,
                      odb_Union relativeDensity,
                      bool temperatureDependency,
                      int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

relativeDensity
The string "NONE" or a Double specifying the initial relative density of the material, r_0. The default value is "NONE".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- q_1.
- q_2.
- q_3.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PorousMetalPlasticity object.

Exceptions

RangeError.

Members

The PorousMetalPlasticity object has members with the same names and descriptions as the arguments to the *PorousMetalPlasticity* method.

In addition, the PorousMetalPlasticity object can have the following members:

Prototype

```
odb_PorousFailureCriteria porousFailureCriteria() const;
odb_VoidNucleation voidNucleation() const;
```

porousFailureCriteria

A *PorousFailureCriteria* object.

voidNucleation

A *VoidNucleation* object.

Corresponding analysis keywords

- **POROUS METAL PLASTICITY*

Potential object

The Potential object defines an anisotropic yield/creep model.

Access

```
materialApi.materials()[name].creep().potential()
materialApi.materials()[name].plastic().potential()
materialApi.materials()[name].viscous().potential()
```

Potential(...)

This method creates a Potential object.

Path

```
materialApi.materials()[name].creep().Potential
materialApi.materials()[name].plastic().Potential
materialApi.materials()[name].viscous().Potential
```

Prototype

```
odb_Potential&
Potential(const odb_SequenceSequenceDouble& table,
          bool temperatureDependency,
          int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- R_{11}.
- R_{22}.
- R_{33}.
- R_{12}.
- R_{13}.
- R_{23}.
- Temperature, if the data depend on temperature.

- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Potential object.

Exceptions

RangeError.

Members

The Potential object has members with the same names and descriptions as the arguments to the *Potential* method.

Corresponding analysis keywords

- **POTENTIAL*

PressureEffect object

The PressureEffect object defines equivalent pressure stress driven mass diffusion.

Access

```
materialApi.materials()[name].diffusivity().pressureEffect()
```

PressureEffect(...)

This method creates a PressureEffect object.

Path

```
materialApi.materials()[name].diffusivity().PressureEffect
```

Prototype

```
odb_PressureEffect&
PressureEffect(const odb_SequenceSequenceDouble& table,
               bool temperatureDependency,
               int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Pressure stress factor, κ_p.
- Concentration.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A PressureEffect object.

Exceptions

RangeError.

Members

The PressureEffect object has members with the same names and descriptions as the arguments to the *PressureEffect* method.

Corresponding analysis keywords

- **KAPPA*

RateDependent object

The RateDependent object defines a rate-dependent viscoplastic model.

Access

```
materialApi.materials()[name].crushableFoam().rateDependent()
materialApi.materials()[name].druckerPrager().rateDependent()
materialApi.materials()[name].plastic().rateDependent()
```

RateDependent(...)

This method creates a RateDependent object.

Path

```
materialApi.materials()[name].crushableFoam().RateDependent
materialApi.materials()[name].druckerPrager().RateDependent
materialApi.materials()[name].plastic().RateDependent
```

Prototype

```
odb_RateDependent&
RateDependent(const odb_SequenceSequenceDouble& table,
              const odb_String& type,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type An odb_String specifying the type of rate-dependent data. Possible values are "POWER_LAW", "YIELD_RATIO", and "JOHNSON_COOK". The default value is "POWER_LAW".

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=POWER_LAW, the table data specify the following:

- D.
- n.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.

- Value of the second field variable.
- Etc.

If *type*=YIELD_RATIO, the table data specify the following:

- Yield stress ratio, $R = \overline{\sigma}/\sigma^0$.
- Equivalent plastic strain rate, $\dot{\bar{\varepsilon}}^{pl}$.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=JOHNSON_COOK, the table data specify the following:

- C.
- $\dot{\varepsilon}_0$.

Return value

A RateDependent object.

Exceptions

RangeError.

Members

The RateDependent object has members with the same names and descriptions as the arguments to the *RateDependent* method.

Corresponding analysis keywords

- **RATE DEPENDENT*

Ratios object

The Ratios object specifies ratios that define anisotropic swelling.

Access

```
materialApi.materials()[name].moistureSwelling().ratios()
materialApi.materials()[name].swelling().ratios()
```

Ratios(...)

This method creates a Ratios object.

Path

```
materialApi.materials()[name].moistureSwelling().Ratios
materialApi.materials()[name].swelling().Ratios
```

Prototype

```
odb_Ratios&
Ratios(const odb_SequenceSequenceDouble& table,
       bool temperatureDependency,
       int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- r_{11}.
- r_{22}.
- r_{33}.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Ratios object.

Exceptions

RangeError.

Members

The Ratios object has members with the same names and descriptions as the arguments to the *Ratios* method.

Corresponding analysis keywords

- **RATIOS*

Regularization object

The Regularization object defines the tolerance to be used for regularizing material data.

Access

```
materialApi.materials()[name].regularization()
```

Regularization(...)

This method creates a Regularization object.

Path

```
materialApi.materials()[name].Regularization
```

Prototype

```
odb_Regularization&
Regularization(double rtol,
               const odb_String& strainRateRegularization);
```

Required arguments

None.

Optional arguments

rtol

A Double specifying the tolerance to be used for regularizing material data. The default value is 0.03.

strainRateRegularization

An odb_String specifying the form of regularization of strain-rate-dependent material data. Possible values are "LOGARITHMIC" and "LINEAR". The default value is "LOGARITHMIC".

Return value

A Regularization object.

Exceptions

RangeError.

Members

The Regularization object has members with the same names and descriptions as the arguments to the *Regularization* method.

Corresponding analysis keywords

- **DASHPOT*

SaturationDependence object

The SaturationDependence object specifies the dependence of the permeability of a material on the saturation of the wetting liquid.

Access

```
materialApi.materials()[name].permeability().saturationDependence()
```

SaturationDependence(...)

This method creates a SaturationDependence object.

Path

```
materialApi.materials()[name].permeability().SaturationDependence
```

Prototype

```
odb_SaturationDependence&
SaturationDependence(const odb_SequenceSequenceDouble& table);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- k_s. (Dimensionless.)
- Saturation, s. (Dimensionless.)

Return value

A SaturationDependence object.

Exceptions

RangeError.

Members

The SaturationDependence object has members with the same names and descriptions as the arguments to the *SaturationDependence* method.

Corresponding analysis keywords

- **PERMEABILITY*

ShearRetention object

The ShearRetention object defines the reduction of the shear modulus associated with crack surfaces in a *Concrete* model as a function of the tensile strain across the crack.

Access

```
materialApi.materials()[name].concrete().shearRetention()
```

ShearRetention(...)

This method creates a ShearRetention object.

Path

```
materialApi.materials()[name].concrete().ShearRetention
```

Prototype

```
odb_ShearRetention&
ShearRetention(const odb_SequenceSequenceDouble& table,
               bool temperatureDependency,
               int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- ϱ^{close} for dry concrete. The default value is 1.0.
- ε^{max} for dry concrete. The default value is a very large number (full shear retention).
- ϱ^{close} for wet concrete. The default value is 1.0.
- ε^{max} for wet concrete. The default value is a very large number (full shear retention).
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A ShearRetention object.

Exceptions

RangeError.

Members

The ShearRetention object has members with the same names and descriptions as the arguments to the *ShearRetention* method.

Corresponding analysis keywords

- **SHEAR RETENTION*

ShearTestData object

The ShearTestData object specifies the normalized shear creep compliance or relaxation modulus as a function of time.

Access

```
materialApi.materials()[name].viscoelastic().shearTestData()
```

ShearTestData(...)

This method creates a ShearTestData object.

Path

```
materialApi.materials()[name].viscoelastic().ShearTestData
```

Prototype

```
odb_ShearTestData&
ShearTestData(const odb_SequenceSequenceDouble& table,
              odb_Union shrinf);
```

Required arguments

table

An odb_SequenceSequenceDouble specifying values that depend on the *time* member of the *Viscoelastic* object.

If *time*="RELAXATION_TEST_DATA", the table data specify the following:

- Normalized shear relaxation modulus $g_R(t)$. $(0 \le g_R(t) \le 1)$.
- Time t. $(t > 0)$.

If *time*="CREEP_TEST_DATA", the table data specify the following:

- Normalized shear compliance $j_S(t)$. $(j_S(t) \ge 1)$.
- Time t. $(t > 0)$.

Optional arguments

shrinf

The string "NONE" or a Double specifying a normalized shear. The value of *shrinf* depends on the value of the *time* member of the *Viscoelastic* object. The default value is "NONE".

If *time*="RELAXATION_TEST_DATA", *shrinf* specifies the value of the long-term, normalized shear modulus $g_R(\infty)$.

If *time*="CREEP_TEST_DATA", *shrinf* specifies the value of the long-term, normalized shear compliance $j_S(\infty)$.

Return value

A ShearTestData object.

Exceptions

None.

Members

The ShearTestData object has members with the same names and descriptions as the arguments to the *ShearTestData* method.

Corresponding analysis keywords

- **SHEAR TEST DATA*

SimpleShearTestData object

The SimpleShearTestData object provides simple shear test data.

Access

```
materialApi.materials()[name].hyperfoam().simpleShearTestData()
```

SimpleShearTestData(...)

This method creates a SimpleShearTestData object.

Path

```
materialApi.materials()[name].hyperfoam().SimpleShearTestData
```

Prototype

```
odb_SimpleShearTestData&
SimpleShearTestData(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- Nominal shear stress, T_S.
- Nominal shear strain, γ.
- Nominal transverse stress, T_T (normal to edge with shear stress). This stress value is optional.

Return value

A SimpleShearTestData object.

Exceptions

None.

Members

The SimpleShearTestData object has members with the same names and descriptions as the arguments to the *SimpleShearTestData* method.

Corresponding analysis keywords

- **SIMPLE SHEAR TEST DATA*

Solubility object

The Solubility object specifies solubility.

Access

```
materialApi.materials()[name].solubility()
```

Solubility(...)

This method creates a Solubility object.

Path

```
materialApi.materials()[name].Solubility
```

Prototype

```
odb_Solubility&
Solubility(const odb_SequenceSequenceDouble& table,
           bool temperatureDependency,
           int dependencies);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Solubility.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Solubility object.

Exceptions

RangeError.

Members

The Solubility object has members with the same names and descriptions as the arguments to the *Solubility* method.

Corresponding analysis keywords

- **SOLUBILITY*

SoretEffect object

The SoretEffect object defines temperature gradient driven mass diffusion.

Access

```
materialApi.materials()[name].diffusivity().soretEffect()
```

SoretEffect(...)

This method creates a SoretEffect object.

Path

```
materialApi.materials()[name].diffusivity().SoretEffect
```

Prototype

```
odb_SoretEffect&
SoretEffect(const odb_SequenceSequenceDouble& table,
            bool temperatureDependency,
            int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Soret effect factor, κ_s.
- Concentration.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A SoretEffect object.

Exceptions

RangeError.

Members

The SoretEffect object has members with the same names and descriptions as the arguments to the *SoretEffect* method.

Corresponding analysis keywords

- **KAPPA*

Sorption object

The Sorption object defines absorption and exsorption behaviors of a partially saturated porous medium in the analysis of coupled wetting liquid flow and porous medium stress.

Access

```
materialApi.materials()[name].sorption()
```

Sorption(...)

This method creates a Sorption object.

Path

```
materialApi.materials()[name].Sorption
```

Prototype

```
odb_Sorption&
Sorption(const odb_SequenceSequenceDouble& absorptionTable,
         const odb_String& lawAbsorption,
         bool exsorption,
         const odb_String& lawExsorption,
         double scanning,
         const odb_SequenceSequenceDouble& exsorptionTable);
```

Required arguments

absorptionTable
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

lawAbsorption
An odb_String specifying absorption behavior. Possible values are "LOG" and "TABULAR". The default value is "TABULAR".

exsorption
A Boolean specifying whether the exsorption data is specified. The default value is false.

lawExsorption
An odb_String specifying exsorption behavior. Possible values are "LOG" and "TABULAR". The default value is "TABULAR".

scanning
A Double specifying the slope of the scanning line, $(du_w/ds)|_s$. This slope must be positive and larger than the slope of the absorption or exsorption behaviors. The default value is 0.0.

exsorptionTable
An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

Table data

If *lawAbsorption*=TABULAR or *lawExsorption*=TABULAR, the *absorptionTable* and *exsorptionTable* data respectively specify the following:

- Pore pressure, u_w.
- Saturation, s.

If *lawAbsorption*=LOG or *lawExsorption*=LOG, the *absorptionTable* and *exsorptionTable* data respectively specify the following:

- A.
- B.
- s_0.
- s_1.

Return value

A Sorption object.

Exceptions

RangeError.

Members

The Sorption object has members with the same names and descriptions as the arguments to the *Sorption* method.

Corresponding analysis keywords

- **SORPTION*

SpecificHeat object

The SpecificHeat object specifies a material's specific heat.

Access

```
materialApi.materials()[name].specificHeat()
```

SpecificHeat(...)

This method creates a SpecificHeat object.

Path

```
materialApi.materials()[name].SpecificHeat
```

Prototype

```
odb_SpecificHeat&
SpecificHeat(const odb_SequenceSequenceDouble& table,
             const odb_String& law,
             bool temperatureDependency,
             int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law
An odb_String specifying the specific heat behavior. Possible values are "CONSTANTVOLUME" and "CONSTANTPRESSURE". The default value is "CONSTANTVOLUME".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Specific heat per unit mass.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A SpecificHeat object.

Exceptions

RangeError.

Members

The SpecificHeat object has members with the same names and descriptions as the arguments to the *SpecificHeat* method.

Corresponding analysis keywords

- **SPECIFIC HEAT*

SuperElasticHardening object

The SuperElasticHardening object specifies the dependence of the yield stress on the total strain to define the piecewise linear hardening of a martensite material model.

Access

```
materialApi.materials()[name].superElasticity()\
.superElasticHardening()
```

SuperElasticHardening(...)

This method creates a SuperElasticHardening object.

Path

```
materialApi.materials()[name].superElasticity()\
.SuperElasticHardening
```

Prototype

```
odb_SuperElasticHardening&
SuperElasticHardening(const odb_SequenceSequenceDouble& table);
```

Required arguments

table A sequence of sequences of Floats specifying the items described below.

Table data

- Yield Stress.
- Total Strain.

Return value

A SuperElasticHardening object.

Exceptions

RangeError.

Members

The SuperElasticHardeningobject has members with the same names and descriptions as the arguments to the *SuperElasticHardening* method.

Corresponding analysis keywords

- **SUPERELASTIC HARDENING*

SuperElasticHardeningModifications object

The SuperElasticHardeningModifications object specifies the variation of the transformation stress levels of a material model.

Access

```
materialApi.materials()[name].superElasticity()\
.superElasticHardeningModifications()
```

SuperElasticHardeningModifications(...)

This method creates a SuperElasticHardeningModifications object.

Path

```
materialApi.materials()[name].superElasticity()\
.SuperElasticHardeningModifications
```

Prototype

```
odb_SuperElasticHardeningModifications&
SuperElasticHardeningModifications(const odb_SequenceSequenceDouble& table);
```

Required arguments

table

A sequence of sequences of Floats specifying the items described below or user-defined data if the dependence of the transformation stress levels on plastic strain is specified in a user subroutine.

Table data

- Start of Transformation (Loading).
- End of Transformation (Loading).
- Start of Transformation (Unloading).
- End of Transformation (Unloading).
- Plastic Strain.

Return value

A SuperElasticHardeningModifications object.

Exceptions

RangeError.

Members

The SuperElasticHardeningModifications object has members with the same names and descriptions as the arguments to the *SuperElasticHardeningModifications* method.

Corresponding analysis keywords

- **SUPERELASTIC HARDENING MODIFICATIONS*

SuperElasticity object

The SuperElasticity object specifies a superelastic material model.

Access

```
materialApi.materials()[name].superElasticity()
```

SuperElasticity(...)

This method creates a SuperElasticity object.

Path

```
materialApi.materials()[name].SuperElasticity
```

Prototype

```
odb_SuperElasticity&
SuperElasticity(const odb_SequenceSequenceDouble& table,
                            odb_Union nonassociated);
```

Required arguments

table
A sequence of sequences of Floats specifying the items described below.

Optional arguments

nonassociated
The string "NONE" or a Double specifying the volumetric transformation strain. If *nonassociated*="NONE", the value of the volumetric transformation strain is equal to the uniaxial transformation strain. The default value is "NONE".

Table data

- Young's Modulus (Martensite).
- Poisson's Ratio (Martensite).
- Transformation Strain.
- Start of Transformation (Loading).
- End of Transformation (Loading).
- Start of Transformation (Unloading).
- End of Transformation (Unloading).
- Start of Transformation in Compression (Loading).
- Reference Temperature.
- Loading.
- Unloading.

Return value

A SuperElasticity object.

Exceptions

RangeError.

Members

The SuperElasticity object has members with the same names and descriptions as the arguments to the *SuperElasticity* method.

In addition, the SuperElasticity object can have the following members:

Prototype

```
odb_SuperElasticHardening superElasticHardening() const;
odb_SuperElasticHardeningModifications superElasticHardeningModifications()
const;
```

superElasticHardening	A *SuperElasticHardening object*.
superElasticHardeningModifications	A *SuperElasticHardeningModifications object*.

Corresponding analysis keywords

- **SUPERELASTIC*

Swelling object

The Swelling object specifies time-dependent volumetric swelling for a material.

Access

```
materialApi.materials()[name].swelling()
```

Swelling(...)

This method creates a Swelling object.

Path

```
materialApi.materials()[name].Swelling
```

Prototype

```
odb_Swelling&
Swelling(const odb_SequenceSequenceDouble& table,
         const odb_String& law,
         bool temperatureDependency,
         int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

This argument is valid only when *law*="INPUT".

Optional arguments

law
An odb_String specifying the type of data defining the swelling behavior. Possible values are "INPUT" and "USER". The default value is "INPUT".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Volumetric swelling strain rate.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Swelling object.

Exceptions

RangeError.

Members

The Swelling object has members with the same names and descriptions as the arguments to the *Swelling* method.

In addition, the Swelling object can have the following member:

Prototype

```
odb_Ratios ratios() const;
```

ratios

A *Ratios* object.

Corresponding analysis keywords

- **SWELLING*

TensileFailure object

The TensileFailure object specifies the material tensile failure.

Access

```
materialApi.materials()[name].plastic().tensileFailure()
materialApi.materials()[name].eos().tensileFailure()
```

TensileFailure(...)

This method creates a TensileFailure object.

Path

```
materialApi.materials()[name].plastic().TensileFailure
materialApi.materials()[name].plastic().TensileFailure
```

Prototype

```
odb_TensileFailure&
TensileFailure(const odb_SequenceSequenceDouble& table,
                   int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

temperatureDependency
A boolean specifying whether the data depend on temperature. The default value is OFF.

elementDeletion
A boolean specifying whether element deletion is allowed. The default value is True.

pressure
An odb_String specifying the pressure stress. The Possible values are BRITTLE and DUCTILE.

shear
An odb_String specifying the deviatoric stress. Possible values are BRITTLE and DUCTILE.

Table data

- The Hydrostatic cutoff stress (positive in tension).
- Temperature, if the data depend on temperature.
- Value of the first field variable if the data depend on field variables.

- Value of the second field variable.
- Etc.

Return value

A TensileFailure object.

Exceptions

RangeError.

Members

The TensileFailure object has members with the same names and descriptions as the arguments to the *TensileFailure* method.

Corresponding analysis keywords

- **TENSILEFAILURE*

TensionCutOff object

The TensionCutOff object specifies tension cutoff for different material models for example the Mohr-Coulomb plasticity model.

Access

```
materialApi.materials()[name].mohrCoulombPlasticity().tensionCutOff()
```

TensionCutOff(...)

This method creates a TensionCutOff object.

Path

```
materialApi.materials()[name].mohrCoulombPlasticity().TensionCutOff
```

Prototype

```
odb_TensionCutOff&
TensionCutOff(const odb_SequenceSequenceDouble& table,
              bool temperatureDependency,
              int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- Tension cutoff stress.
- The value of the corresponding tensile plastic strain.(The first tabular value entered must always be zero.)
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A TensionCutOff object.

Exceptions

RangeError.

Members

The TensionCutOff object has members with the same names and descriptions as the arguments to the *TensionCutOff* method.

Corresponding analysis keywords

- **TENSION CUTOFF*

TensionStiffening object

The TensionStiffening object defines the retained tensile stress normal to a crack in a *Concrete* model.

Access

```
materialApi.materials()[name].concrete().tensionStiffening()
```

TensionStiffening(...)

This method creates a TensionStiffening object.

Path

```
materialApi.materials()[name].concrete().TensionStiffening
```

Prototype

```
odb_TensionStiffening&
TensionStiffening(const odb_SequenceSequenceDouble& table,
                  const odb_String& type,
                  bool temperatureDependency,
                  int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying how the postcracking behavior is defined. Possible values are "DISPLACEMENT" and "STRAIN". The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=STRAIN, the table data specify the following:

- Fraction of remaining stress to stress at cracking.
- Absolute value of the direct strain minus the direct strain at cracking.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

If *type*=DISPLACEMENT, the table data specify the following:

- Displacement, u_0, at which a linear loss of strength after cracking gives zero stress.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A TensionStiffening object.

Exceptions

RangeError.

Members

The TensionStiffening object has members with the same names and descriptions as the arguments to the *TensionStiffening* method.

Corresponding analysis keywords

- **TENSION STIFFENING*

TriaxialTestData object

The TriaxialTestData object provides triaxial test data.

Access

```
materialApi.materials()[name].druckerPrager().triaxialTestData()
```

TriaxialTestData(...)

This method creates a TriaxialTestData object.

Path

```
materialApi.materials()[name].druckerPrager().TriaxialTestData
```

Prototype

```
odb_TriaxialTestData&
TriaxialTestData(const odb_SequenceSequenceDouble& table,
                 odb_Union a,
                 odb_Union b,
                 odb_Union pt);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

a
The string "NONE" or a Double specifying the value of the material constant a. "NONE" is used when the value is unknown or it is not held fixed at the input value. The default value is "NONE".

b
The string "NONE" or a Double specifying the value of the material constant b. "NONE" is used when the value is unknown or it is not held fixed at the input value. The default value is "NONE".

pt
The string "NONE" or a Double specifying the value of the material constant p_t. "NONE" is used when the value is unknown or it is not held fixed at the input value. The default value is "NONE".

Table data

- Sign and magnitude of confining stress, $\sigma_1 = \sigma_2$.
- Sign and magnitude of the stress in loading direction, σ_3.

Return value

A TriaxialTestData object.

Exceptions

RangeError.

Members

The TriaxialTestData object has members with the same names and descriptions as the arguments to the *TriaxialTestData* method.

Corresponding analysis keywords

- **TRIAXIAL TEST DATA*

Trs object

The Trs object defines the temperature-time shift for time history viscoelastic analysis.

Access

```
materialApi.materials()[name].viscoelastic().trs()
materialApi.materials()[name].viscosity().trs()
```

Trs(...)

This method creates a Trs object.

Path

```
materialApi.materials()[name].viscoelastic().Trs
materialApi.materials()[name].viscosity().Trs
```

Prototype

```
odb_Trs&
Trs(const odb_String& definition,
    const odb_SequenceSequenceDouble& table);
```

Required arguments

None.

Optional arguments

definition

An odb_String specifying the definition of the shift function. Possible values are "WLF", "ARRHENIUS", and "USER". The default value is "WLF".

table

An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

This argument is valid only when *definition*="WLF".

Table data

- Reference temperature, θ_0.
- Calibration constant, C_1.
- Calibration constant, C_2.

Return value

A Trs object.

Exceptions

None.

Members

The Trs object has members with the same names and descriptions as the arguments to the *Trs* method.

Corresponding analysis keywords

- **TRS*

UniaxialTestData object

The UniaxialTestData object provides uniaxial test data (compression and/or tension).

Access

```
materialApi.materials()[name].hyperelastic().uniaxialTestData()
materialApi.materials()[name].hyperfoam().uniaxialTestData()
materialApi.materials()[name].lowDensityFoam()\
.uniaxialCompressionTestData()
materialApi.materials()[name].lowDensityFoam().uniaxialTensionTestData()
materialApi.materials()[name].mullinsEffect().uniaxialTests(i)
```

UniaxialTestData(...)

This method creates a UniaxialTestData object.

Path

```
materialApi.materials()[name].hyperelastic().UniaxialTestData
materialApi.materials()[name].hyperfoam().UniaxialTestData
materialApi.materials()[name].lowDensityFoam().UniaxialTestData
materialApi.materials()[name].mullinsEffect().UniaxialTestData
```

Prototype

```
odb_UniaxialTestData&
UniaxialTestData(const odb_SequenceSequenceDouble& table,
                 odb_Union smoothing,
                 bool lateralNominalStrain,
                 bool temperatureDependency,
                 int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

smoothing
The string "NONE" or an Int specifying the value for smoothing. If *smoothing*="NONE", no smoothing is employed. The default value is "NONE".

lateralNominalStrain
A Boolean specifying whether to include lateral nominal strain. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic material model, the table data specify the following:

- Nominal stress, T_U.
- Nominal strain, ϵ_U.

For a hyperfoam material model, the table data specify the following:

- Nominal stress, T_L.
- Nominal strain, ϵ_U.
- Nominal lateral strain, $\epsilon_2 = \epsilon_3$. The default value is 0.

For a low-density foam material model, the table data specify the following:

- Nominal stress, T_U.
- Nominal strain, ϵ_U.
- Nominal strain rate, $\dot{\epsilon_U}$.

Return value

A UniaxialTestData object.

Exceptions

None.

Members

The UniaxialTestData object has members with the same names and descriptions as the arguments to the *UniaxialTestData* method.

Corresponding analysis keywords

- **UNIAXIAL TEST DATA*

UserDefinedField object

The UserDefinedField object redefines field variables at a material point.

Access

```
materialApi.materials()[name].userDefinedField()
```

UserDefinedField()

This method defines a UserDefinedField object.

Path

```
materialApi.materials()[name].UserDefinedField
```

Prototype

```
odb_UserDefinedField&
UserDefinedField();
```

Arguments

None.

Return value

A UserDefinedField object.

Exceptions

None.

Members

The UserDefinedField object has no members.

Corresponding analysis keywords

- **USER DEFINED FIELD*

UserMaterial object

The UserMaterial object defines material constants for use in subroutines *UMAT*, *UMATHT*, or *VUMAT*.

Access

```
materialApi.materials()[name].userMaterial()
```

UserMaterial(...)

This method creates a UserMaterial object.

Path

```
materialApi.materials()[name].UserMaterial
```

Prototype

```
odb_UserMaterial&
UserMaterial(const odb_String& type,
             bool unsymm,
             const odb_SequenceDouble& mechanicalConstants,
             const odb_SequenceDouble& thermalConstants);
```

Required arguments

None.

Optional arguments

type

An odb_String specifying the type of material behavior defined by the command. Possible values are "MECHANICAL", "THERMAL", and "THERMOMECHANICAL". The default value is "MECHANICAL".

unsymm

A Boolean specifying if the material stiffness matrix, $\partial \Delta \boldsymbol{\sigma} / \partial \Delta \boldsymbol{\varepsilon}$, is not symmetric or, when a thermal constitutive model is used, if $\partial \mathbf{f} / \partial (\partial \theta / \partial \mathbf{x})$ is not symmetric. The default value is false.

This argument is valid only in an Abaqus/Standard analysis.

mechanicalConstants

An odb_SequenceDouble specifying the mechanical constants of the material. This argument is valid only when *type*="MECHANICAL" or "THERMOMECHANICAL". The default value is an empty sequence.

thermalConstants

An odb_SequenceDouble specifying the thermal constants of the material. This argument is valid only when *type*="THERMAL" or "THERMOMECHANICAL". The default value is an empty sequence.

Return value

A UserMaterial object.

Exceptions

RangeError.

Members

The UserMaterial object has members with the same names and descriptions as the arguments to the *UserMaterial* method.

Corresponding analysis keywords

- **USER MATERIAL*

UserOutputVariables object

The UserOutputVariables object specifies the number of user-defined output variables.

Access

```
materialApi.materials()[name].userOutputVariables()
```

UserOutputVariables(...)

This method creates a UserOutputVariables object.

Path

```
materialApi.materials()[name].UserOutputVariables
```

Prototype

```
odb_UserOutputVariables&
UserOutputVariables(int n);
```

Required arguments

None.

Optional arguments

n
An Int specifying the number of user-defined variables required at each material point. The default value is 0.

Return value

A UserOutputVariables object.

Exceptions

RangeError.

Members

The UserOutputVariables object has members with the same names and descriptions as the arguments to the *UserOutputVariables* method.

Corresponding analysis keywords

- **USER OUTPUT VARIABLES*

VelocityDependence object

The VelocityDependence object specifies the dependence of the permeability of a material on the velocity of fluid flow.

Access

```
materialApi.materials()[name].permeability().velocityDependence()
```

VelocityDependence(...)

This method creates a VelocityDependence object.

Path

```
materialApi.materials()[name].permeability().VelocityDependence
```

Prototype

```
odb_VelocityDependence&
VelocityDependence(const odb_SequenceSequenceDouble& table);
```

Required arguments

table An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

None.

Table data

- β. Only $\beta > 0.0$ is allowed.
- Void ratio, e.

Return value

A VelocityDependence object.

Exceptions

RangeError.

Members

The VelocityDependence object has members with the same names and descriptions as the arguments to the *VelocityDependence* method.

Corresponding analysis keywords

- **PERMEABILITY*

Viscoelastic object

The Viscoelastic object specifies dissipative behavior for use with elasticity.

Access

```
materialApi.materials()[name].viscoelastic()
```

Viscoelastic(...)

This method creates a Viscoelastic object.

Path

```
materialApi.materials()[name].Viscoelastic
```

Prototype

```
odb_Viscoelastic&
Viscoelastic(const odb_String& domain,
           const odb_SequenceSequenceDouble& table,
           const odb_String& frequency,
           const odb_String& type,
           const odb_String& preload,
           const odb_String& time,
           double errtol,
           int nmax,
           const odb_SequenceSequenceDouble& volumetricTable);
```

Required arguments

domain

An odb_String specifying the domain definition. Possible values are:

- "FREQUENCY", specifying a frequency domain. This domain is only available for an Abaqus/Standard analysis.
- "TIME", specifying a time domain.

table

An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

frequency

An odb_String specifying the frequency domain definition. This argument is required only when *domain*="FREQUENCY". Possible values are "FORMULA", "TABULAR", "PRONY", "CREEP_TEST_DATA", and "RELAXATION_TEST_DATA". The default value is "FORMULA".

type

An odb_String specifying the type. This argument is required only when *domain*="FREQUENCY" and *frequency*="TABULAR". Possible values are "ISOTROPIC" and "TRACTION". The default value is "ISOTROPIC".

preload

An odb_String specifying the preload. This argument is required only when *domain*="FREQUENCY" and *frequency*="TABULAR". Possible values are "NONE", "UNIAXIAL", "VOLUMETRIC", and "UNIAXIAL_VOLUMETRIC". The default value is "NONE".

time
An odb_String specifying the time domain definition. This argument is required only when *domain*="TIME". Possible values are "PRONY", "CREEP_TEST_DATA", "RELAXATION_TEST_DATA", and "FREQUENCY_DATA". The default value is "PRONY".

errtol
A Double specifying the allowable average root-mean-square error of the data points in the least-squares fit. The Float values correspond to percentages; for example, 0.01 is 1%. The default value is 0.01.

This argument is valid only when *time*="CREEP_TEST_DATA", "RELAXATION_TEST_DATA" or "FREQUENCY_DATA"; or only when *frequency*="CREEP_TEST_DATA" or "RELAXATION_TEST_DATA".

nmax
An Int specifying the maximum number of terms N in the Prony series. The maximum value is 13. The default value is 13.

This argument is valid only when *time*="CREEP_TEST_DATA", "RELAXATION_TEST_DATA" or "FREQUENCY_DATA"; or only when *frequency*="CREEP_TEST_DATA" or "RELAXATION_TEST_DATA".

volumetricTable
An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

Table data

If *frequency*=FORMULA, the table data for *table* specify the following:

- Real part of $g_1^*(g^*(\omega) = g_1^* f^{-a})$.
- Imaginary part of g_1^*.
- Value of a.
- Real part of $k_1^*(k^*(\omega) = k_1^* f^{-b})$. If the material is incompressible, this value is ignored.
- Imaginary part of k_1^*. If the material is incompressible, this value is ignored.
- Value of b. If the material is incompressible, this value is ignored.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=NONE, or *time*=FREQUENCY_DATA the table data for *table* specify the following:

- Real part of $\omega g^*(\omega \Re(g^*) = G_\ell / G_\infty)$.
- Imaginary part of $\omega g^*(\omega \Im(g^*) = 1 - G_s / G_\infty)$.
- Real part of $\omega k^*(\omega \Re(k^*) = \mathrm{K}_\ell / \mathrm{K}_\infty)$. If the material is incompressible, this value is ignored.
- Imaginary part of $\omega k^*(\omega \Im(k^*) = 1 - \mathrm{K}_s / \mathrm{K}_\infty)$. If the material is incompressible, this value is ignored.
- Frequency f in cycles per time.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=UNIAXIAL the table data for *table* specify the following:

- Loss modulus.
- Storage modulus.

- Frequency.
- Uniaxial strain.

If *frequency*=TABULAR and *type*=TRACTION and *preload*=NONE the table data for *table* specify the following:

- Normalized loss modulus.
- Normalized shear modulus.
- Frequency.

If *frequency*=TABULAR and *type*=TRACTION and *preload*=UNIAXIAL or *preload*=UNIAXIAL_VOLUMETRIC the table data for *table* specify the following:

- Loss modulus.
- Storage modulus.
- Frequency.
- Closure.

If *time*=PRONY or *frequency*=PRONY, the table data for *table* specify the following:

- $\overline{g}_1^P$, the modulus ratio in the first term in the Prony series expansion of the shear relaxation modulus.
- $\overline{k}_1^P$, the modulus ratio in the first term in the Prony series expansion of the bulk relaxation modulus.
- τ_1, the relaxation time for the first term in the Prony series expansion.

If *frequency*=TABULAR and *type*=ISOTROPIC and *preload*=VOLUMETRIC or *preload*=UNIAXIAL_VOLUMETRIC the table data for *volumetricTable* specify the following:

- Loss modulus.
- Storage modulus.
- Frequency.
- Volume ratio.

Return value

A Viscoelastic object.

Exceptions

RangeError.

Members

The Viscoelastic object has members with the same names and descriptions as the arguments to the *Viscoelastic* method.

In addition, the Viscoelastic object can have the following members:

Prototype

```
odb_CombinedTestData combinedTestData() const;
odb_ShearTestData shearTestData() const;
odb_Trs trs() const;
odb_VolumetricTestData volumetricTestData() const;
```

combinedTestData	A *CombinedTestData* object.
shearTestData	A *ShearTestData* object.
trs	A *Trs* object.
volumetricTestData	A *VolumetricTestData* object.

Corresponding analysis keywords

- **VISCOELASTIC*

Viscosity object

The Viscosity object specifies mechanical viscosity.

Access

```
materialApi.materials()[name].viscosity()
```

Viscosity(...)

This method creates a Viscosity object.

Path

```
materialApi.materials()[name].Viscosity
```

Prototype

```
odb_Viscosity&
Viscosity(const odb_SequenceSequenceDouble& table,
          const odb_String& type,
          bool temperatureDependency,
          int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

type
An odb_String specifying the type of viscosity. The default value is "NEWTONIAN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *type*=NEWTONIAN, the table data specify the following:

- Viscosity, k.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Viscosity object.

Exceptions

RangeError.

Members

The Viscosity object has members with the same names and descriptions as the arguments to the *Viscosity* method. In addition, the Viscosity object can have the following member:

Prototype

```
odb_Trs trs() const;
```

trs A *Trs* object.

Corresponding analysis keywords

- **VISCOSITY*

Viscous object

The Viscous object specifies the viscous properties for a two-layer viscoplastic material model.

Access

```
materialApi.materials()[name].viscous()
```

Viscous(...)

This method creates a Viscous object.

Path

```
materialApi.materials()[name].Viscous
```

Prototype

```
odb_Viscous&
Viscous(const odb_SequenceSequenceDouble& table,
        const odb_String& law,
        bool temperatureDependency,
        int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

law
An odb_String specifying the creep law. Possible values are "STRAIN", "TIME", and "USER". The default value is "STRAIN".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

If *law*=STRAIN or *law*=TIME, the table data specify the following:

- A.
- n.
- m.
- f.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.

- Etc.

If *law*=USER, the table data specify the following:

- f.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A Viscous object.

Exceptions

None.

Members

The Viscous object has members with the same names and descriptions as the arguments to the *Viscous* method.

In addition, the Viscous object can have the following member:

Prototype

```
odb_Potential potential() const;
```

potential

A *Potential* object.

Corresponding analysis keywords

- **VISCOUS*

VoidNucleation object

The VoidNucleation object defines the nucleation of voids in a porous material.

Access

```
materialApi.materials()[name].porousMetalPlasticity().voidNucleation()
```

VoidNucleation(...)

This method creates a VoidNucleation object.

Path

```
materialApi.materials()[name].porousMetalPlasticity().VoidNucleation
```

Prototype

```
odb_VoidNucleation&
VoidNucleation(const odb_SequenceSequenceDouble& table,
               bool temperatureDependency,
               int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

- ε_N, the mean value of the nucleation-strain normal distribution.
- s_N, the standard deviation of the nucleation-strain normal distribution.
- f_N, the volume fraction of nucleating voids.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A VoidNucleation object.

Exceptions

RangeError.

Members

The VoidNucleation object has members with the same names and descriptions as the arguments to the *VoidNucleation* method.

Corresponding analysis keywords

- **VOID NUCLEATION*

VolumetricTestData object

The VolumetricTestData object provides volumetric test data.

Access

```
materialApi.materials()[name].hyperelastic().volumetricTestData()
materialApi.materials()[name].hyperfoam().volumetricTestData()
materialApi.materials()[name].viscoelastic().volumetricTestData()
```

VolumetricTestData(...)

This method creates a VolumetricTestData object.

Path

```
materialApi.materials()[name].hyperelastic().VolumetricTestData
materialApi.materials()[name].hyperfoam().VolumetricTestData
materialApi.materials()[name].viscoelastic().VolumetricTestData
```

Prototype

```
odb_VolumetricTestData&
VolumetricTestData(const odb_SequenceSequenceDouble& table,
                   odb_Union volinf,
                   odb_Union smoothing,
                   bool temperatureDependency,
                   int dependencies);
```

Required arguments

table
An odb_SequenceSequenceDouble specifying the items described below.

Optional arguments

volinf
The string "NONE" or a Double specifying a normalized volumetric value that depends on the value of the *time* member of the *Viscoelastic* object. The default value is "NONE".

If *time*="RELAXATION_TEST_DATA", *volinf* specifies the value of the long-term, normalized volumetric modulus, $k_R(\infty)$.

If *time*="CREEP_TEST_DATA", *volinf* specifies the value of the long-term, normalized volumetric compliance, $j_K(\infty)$.

This argument is valid only for a viscoelastic material model.

smoothing
The string "NONE" or an Int specifying the value for smoothing. If *smoothing*="NONE", no smoothing is employed. The default value is "NONE".

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

Table data

For a hyperelastic or hyperfoam material model, the table data specify the following:

- Pressure, p.
- Volume ratio, J (current volume/original volume).

For a viscoelastic material model, the values depend on the value of the *time* member of the *Viscoelastic* object.

If *time*=RELAXATION_TEST_DATA, the table data specify the following:

- Normalized volumetric (bulk) modulus $k_R(t), \quad (0 \le k_R(t) \le 1)$.
- Time $t(t > 0)$.

If *time*=CREEP_TEST_DATA, the table data specify the following:

- Normalized volumetric (bulk) compliance $j_K(t), \quad (j_K(t) \ge 1)$.
- Time $t(t > 0)$.

Return value

A VolumetricTestData object.

Exceptions

None.

Members

The VolumetricTestData object has members with the same names and descriptions as the arguments to the *VolumetricTestData* method.

Corresponding analysis keywords

- **VOLUMETRIC TEST DATA*

Odb commands

The C++ ODB API commands are used to read and write data from an output database (`.odb`) file. In this chapter the `Access` and `Path` statements refer to a variable called `odb` that represents an existing Odb object.

In this section:

- *Odb object*
- *AnalyticSurface object*
- *AnalyticSurfaceSegment object*
- *BeamOrientation object*
- *FieldBulkData object*
- *FieldLocation object*
- *FieldOutput object*
- *FieldValue object*
- *HistoryOutput object*
- *HistoryPoint object*
- *HistoryRegion object*
- *JobData object*
- *OdbAssembly object*
- *OdbDatumCsys object*
- *OdbFrame object*
- *OdbInstance object*
- *OdbLoadCase object*
- *OdbMeshElement object*
- *OdbMeshNode object*
- *OdbPart object*
- *OdbPretensionSection object*
- *OdbRigidBody object*
- *OdbSequenceAnalyticSurfaceSegment object*
- *OdbSet object*
- *OdbStep object*
- *RebarOrientation object*
- *SectionCategory object*
- *SectionPoint object*
- *SectorDefinition object*
- *UserData object*
- *UserXYData object*
- *Top level commands*

Odb object

The Odb object is the in-memory representation of an output database (ODB) file.

Access

```
odb
```

Odb(...)

This method creates a new Odb object.

Path

```
Odb
```

Prototype

```
odb_Odb&
Odb(const odb_String& name,
    const odb_String& analysisTitle,
    const odb_String& description,
    const odb_String& path);
```

Required arguments

name An odb_String specifying the repository key.

Optional arguments

analysisTitle An odb_String specifying the title of the output database. The default value is an empty string.

description An odb_String specifying the description of the output database. The default value is an empty string.

path An odb_String specifying the path to the file where the new output database (`.odb`) file will be written. The default value is an empty string.

Return value

An Odb object.

Exceptions

None.

`close()`

This method closes an output database.

Prototype

```
void
close();
```

Arguments

None.

Return value

None.

Exceptions

None.

getFrame(...)

This method returns the frame at the specified time, frequency, or mode. It will not interpolate values between frames. The method is not applicable to an Odb object containing steps with different domains or to an Odb object containing a step with load case specific data.

Prototype

```
odb_Frame
getFrame(double frameValue,
         odb_Enum::odb_MatchEnum match);
```

Required arguments

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the total time or frequency.

Optional arguments

match An odb_Enum::odb_MatchEnum specifying which frame to return if there is no frame at the exact frame value. Possible values are odb_Enum::CLOSEST, odb_Enum::BEFORE, odb_Enum::AFTER, and odb_Enum::EXACT. The default value is odb_Enum::CLOSEST.

When *match*=odb_Enum::CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=odb_Enum::EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the exact frame is not found: OdbError: Frame not found.

save()

This method saves output to an output database (`.odb`) file.

Prototype

```
void
save();
```

Arguments

None.

Return value

`None.`

Exceptions

OdbError	Database save failed. The database was opened as read-only. Modification of data is not permitted.

update()

This method is used to update an Odb object in memory while an Abaqus analysis writes data to the associated output database. `update` checks if additional steps have been written to the output database since it was opened or last updated. If additional steps have been written to the output database, `update` adds them to the Odb object.

Prototype

```
bool
update();
```

Arguments

None.

Return value

A Boolean specifying whether additional steps or frames were added to the Odb object.

Exceptions

None.

hasSectorDefinition()

This method checks whether or not a valid *SectorDefinition* object, indicating a cyclic symmetry model, is present.

Prototype

```
bool
hasSectorDefinition();
```

Arguments

None.

Return value

A Boolean specifying whether a valid sector definition is available.

Exceptions

None.

Members

The Odb object has members with the same names and descriptions as the arguments to the *Odb* method.

In addition, the Odb object can have the following members:

Prototype

```
odb_String name() const;
               odb_String analysisTitle() const;
               odb_String description() const;
               odb_String path() const;
               bool isReadOnly() const;
               odb_Assembly& rootAssembly();
               odb_JobData jobData() const;
               odb_PartRepository& parts();
               odb_StepRepository& steps();
               odb_SectionCategoryRepository& sectionCategories();
               odb_SectorDefinition& sectorDefinition();
               odb_InteractionRepository& interactions();
               odb_InteractionPropertyRepository& interactionProperties();
               odb_ConstraintRepository& constraints();
```

isReadOnly	A Boolean specifying whether the output database was opened with read-only access.
rootAssembly	An *OdbAssembly* object.
jobData	A *JobData* object.
parts	A repository of *OdbPart* objects.
steps	A repository of *OdbStep* objects.
sectionCategories	A repository of *SectionCategory* objects.
sectorDefinition	A *SectorDefinition* object.
userData	A *UserData* object.

AnalyticSurface object

The AnalyticSurface object is a geometric surface that can be described with straight and/or curved line segments.

Access

```
odb.parts()[name].analyticSurface()
odb.rootAssembly().instances()[name].analyticSurface()
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.analyticSurface()
```

Members

The AnalyticSurface object has the following members:

Prototype

```
odb_String name() const;
odb_String type() const;
double filletRadius() const;
odb_SequenceSequenceFloat localCoordData() const;
odb_SequenceAnalyticSurfaceSegment segments() const;
```

name

An odb_String specifying the name of the analytic surface.

type

An odb_String specifying the type of AnalyticSurface object. Possible odb_String names are SEGMENTS, CYLINDER, and REVOLUTION.

filletRadius

A Double specifying radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

segments

An *OdbSequenceAnalyticSurfaceSegment* object specifying the profile associated with the surface.

localCoordData

An odb_SequenceSequenceFloat specifying the global coordinates of points representing the local coordinate system, if used.

AnalyticSurfaceSegment object

An individual segment of the analytic surface.

Access

```
odb.parts()[name].analyticSurface().segments()[i]
odb.rootAssembly().instances()[name].analyticSurface().segments()[i]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.analyticSurface().segments()[i]
```

Members

The AnalyticSurfaceSegment object has the following members:

Prototype

```
odb_String type() const;
odb_SequenceSequenceFloat data() const;
```

type

An *odb_String* specifying the type of AnalyticSurfaceSegment. Possible *odb_String* names are START, LINE, CIRCLE, and PARABOLA.

data

An *odb_SequenceSequenceFloat* specifying the coordinates of points representing the segment of the *AnalyticSurface* object. If *type*=odb_Enum::CIRCLE, the first row contains coordinates of the end point and the second row contains coordinates of the center point. If *type*=odb_Enum::PARABOLA, the first row contains coordinates of the middle point and the second row contains coordinates of the end point. If *type*=odb_Enum::START or *type*=odb_Enum::LINE, a single row contains coordinates of the start/end point.

BeamOrientation object

The BeamOrientation object represents the direction of the first beam section axis $\mathbf{n_1}$. Specifying the beam orientation using an additional node in the element connectivity list is not supported.

Access

```
odb.parts()[name].beamOrientations(i)
odb.rootAssembly().instances()[name].beamOrientations(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.beamOrientations(i)
```

Members

The BeamOrientation object can have the following members:

Prototype

```
odb_Set region() const;
odb_Enum::odb_OrientationMethodEnum method() const;
odb_SequenceFloat vector() const;
```

method

An odb_Enum::odb_OrientationMethodEnum specifying the orientation assignment method. Possible values are odb_Enum::N1_COSINES, odb_Enum::CSYS, and odb_Enum::VECT.

region

An *OdbSet* object specifying a region for which the beam orientation is defined.

vector

An odb_SequenceFloat specifying direction cosines of the $\mathbf{n_1}$-direction of the beam cross-section.

FieldBulkData object

The FieldBulkData object represents the entire field data for a class of elements or nodes. All elements in a class correspond to the same element type and material.

Access

```
odb.steps()[name].frames(i).fieldOutputs()[name].bulkDataBlocks(i)
```

Members

The FieldBulkData object can have the following members:

Prototype

```
odb_Enum::odb_ResultPositionEnum position() const;
               odb_Enum::odb_PrecisionEnum precision() const;
               int* elementLabels() const;
               int* nodeLabels() const;
               int* integrationPoints() const;
               odb_String baseElementType() const;
               odb_Enum::odb_ElementFaceEnum* faces() const;
               odb_Enum::odb_DataTypeEnum type() const;
               float* data() const;
               double* dataDouble() const;
               float* conjugateData() const;
               double* conjugateDataDouble() const;
               float* mises() const;
               float* localCoordSystem() const;
               double* localCoordSystemDouble() const;
               int orientationWidth() const;
               int numberOfElements() const;
               int length() const;
               int valuesPerElement() const;
               int width() const;
               const odb_Instance& instance() const;
               const odb_SectionPoint& sectionPoint() const;
               odb_SequenceString componentLabels() const;
```

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::ELEMENT_FACE, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

precision

An odb_Enum::odb_PrecisionEnum specifying the precision of the output. Possible values are:

- odb_Enum::SINGLE_PRECISION, specifying that the output is in single precision.
- odb_Enum::DOUBLE_PRECISION, specifying that the output is in double precision.

type

An odb_Enum::odb_DataTypeEnum specifying the output type. Possible values are odb_Enum::SCALAR, odb_Enum::VECTOR, odb_Enum::TENSOR_3D_FULL, odb_Enum::TENSOR_3D_PLANAR, odb_Enum::TENSOR_3D_SURFACE, odb_Enum::TENSOR_2D_PLANAR, and odb_Enum::TENSOR_2D_SURFACE.

orientationWidth

An Int specifying the number of direction cosines necessary to specify the local coordinate system at each output location. You use *orientationWidth* to read the orientation data from the *localCoordSystem*.

numberOfElements

An Int specifying the number of elements in the current block of data.

length

An Int specifying the number of output locations in the current block of data.

valuesPerElement

An Int specifying the number of values per element in the current block of data. If *position*=odb_Enum::ELEMENT_NODAL, *valuesPerElement* is the number of nodes per element for all elements in the current block of data.

width

An Int specifying the number of components at each output location.

baseElementType

An odb_String specifying the element type corresponding to the current block of data.

instance

An *OdbInstance* object specifying the part to which the labels belong.

sectionPoint

A *SectionPoint* object specifying the section point number of the current block of data.

elementLabels

A pointer to an array of Ints specifying the element labels of the elements in the block. *elementLabels* is valid only if *position* =odb_Enum::INTEGRATION_POINT , odb_Enum::CENTROID , odb_Enum::ELEMENT_NODAL , or odb_Enum::ELEMENT_FACE. If *position* =odb_Enum::NODAL , *elementLabels* returns a NULL pointer.

nodeLabels

A pointer to an array of Ints specifying the node labels of the nodes in the block. *nodeLabels* is valid only if *position* =odb_Enum::NODAL or odb_Enum::ELEMENT_NODAL. If *position* =odb_Enum::INTEGRATION_POINT , odb_Enum::CENTROID , odb_Enum::ELEMENT_NODAL , or odb_Enum::ELEMENT_FACE , *nodeLabels* returns a NULL pointer.

integrationPoints

A pointer to an array of Ints specifying the integration points in the elements in the block. *integrationPoint* is available only if *position* =odb_Enum::INTEGRATION_POINT.

faces

A pointer to an array of odb_Enum::odb_ElementFaceEnum enumerations specifying the faces of the elements in the block. *faces* is available only if *position* =odb_Enum::ELEMENT_FACE.

data

A pointer to an array of Floats specifying the field's data in the order described by *type*. If *type* = odb_Enum::TENSOR or odb_Enum::VECTOR, *data* is an array containing the components for each element or node in the block. If the underlying data are in double precision, an exception will be thrown.

dataDouble

A pointer to an array of Doubles specifying the field's data in the order described by *type*. If *type* = odb_Enum::TENSOR or odb_Enum::VECTOR, *data* is an array containing the components for each element or node in the block. If the underlying data are in single precision, an exception will be thrown.

conjugateData

A pointer to an array of Floats specifying the imaginary portion of a complex result. The order of the Floats is described by*type*. If *type* = odb_Enum::TENSOR or odb_Enum::VECTOR, *conjugateData* is an array containing the imaginary part of the components for each element or node in the block. If the underlying data are in double precision, an exception will be thrown.

conjugateDataDouble

A pointer to an array of Doubles specifying the imaginary portion of a complex result. The order of the Doubles is described by *type*. If *type* = odb_Enum::TENSOR or odb_Enum::VECTOR, *conjugateData* is an array containing the imaginary part of the components for each element or node in the block. If the underlying data are in single precision, an exception will be thrown.

mises

A pointer to an array of Floats specifying the calculated von Mises stress at each output location in the block of element data, or NULL. If *validInvariants* includes odb_Enum::MISES, *mises* returns an array pointer. If *validInvariants* does not include odb_Enum::MISES, *mises* returns a NULL pointer. Conjugate data will be ignored in invariant calculation.

localCoordSystem

A pointer to an array of Floats specifying the quaternion representing the local coordinate system (the rotation from global to local) at each output location. The quaternion is returned

in the form $q = (\mathbf{q}, q_0)$, which is the reverse of that shown in *Rotation variables*. *localCoordSystem* is available for odb_Enum::TENSOR data written in a local coordinate system. It is also available for odb_Enum::VECTOR data for connector element outputs.

For connector element outputs the quaternion form is $q = (q_0, \mathbf{q})$, which represents the rotation from local to global. If the underlying data are in double precision, an exception will be thrown.

localCoordSystemDouble

A pointer to an array of Doubles specifying the quaternion representing the local coordinate system (the rotation from global to local) at each output location. The quaternion is returned

in the form $q = (\mathbf{q}, q_0)$, which is the reverse of that shown in *Rotation variables*. *localCoordSystemDouble* is available for odb_Enum::TENSOR data written in a local coordinate system. It is also available for odb_Enum::VECTOR data for connector element

outputs. For connector element outputs the quaternion form is $q = (q_0, \mathbf{q})$, which represents the rotation from local to global. If the underlying data are in single precision, an exception will be thrown.

componentLabels

An odb_SequenceString specifying the component labels.

FieldLocation object

The FieldLocation object specifies locations for which data are available in the field. For example, a displacement field will have a FieldLocation object with a *position* member value of NODAL. The FieldLocation object has no constructor; it is created automatically as an element of the *location* member of a *FieldOutput* object by the `addData` method of a *FieldOutput* object.

Access

```
odb.steps()[name].frames(i).fieldOutputs()[name].locations(i)
```

Members

The FieldLocation object can have the following members:

Prototype

```
odb_Enum::odb_ResultPositionEnum position() const;
const odb_SequenceSectionPoint& sectionPoint() const;
const odb_SectionPoint& sectionPoint(int index) const;
```

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output in the element. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::ELEMENT_FACE.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

sectionPoint

A sequence of *SectionPoint* objects.

FieldOutput object

A FieldOutput object contains field data for a specific output variable.

Access

```
odb.steps()[name].frames(i).fieldOutputs()[name]
```

FieldOutput(...)

This method creates a FieldOutput object.

Path

```
odb.steps()[name].frames(i).FieldOutput
```

Prototype

```
odb_FieldOutput& FieldOutput(const odb_String& name,
const odb_String& description, odb_Enum::odb_DataTypeEnum type, const
odb_SequenceString& componentLabels, const odb_SequenceInvariant&
validInvariants, bool isEngineeringTensor);
```

Required arguments

name
An odb_String specifying the output variable name.

description
An odb_String specifying the output variable. Colon (:) should not be used as a part of the field output description.

type
An odb_Enum::odb_DataTypeEnum specifying the output type. Possible values are odb_Enum::SCALAR, odb_Enum::VECTOR, odb_Enum::TENSOR_3D_FULL, odb_Enum::TENSOR_3D_PLANAR, odb_Enum::TENSOR_3D_SURFACE, odb_Enum::TENSOR_2D_PLANAR, and odb_Enum::TENSOR_2D_SURFACE.

Optional arguments

componentLabels
An odb_SequenceString specifying the labels for each component of the value. The length of the sequence must match the type. If *type*=odb_Enum::TENSOR, the default value is *name* with the suffixes ('11', '22', '33', '12', '13', '23'). If *type*=odb_Enum::VECTOR, the default value is *name* with the suffixes ('1', '2', '3'). If *type*=odb_Enum::SCALAR, the default value is an empty sequence.

validInvariants
An odb_SequenceInvariant specifying which invariants should be calculated for this field. An empty sequence indicates that no invariants are valid for this field. Possible values are:

- odb_Enum::MAGNITUDE
- odb_Enum::MISES
- odb_Enum::TRESCA
- odb_Enum::PRESS

- odb_Enum::INV3
- odb_Enum::MAX_PRINCIPAL
- odb_Enum::MID_PRINCIPAL
- odb_Enum::MIN_PRINCIPAL
- odb_Enum::MAX_INPLANE_PRINCIPAL
- odb_Enum::MIN_INPLANE_PRINCIPAL
- odb_Enum::OUTOFPLANE_PRINCIPAL

The default value is an empty sequence.

isEngineeringTensor A Boolean specifying whether the field is an engineering tensor or not. Setting isEngineeringTensor to true makes a tensor field behave as a strain like quantity where the off-diagonal components of tensor are halved for invariants computation. This parameter applies only to tensor field outputs. The default value is false.

Return value

A FieldOutput object.

Exceptions

None.

FieldOutput(...)

This method creates a FieldOutput object from an existing FieldOutput object.

Path

```
odb.steps()[name].frames(i).FieldOutput
```

Prototype

```
odb_FieldOutput& FieldOutput(const odb_FieldOutput&
field, const odb_String& name, const odb_String&
description);
```

Required arguments

field A FieldOutput object.

Optional arguments

name A String specifying the name of the FieldOutput object.

description An odb_String specifying the output variable. Colon (:) should not be used as a part of the field output description.

Return value

A FieldOutput object.

Exceptions

None.

VectorOutput(...)

This method creates a FieldOutput object.

Path

odb.steps()[*name*].frames(*i*).VectorOutput

Prototype

```
odb_FieldOutput& VectorOutput(const odb_String& name,
const odb_String& description, int width);
```

Required arguments

name	An odb_String specifying the output variable name.
description	An odb_String specifying the output variable. Colon (:) should not be used as a part of the field output description.
width	An Int specifying the width of the vector.

Optional arguments

None.

Return value

A FieldOutput object.

Exceptions

None.

MatrixOutput(...)

This method creates a FieldOutput object.

Path

odb.steps()[*name*].frames(*i*).MatrixOutput

Prototype

```
odb_FieldOutput& MatrixOutput(const odb_String& name,
const odb_String& description, int rows, int cols);
```

Required arguments

name	An odb_String specifying the output variable name.
description	An odb_String specifying the output variable. Colon (:) should not be used as a part of the field output description.
rows	An Int specifying the number of rows in the matrix.
cols	An Int specifying the number of columns in the matrix.

Optional arguments

None.

Return value

A FieldOutput object.

Exceptions

None.

SymmetricMatrixOutput(...)

This method creates a FieldOutput object.

Path

odb.steps()[*name*].frames(*i*).SymmetricMatrixOutput

Prototype

```
odb_FieldOutput& SymmetricMatrixOutput(const odb_String&
name, const odb_String& description, int dim);
```

Required arguments

name	An odb_String specifying the output variable name.
description	An odb_String specifying the output variable. Colon (:) should not be used as a part of the field output description.
dim	An Int specifying the dimension of the symmetric matrix.

Optional arguments

None.

Return value

A FieldOutput object.

Exceptions

None.

addData(...)

This method adds data to a FieldOutput object.

Prototype

```
void
addData(odb_Enum::odb_ResultPositionEnum position, const odb_Instance&
instance, const odb_SequenceInt& labels, const
odb_SequenceSequenceFloat& data, const odb_SectionPoint& sectionPoint,
const odb_SequenceSequenceFloat& localCoordSystem, const
odb_SequenceSequenceFloat& conjugateData);
```

Required arguments

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

instance

An *OdbInstance* object specifying the namespace for labels.

labels

An odb_SequenceInt specifying the labels of the nodes or elements where the values in *data* are located. For better performance, the node or element labels are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data

An odb_SequenceSequenceFloat specifying the data values for the specified *position*, *instance*, and *labels*. The values must be given in the correct order. Element nodal data follow the order of nodal connectivity defined in the Abaqus documentation. Integration point data follow the order defined in the Abaqus documentation. Section point data for beams and shells follow the convention given in the Abaqus documentation. For more information, see the *Abaqus Elements Guide*. These data create *FieldValue* objects internally.

Optional arguments

sectionPoint

A *SectionPoint* object specifying the location in the section. Although *sectionPoint* is an optional argument to the `addData` method, omitting the argument does have consequences for visualization. If you omit the argument when you are writing field output data for a shell or a beam, you cannot subsequently select the section point to display when you are displaying the field output data using the Visualization module.

localCoordSystem An odb_SequenceSequenceFloat specifying the 3 × 3 matrix of direction cosines of the local coordinate system. This argument is available only for fields with type=odb_Enum::TENSOR or odb_Enum::VECTOR.

User-supplied values of localCoordSystem are transposed before storing in the database.

conjugateData An odb_SequenceSequenceFloat specifying the imaginary data values for the specified position, instance, and labels. You must provide these data when you add complex fields to the output database. The order of the values follows the conventions defined in the *Abaqus Elements Guide*.

Return value
`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds the data from a field created using the `getSubset` method and mathematical operators to the database. The user must create a field to contain the new data and then use the `addData` method to assign the data from the fields.

Prototype
```
void
addData(const odb_FieldOutput& field);
```

Required arguments

field A FieldOutput object specifying the data to add.

Optional arguments

None.

Return value
`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds data to a FieldOutput object.

Prototype

```
void
addData(odb_Enum::odb_ResultPositionEnum position, const odb_Set& set,
const odb_SequenceSequenceFloat& data, const odb_SectionPoint&
sectionPoint, const odb_SequenceSequenceFloat&
conjugateData);
```

Required arguments

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.
- odb_Enum::ELEMENT_FACE_INTEGRATION_POINT, specifying the values calculated at the element face integration points.
- odb_Enum::SURFACE_INTEGRATION_POINT, specifying the values calculated at the surface integration points. Selecting this value prompts the Visualization module to calculate the sum of the values at the ELEMENT_FACE_INTEGRATION_POINT position from multiple surfaces.

set

An *OdbSet* object specifying the instance-level set defining the region for `addData`. The set must be defined in the same output database as the output database into which the new field output data is being written. For better performance, the node or element labels in the set are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data

An odb_SequenceSequenceFloat specifying the data values for the specified position and labels in the set. Each row of data provides the value at one unique position. The width of each row should match the number of required components for the data. The values must be given in the order that matches the ordering of labels in the set.

The order of the element nodal data, integration point data, and section point data for beams and shells follows the conventions defined in the *Abaqus Elements Guide*.

Optional arguments

sectionPoint

A *SectionPoint* object specifying the location in the section. Although *sectionPoint* is an optional argument to the`addData` method, omitting the argument does have consequences for visualization. If you omit the argument when you are writing field output data for a shell or a beam, you cannot subsequently select the section point to display when you are displaying the field output data using the Visualization module.

conjugateData

An odb_SequenceSequenceFloat specifying the imaginary data values for the specified position, instance, and labels. You must provide this data when you add complex fields to the output database. The order of the values follows the conventions defined in the *Abaqus Elements Guide*.

Return value

None.

Exceptions

If you specify an odbSet containing entities from multiple instances:	odbException: Entities from multiple instances present in set.
The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds data to a FieldOutput object.

Prototype

```
 void
addData(odb_Enum::odb_ResultPositionEnum position, const odb_Instance&
instance, const odb_SequenceInt& labels, const
odb_SequenceSequenceFloat& data, const odb_SectionPoint& sectionPoint,
const odb_SequenceSequenceSequenceFloat& localCoordSystem, const
odb_SequenceSequenceFloat& conjugateData);
```

Required arguments

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

instance

An *OdbInstance* object specifying the namespace for labels.

labels

An odb_SequenceInt specifying the labels of the nodes or elements where the values in *data* are located. For better performance, the node or element labels are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data

An odb_SequenceSequenceFloat specifying the data values for the specified *position*, *instance*, and *labels*. The values must be given in the correct order. Element nodal data follow the order of nodal connectivity defined in the Abaqus documentation. Integration point data follow the order defined in the Abaqus documentation. Section point data for beams and shells follow the convention given in the Abaqus documentation. For more information, see the *Abaqus Elements Guide*. These data create *FieldValue* objects internally.

Optional arguments

sectionPoint

A *SectionPoint* object specifying the location in the section. Although *sectionPoint* is an optional argument to the `addData` method, omitting the argument does have consequences for visualization. If you omit the argument when you are writing field output data for a shell or a beam, you cannot subsequently select the section point to display when you are displaying the field output data using the Visualization module.

localCoordSystem

An odb_SequenceSequenceSequenceFloat specifying the direction cosines of the local coordinates systems, where the sequence is the same length as *data*. When specified this way a different local coordinate system applies to each data value.

User-supplied values of localCoordSystem are transposed before storing in the database.

conjugateData

An odb_SequenceSequenceFloat specifying the imaginary data values for the specified *position*, *instance*, and *labels*. You must provide this data when you add complex fields to the output database. The order of the values follows the conventions defined in the *Abaqus Elements Guide*.

Return value

`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`addData(...)`

This method adds double precision data to a FieldOutput object.

Prototype

```
void
addData(odb_Enum::odb_ResultPositionEnum position, const odb_Instance&
instance, const odb_SequenceInt& labels, const
odb_SequenceSequenceDouble& data, const
odb_SequenceSequenceSequenceDouble& localCoordSystem, const
odb_SequenceSequenceDouble& conjugateData);
```

Required arguments

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output. Only odb_Enum::NODAL, specifying the values calculated at the nodes, is supported.

instance

An *OdbInstance* object specifying the namespace for labels.

labels

An odb_SequenceInt specifying the labels of the nodes or elements where the values in *data* are located. For better performance, the node or element labels are preferred to be sorted in ascending order and must be specified in the same order as the values provided for the *data* argument.

data
An odb_SequenceSequenceDouble specifying the data values for the specified *position*, *instance*, and *labels*. The values must be given in the correct order. These data create *FieldValue* objects internally.

Optional arguments

localCoordSystem
An odb_SequenceSequenceSequenceDouble specifying the direction cosines of the local coordinates systems, where the sequence is the same length as *data*. When specified this way a different local coordinate system applies to each data value.

User-supplied values of localCoordSystem are transposed before storing in the database.

conjugateData
An odb_SequenceSequenceDouble specifying the imaginary data values for the specified *position*, *instance*, and *labels*. You must provide this data when you add complex fields to the output database. The order of the values follows the conventions defined in the *Abaqus Elements Guide*.

Return value

`None.`

Exceptions

The `addData` method throws many exceptions of type odbException. For example, if the local coordinate system is specified for scalar data:	odbException: Transformation not allowed for scalar data.

`getScalarField(...)`

This method generates a scalar field containing the extracted component or calculated invariant values. The new field will hold values for the same nodes or elements as the parent field. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Prototype

```
odb_FieldOutput getScalarField(odb_Enum::odb_InvariantEnum
invariant);
```

Required arguments

invariant
An odb_Enum::odb_InvariantEnum specifying the invariant. Possible values areodb_Enum::MAGNITUDE, odb_Enum::MISES, odb_Enum::TRESCA, odb_Enum::PRESS, odb_Enum::INV3, odb_Enum::MAX_PRINCIPAL, odb_Enum::MID_PRINCIPAL, odb_Enum::MIN_PRINCIPAL, odb_Enum::MAX_INPLANE_PRINCIPAL, odb_Enum::MIN_INPLANE_PRINCIPAL, and odb_Enum::OUTOFPLANE_PRINCIPAL.

Optional arguments

None.

Return value

AFieldOutput object.

Exceptions

None.

getScalarField(...)

This method generates a scalar field containing the extracted component or calculated invariant values. The new field will hold values for the same nodes or elements as the parent field. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Prototype

```
odb_FieldOutput getScalarField(const odb_String&
componentLabel);
```

Required arguments

componentLabel A String specifying the component label, such as “S11”.

Optional arguments

None.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(odb_Enum::odb_ResultPositionEnum
position, bool readOnly);
```

Required arguments

None.

Optional arguments

position An odb_Enum::odb_ResultPositionEnum specifying the position of the output in the element. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.

- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

If the requested field values are not found in the output database at the specified odb_Enum::ELEMENT_NODAL or odb_Enum::CENTROID positions, they are extrapolated from the field data at the odb_Enum::INTEGRATION_POINT position.

readOnly

A Boolean specifying whether the extrapolated data returned by this call is written to the output database. The default value is false.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_Set&
region);
```

Required arguments

None.

Optional arguments

region

An *OdbSet* specifying the region for which to extract values. For better performance, the node or element labels in the sets are preferred to be sorted in ascending order.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_SequenceSequenceFloat&
localCoordSystem);
```

Required arguments

None.

Optional arguments

localCoordSystem An odb_SequenceSequenceFloat specifying the 3 × 3 matrix of direction cosines. Field values associated with the supplied coordinate system will be extracted.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_SectionPoint&
sectionPoint);
```

Required arguments

None.

Optional arguments

sectionPoint A *SectionPoint* object.

Return value

A FieldOutput object.

Exceptions

None.

getSubset(...)

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_FieldLocation&
location);
```

Required arguments

None.

Optional arguments

location A *FieldLocation* object.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_Element&
region);
```

Required arguments

None.

Optional arguments

region An *OdbMeshElement* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_Node&
region);
```

Required arguments

None.

Optional arguments

region An *OdbMeshNode* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_Instance&
region);
```

Required arguments

None.

Optional arguments

region An *OdbInstance* specifying the region for which to extract values.

Return value

A FieldOutput object.

Exceptions

None.

`getSubset(...)`

A FieldOutput object with a subset of the field values.

Prototype

```
odb_FieldOutput getSubset(const odb_String&
elementType);
```

Required arguments

None.

Optional arguments

elementType A String specifying the element type for which to extract values. The string must correspond to a valid Abaqus element type.

Return value

A FieldOutput object.

Exceptions

None.

getTransformedField(...)

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *OdbDatumCsys object*. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Prototype

```
odb_FieldOutput getTransformedField(const odb_DatumCsys&
datumCsys, int projected22Axis, double projectionTol);
```

Required arguments

datumCsys A valid *OdbDatumCsys object* designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

projected22Axis An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.

projectionTol A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes.	odbException: Cannot apply transformation to field containing assembly level nodes.

getTransformedField(...)

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *OdbDatumCsys object*. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Prototype

```
odb_FieldOutput getTransformedField(const odb_DatumCsys&
datumCsys, const odb_FieldOutput& deformationField, int projected22Axis,
double projectionTol);
```

Required arguments

datumCsys A valid *OdbDatumCsys object* designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

deformationField A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.

projected22Axis An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.

projectionTol A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes. odbException: Cannot apply transformation to field containing assembly level nodes.

`getTransformedField(...)`

This method generates a new vector or tensor field containing the transformed component values of the parent field. The new field will hold values for the same nodes or elements as the parent field. Results will be transformed based on the orientations specified by the input *OdbDatumCsys object*. Abaqus will perform this operation on only the real part of the FieldOutput object. The operation is not performed on the conjugate data (the imaginary portion of a complex result).

Prototype

```
odb_FieldOutput getTransformedField(const odb_DatumCsys&
datumCsys, const odb_FieldOutput& deformationField, const
odb_FieldOutput& rotationField, int projected22Axis, double
projectionTol);
```

Required arguments

datumCsys A valid *OdbDatumCsys object* designating the coordinate system. Valid systems can be fixed or positioned with respect to nodes on the model and can be cartesian, cylindrical, or spherical.

Optional arguments

deformationField	A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.
rotationField	A FieldOutput object specifying the nodal rotational displacement vectors required by moving coordinate systems that follow a 6-dof node, to determine instantaneous configurations.
projected22Axis	An Int specifying which axis of the coordinate system will be projected as the second component for local result orientations. Valid values are 1, 2, or 3; the default value is 2.
projectionTol	A Double specifying the minimum allowable angle (radians) between the specified projection axis and the element normal. The next axis will be used for projection if this tolerance test fails.

Return value

A FieldOutput object.

Exceptions

The `getTransformedField` method throws an exception if the field contains any assembly level nodes.	odbException: Cannot apply transformation to field containing assembly level nodes.

`getConnectorFieldXformedToNodeA(...)`

This method generates a new vector field containing the transformed component values of the parent connector field to the node A coordinate system. The new field will hold values for the same connector elements as the parent field. Some connection types such as Axial, Link, Slip Ring, and Radial Thrust require that the deformationField be specified.

Prototype

```
odb_FieldOutput getConnectorFieldXformedToNodeA(const
odb_FieldOutput& deformationField) const;
```

Required arguments

None.

Optional arguments

deformationField	A FieldOutput object specifying the nodal displacement vectors required by moving coordinate systems to determine instantaneous configurations.

Return value

A FieldOutput object.

Exceptions

The `getConnectorFieldXformedToNodeA` method throws an exception if the field requires a deformationField and the argument is not supplied.	odbException: Deformation field is required for transforming this connector field.

`setComponentLabels(...)`

This method sets the component labels for the FieldOutput object.

Prototype

```
void
setComponentLabels(const odb_SequenceString&
componentLabels);
```

Required arguments

componentLabels An odb_SequenceString specifying the labels for each component of the value. The length of the sequence must match the type. If *type*=odb_Enum::TENSOR, the default value is *name* with the suffixes ('11', '22', '33', '12', '13', '23'). If *type*=odb_Enum::VECTOR, the default value is *name* with the suffixes ('1', '2', '3'). If *type*=odb_Enum::SCALAR, the default value is an empty sequence.

Return value

`None.`

Exceptions

None.

`setDataType(...)`

This method sets the data type of a FieldOutput object.

Prototype

```
void
setDataType(odb_Enum::odb_DataTypeEnum type);
```

Required arguments

type An odb_Enum::odb_DataTypeEnum specifying the output type. Possible values are odb_Enum::SCALAR, odb_Enum::VECTOR, odb_Enum::TENSOR_3D_FULL, odb_Enum::TENSOR_3D_PLANAR, odb_Enum::TENSOR_3D_SURFACE, odb_Enum::TENSOR_2D_PLANAR, and odb_Enum::TENSOR_2D_SURFACE.

Return value

`None.`

Exceptions

None.

setValidInvariants(...)

This method sets the invariants valid for the FieldOutput object.

Prototype

```
void
setValidInvariants(const odb_SequenceInvariant&
validInvariants);
```

Required arguments

validInvariants An odb_SequenceInvariant specifying which invariants should be calculated for this field. An empty sequence indicates that no invariants are valid for this field. Possible values are:

- odb_Enum::MAGNITUDE
- odb_Enum::MISES
- odb_Enum::TRESCA
- odb_Enum::PRESS
- odb_Enum::INV3
- odb_Enum::MAX_PRINCIPAL
- odb_Enum::MID_PRINCIPAL
- odb_Enum::MIN_PRINCIPAL
- odb_Enum::MAX_INPLANE_PRINCIPAL
- odb_Enum::MIN_INPLANE_PRINCIPAL
- odb_Enum::OUTOFPLANE_PRINCIPAL

The default value is an empty sequence.

Return value

None.

Exceptions

None.

Members

The FieldOutput object has members with the same names and descriptions as the arguments to the *FieldOutput* method.

In addition, the FieldOutput object can have the following members:

Prototype

```
odb_String name() const; odb_String description() const;
odb_Enum::odb_DataTypeEnum type() const; odb_Enum::odb_DataSubtypeEnum
```

```
subtype() const; int dim() const; int dim2() const; bool isComplex() const;
odb_SequenceInvariant validInvariants() const; odb_SequenceString
componentLabels() const; odb_SequenceString baseElementTypes() const; const
odb_SequenceFieldLocation& locations() const; odb_FieldLocation
locations(int index) const; const odb_FieldValue values(int i);
odb_SequenceFieldValue values(); const odb_FieldBulkData&
bulkDataBlocks(int i); odb_SequenceFieldBulkData&
bulkDataBlocks();
```

dim

An Int specifying the dimension of vector or the first dimension (number of rows) of matrix.

dim2

An Int specifying the second dimension (number of columns) of matrix.

isComplex

A Boolean specifying whether the data are complex.

locations

A sequence of *FieldLocation* objects.

values

A sequence of *FieldValue* objects specifying the order of the objects in the array is determined by the Abaqus Scripting Interface; see the *data* argument to the `addData` method for a description of the order.

bulkDataBlocks

A sequence of *FieldBulkData* objects.

FieldValue object

The FieldValue object represents the field data at a point. The FieldValue object has no constructor; it is created by the *Odb* object when data are added to the *FieldOutput* object using the `addData` method. For faster, bulk-data access, see *Using bulk data access to an output database*.

Access

```
odb.steps()[name].frames(i).fieldOutputs()[name].values(i)
```

Members

The FieldValue object has the following members:

Prototype

```
odb_Enum::odb_ResultPositionEnum position() const;
const odb_Instance& instance() const;
int elementLabel() const;
int nodeLabel() const;
int integrationPoint() const;
odb_Enum::odb_ElementFaceEnum face() const;
odb_SectionPoint sectionPoint() const;
odb_Enum::odb_PrecisionEnum precision() const;
odb_Enum::odb_DataTypeEnum type() const;
const float* data(int& numVal) const;
odb_SequenceFloat data() const;
const float* conjugateData(int& numVal) const;
odb_SequenceFloat conjugateData() const;
const double* dataDouble(int& numVal) const;
odb_SequenceDouble dataDouble() const;
const double* conjugateDataDouble(int& numVal) const;
odb_SequenceDouble conjugateDataDouble() const;
odb_SequenceSequenceFloat localCoordSystem() const;
odb_SequenceSequenceDouble localCoordSystemDouble() const;
float magnitude() const;
float mises() const;
float tresca() const;
float press() const;
float inv3() const;
float maxPrincipal() const;
float midPrincipal() const;
float minPrincipal() const;
float maxInPlanePrincipal() const;
float minInPlanePrincipal() const;
float outOfPlanePrincipal() const;
```

position

An odb_Enum::odb_ResultPositionEnum specifying the position of the output in the element. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.

- odb_Enum::ELEMENT_FACE, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element.
- odb_Enum::CENTROID, specifying the value at the centroid obtained by extrapolating results calculated at the integration points.

precision

An odb_Enum::odb_PrecisionEnum specifying the precision of the output in the element. Possible values are:

- odb_Enum::SINGLE_PRECISION, specifying that the output values are in single precision.
- odb_Enum::DOUBLE_PRECISION, specifying that the output values are in double precision.

elementLabel

An Int specifying the element label of the element containing the location. *elementLabel* is available only if *position*=odb_Enum::INTEGRATION_POINT, odb_Enum::CENTROID, odb_Enum::ELEMENT_NODAL, or odb_Enum::ELEMENT_FACE.

nodeLabel

An Int specifying the node label of the node containing the location. *nodelabel* is available only if *position*=odb_Enum::ELEMENT_NODAL or odb_Enum::NODAL.

integrationPoint

An Int specifying the integration point in the element. *integrationPoint* is available only if *position*=odb_Enum::INTEGRATION_POINT.

face

An odb_Enum::odb_ElementFaceEnum specifying the face of the element. *face* is available only if *position*=odb_Enum::ELEMENT_FACE.

type

An odb_Enum::odb_DataTypeEnum specifying the output type. Possible values are odb_Enum::SCALAR, odb_Enum::VECTOR, odb_Enum::TENSOR_3D_FULL, odb_Enum::TENSOR_3D_PLANAR, odb_Enum::TENSOR_3D_SURFACE, odb_Enum::TENSOR_2D_PLANAR, and odb_Enum::TENSOR_2D_SURFACE.

magnitude

A Float specifying the length or magnitude of the vector. *magnitude* is valid only when *type*=odb_Enum::VECTOR.

mises

A Float specifying the calculated von Mises stress. The value is valid only when the *validInvariants* member includes odb_Enum::MISES; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

tresca

A Float specifying the calculated Tresca stress. The value is valid only when the *validInvariants* member includes odb_Enum::TRESCA; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

press

A Float specifying the calculated pressure stress. The value is valid only when the *validInvariants* member includes odb_Enum::PRESS; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

inv3

A Float specifying the calculated third stress invariant. The value is valid only when the *validInvariants* member includes odb_Enum::INV3; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

maxPrincipal

A Float specifying the calculated maximum principal stress. The value is valid only when the *validInvariants* member includes odb_Enum::MAX_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

midPrincipal

A Float specifying the calculated intermediate principal stress. The value is valid only when the *validInvariants* member includes odb_Enum::MID_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

minPrincipal

A Float specifying the minimum principal stress. The value is valid only when the *validInvariants* member includes odb_Enum::MIN_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

maxInPlanePrincipal

A Float specifying the maximum principal in-plane stress. The value is valid only when the *validInvariants* member includes odb_Enum::MAX_INPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

minInPlanePrincipal

A Float specifying the calculated minimum principal in-plane stress. The value is valid only when the *validInvariants* member includes odb_Enum::MIN_INPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

outOfPlanePrincipal

A Float specifying the calculated principal out-of-plane stress. The value is valid only when the *validInvariants* member includes odb_Enum::OUTOFPLANE_PRINCIPAL; otherwise, the value is indeterminate. Conjugate data will be ignored in invariant calculation.

instance

An *OdbInstance* object specifying the part to which the labels belong.

sectionPoint

A *SectionPoint* object.

localCoordSystem

An odb_SequenceSequenceFloat specifying the 3 × 3 matrix of Floats specifying the direction cosines of the local coordinate system (the rotation from global to local). Each sequence represents a row in the direction cosine matrix. *localCoordSystem* is available for odb_Enum::TENSOR data written in a local coordinate system. It is also available for odb_Enum::VECTOR data for connector element outputs. For connector element outputs, the rotation is from local to global. If the underlying data are in double precision, an exception will be thrown.

localCoordSystemDouble

An odb_SequenceSequenceDouble specifying the 3 × 3 matrix of Doubles specifying the direction cosines of the local coordinate system (the rotation from global to local). Each sequence represents a row in the direction cosine matrix. *localCoordSystemDouble* is available for odb_Enum::TENSOR data written in a local coordinate system. It is also available for odb_Enum::VECTOR data for connector element outputs. For connector element outputs the rotation is from local to global. If the underlying data are in single precision, an exception will be thrown.

data

A pointer to an array of Floats specifying data in the form described by *type*. If *type*=odb_Enum::TENSOR or odb_Enum::VECTOR, *data* is a sequence containing the components. If the underlying data are in double precision an exception will be thrown.

dataDouble

A pointer to an array of Doubles specifying data in the form described by *type*. If *type*=odb_Enum::TENSOR or odb_Enum::VECTOR, *data* is a sequence containing the components. If the underlying data are in single precision, an exception will be thrown.

conjugateData

A pointer to an array of Floats specifying data in the form described by *type*. If *type*=odb_Enum::TENSOR or odb_Enum::VECTOR, *conjugateData* is a sequence containing the components. If the underlying data are in double precision, an exception will be thrown.

conjugateDataDouble

A pointer to an array of Doubles specifying data in the form described by *type*. If *type*=odb_Enum::TENSOR or odb_Enum::VECTOR, *conjugateData* is a sequence containing the components. If the underlying data are in single precision, an exception will be thrown.

HistoryOutput object

The HistoryOutput object contains the history output at a point for the specified variable.

Access

```
odb.steps()[name].historyRegions()[name].historyOutputs()[name]
```

HistoryOutput(...)

This method creates a HistoryOutput object.

Path

```
odb.steps()[name].historyRegions()[name].HistoryOutput
```

Prototype

```
odb_HistoryOutput&
HistoryOutput(const odb_String& name,
              const odb_String& description,
              odb_Enum::odb_DataTypeEnum type,
              const odb_SequenceInvariant& validInvariants);
```

Required arguments

name
An odb_String specifying the output variable name.

description
An odb_String specifying the output variable.

type
An odb_Enum::odb_DataTypeEnum specifying the output type. Only odb_Enum::SCALAR is currently supported.

Optional arguments

validInvariants
An odb_SequenceInvariant specifying which invariants should be calculated for this field. Possible values are odb_Enum::MAGNITUDE, odb_Enum::MISES, odb_Enum::TRESCA, odb_Enum::PRESS, odb_Enum::INV3, odb_Enum::MAX_PRINCIPAL, odb_Enum::MID_PRINCIPAL, and odb_Enum::MIN_PRINCIPAL. The default value is an empty sequence.

Return value

A HistoryOutput object.

Exceptions

None.

addData(...)

This method adds data to the *data* member of the HistoryOutput object.

Prototype

```
void
addData(double frame,
        double value);
```

Required arguments

frame A Double specifying the frame value. *frame* can be specified in step time, frequency, or mode number.

value A Double specifying the value of the variable at the frame value specified in *frame*.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

addData(...)

This method adds data to the *data* member of the HistoryOutput object.

Prototype

```
void
addData(const odb_SequenceFloat& frame,
        const odb_SequenceFloat& value);
```

Required arguments

frame An odb_SequenceFloat specifying the frame values. *frame* can be specified in step time, frequency, or mode number.

value An odb_SequenceFloat specifying the value of the variable at the frame values specified in *frame*.

Optional arguments

None.

Return value

`None.`

Exceptions

If the length of *frame* is not the same as the length of *value* a ValueError is raised.

addData(...)

This method adds data to the *data* member of the HistoryOutput object.

Prototype

```
void
addData(const odb_SequenceSequenceFloat& data);
```

Required arguments

data

An odb_SequenceSequenceFloat specifying the pairs (*frameValue*, *value*) where *frameValue* is either time, frequency, or mode and *value* is the value of the specified variable at *frameValue*. (This value depends on the type of the variable.)

Optional arguments

None.

Return value

None.

Exceptions

None.

Members

The HistoryOutput object has members with the same names and descriptions as the arguments to the *HistoryOutput* method.

In addition, the HistoryOutput object has the following members:

Prototype

```
odb_String name() const;
odb_String description() const;
odb_SequenceSequenceFloat data() const;
odb_SequenceSequenceFloat conjugateData() const;
```

data

An odb_SequenceSequenceFloat specifying the pairs (*frameValue*, *value*) where *frameValue* is either time, frequency, or mode and *value* is the value of the specified variable at *frameValue*. (This value depends on the type of the variable.)

conjugateData

An odb_SequenceSequenceFloat specifying the imaginary portion of a specified complex variable at each frame value (time, frequency, or mode). The pairs have the form (*frameValue*, *value*).

HistoryPoint object

The HistoryPoint object specifies the point at which history data will be collected. The HistoryPoint object is a temporary object used as an argument to the `HistoryRegion` method.

Access

```
odb.steps()[name].historyRegions()[name].point()
```

HistoryPoint(...)

This method creates a HistoryPoint object for a node.

Path

HistoryPoint

Prototype

```
odb_HistoryPoint&
HistoryPoint(const odb_Node& node);
```

Required arguments

node An *OdbMeshNode* object specifying the node for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for an element.

Path

```
odb.steps()[name].historyRegions()[name].HistoryPoint
```

Prototype

```
odb_HistoryPoint&
HistoryPoint(const odb_Element& element,
             int ipNumber,
             const odb_SectionPoint& sectionPoint,
             odb_Enum::odb_ElementFaceEnum face,
             const odb_Node& node);
```

Required arguments

element An *OdbMeshElement* object specifying the element for which the data are to be collected.

Optional arguments

ipNumber An Int specifying the integration point. This argument is used to define a history output position of odb_Enum::INTEGRATION_POINT or odb_Enum::ELEMENT_FACE_INTEGRATION_POINT. The default value is 0.

sectionPoint A *SectionPoint* object.

face An odb_Enum::odb_ElementFaceEnum specifying the element face. This argument is used to define a history output position of odb_Enum::ELEMENT_FACE or odb_Enum::ELEMENT_FACE_INTEGRATION_POINT. Possible values are:

- odb_Enum::FACE_UNKOWN, specifying this value indicates that no value has been specified.
- odb_Enum::FACE1, specifying this value indicates that element face 1 has been specified.
- odb_Enum::FACE2, specifying this value indicates that element face 2 has been specified.
- odb_Enum::FACE3, specifying this value indicates that element face 3 has been specified.
- odb_Enum::FACE4, specifying this value indicates that element face 4 has been specified.
- odb_Enum::FACE5, specifying this value indicates that element face 5 has been specified.
- odb_Enum::FACE6, specifying this value indicates that element face 6 has been specified.
- odb_Enum::SIDE1, specifying this value indicates that element side 1 has been specified.
- odb_Enum::SIDE2, specifying this value indicates element side 2 has been specified.
- odb_Enum::END1, specifying this value indicates that element end 1 has been specified.
- odb_Enum::END2, specifying this value indicates that element end 2 has been specified.
- odb_Enum::END3, specifying this value indicates that element end 3 has been specified.

The default value is odb_Enum::FACE_UNKNOWN.

node An *OdbMeshNode* object specifying the node for which the data are to be collected.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for a region.

Path

```
odb.steps()[name].historyRegions()[name].HistoryPoint
```

Prototype

```
odb_HistoryPoint&
HistoryPoint(const odb_Set& region);
```

Required arguments

region An *OdbSet* object specifying the region for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for the *OdbAssembly* object.

Path

```
odb.steps()[name].historyRegions()[name].HistoryPoint
```

Prototype

```
odb_HistoryPoint&
HistoryPoint(const odb_Assembly& assembly);
```

Required arguments

assembly An *OdbAssembly* object specifying the assembly for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

HistoryPoint(...)

This method creates a HistoryPoint object for the *OdbInstance* object.

Path

```
odb.steps()[name].historyRegions()[name].HistoryPoint
```

Prototype

```
odb_HistoryPoint&
HistoryPoint(const odb_Instance& instance);
```

Required arguments

instance An *OdbInstance* object specifying the instance for which the data are to be collected.

Optional arguments

None.

Return value

A HistoryPoint object.

Exceptions

None.

Members

The HistoryPoint object has members with the same names and descriptions as the arguments to the *HistoryPoint* method.

In addition, the HistoryPoint object can have the following members:

Prototype

```
odb_Node node() const;
odb_Element element() const;
int ipNumber() const;
odb_Enum::odb_ElementFaceEnum face() const;
odb_Enum::odb_ResultPositionEnum position() const;
odb_SectionPoint sectionPoint() const;
const odb_Set& region() const;
const odb_Instance& instance() const;
const odb_Assembly& assembly() const;
```

ipNumber An Int specifying the integration point. This argument is used to define a history output position of odb_Enum::INTEGRATION_POINT or odb_Enum::ELEMENT_FACE_INTEGRATION_POINT. The default value is 0.

face An odb_Enum::odb_ElementFaceEnum specifying the element face. This argument is used to define a history output position of odb_Enum::ELEMENT_FACE or odb_Enum::ELEMENT_FACE_INTEGRATION_POINT. Possible values are:

- odb_Enum::FACE_UNKOWN, specifying this value indicates that no value has been specified.
- odb_Enum::FACE1, specifying this value indicates that element face 1 has been specified.
- odb_Enum::FACE2, specifying this value indicates that element face 2 has been specified.

- odb_Enum::FACE3, specifying this value indicates that element face 3 has been specified.
- odb_Enum::FACE4, specifying this value indicates that element face 4 has been specified.
- odb_Enum::FACE5, specifying this value indicates that element face 5 has been specified.
- odb_Enum::FACE6, specifying this value indicates that element face 6 has been specified.
- odb_Enum::SIDE1, specifying this value indicates that element side 1 has been specified.
- odb_Enum::SIDE2, specifying this value indicates element side 2 has been specified.
- odb_Enum::END1, specifying this value indicates that element end 1 has been specified.
- odb_Enum::END2, specifying this value indicates that element end 2 has been specified.
- odb_Enum::END3, specifying this value indicates that element end 3 has been specified.

The default value is odb_Enum::FACE_UNKNOWN.

position An odb_Enum::odb_ResultPositionEnum specifying the result position of the history point. Possible values are:

- odb_Enum::NODAL, specifying the values calculated at the nodes.
- odb_Enum::ELEMENT_NODAL, specifying the values obtained by extrapolating results calculated at the integration points.
- odb_Enum::INTEGRATION_POINT, specifying the values calculated at the integration points.
- odb_Enum::ELEMENT_FACE, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element.
- odb_Enum::ELEMENT_FACE_INTEGRATION_POINT, specifying the results obtained for surface variables such as cavity radiation that are defined for the surface facets of an element when the surface facets have integration points.
- odb_Enum::WHOLE_ELEMENT, specifying the results obtained for whole element variables.
- odb_Enum::WHOLE_REGION, specifying the results for an entire region of the model.
- odb_Enum::WHOLE_PART_INSTANCE, specifying the results for an entire part instance of the model.
- odb_Enum::WHOLE_MODEL, specifying the results for the entire model.

element An *OdbMeshElement* object specifying the element for which the data are to be collected.

sectionPoint A *SectionPoint* object.

region An *OdbSet* object specifying the region for which the data are to be collected.

assembly An *OdbAssembly* object specifying the assembly for which the data are to be collected.

instance An *OdbInstance* object specifying the instance for which the data are to be collected.

HistoryRegion object

The HistoryRegion object contains history data for a single location in the model.

Access

```
odb.steps()[name].historyRegions()[name]
```

HistoryRegion(...)

This method creates a HistoryRegion object.

Path

```
odb.steps()[name].HistoryRegion
```

Prototype

```
odb_HistoryRegion&
HistoryRegion(const odb_String& name,
              const odb_String& description,
              const odb_HistoryPoint& point,
              const odb_LoadCase& loadCase);
```

Required arguments

name An odb_String specifying the name of the HistoryRegion object.

description An odb_String specifying the description of the HistoryRegion object.

point A *HistoryPoint* object specifying the point to which the history data refer.

Optional arguments

loadCase An *OdbLoadCase* object specifying the load case associated with the HistoryRegion object.

Return value

A HistoryRegion object.

Exceptions

None.

getSubset(...)

This method returns a subset of the data in the HistoryRegion object.

Prototype

```
odb_HistoryRegion
getSubset(const odb_String& variableName);
```

Required arguments

variableName A String specifying the name of the output variable to return.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

`getSubset(...)`

This method returns a subset of the data in the HistoryRegion object.

Prototype

```
odb_HistoryRegion
getSubset(float start);
```

Required arguments

start A Float specifying the start of the subset. This is the same as the first item in the data array member of the *HistoryOutput* object.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

`getSubset(...)`

This method returns a subset of the data in the HistoryRegion object.

Prototype

```
odb_HistoryRegion
getSubset(float start,
          float end);
```

Required arguments

start A Float specifying the start of the subset. This is the same as the first item in the data array member of the *HistoryOutput* object.

end A Float specifying the end of the subset.

Optional arguments

None.

Return value

A HistoryRegion object.

Exceptions

None.

Members

The HistoryRegion object has members with the same names and descriptions as the arguments to the *HistoryRegion* method.

In addition, the HistoryRegion object can have the following members:

Prototype

```
odb_String description() const;
const odb_LoadCase& loadCase() const;
odb_String name() const;
odb_HistoryPoint point() const;
odb_Enum::odb_ResultPositionEnum position() const;
const odb_HistoryOutputRepository& historyOutputs();
```

position An odb_Enum::odb_ResultPositionEnum specifying the position of the history output. Possible values are odb_Enum::NODAL, odb_Enum::INTEGRATION_POINT, odb_Enum::WHOLE_ELEMENT, odb_Enum::WHOLE_REGION, and odb_Enum::WHOLE_MODEL.

historyOutputs A repository of *HistoryOutput* objects.

JobData object

The JobData object describes the context in which the analysis was run.

Access

```
odb.jobData()
```

Members

The JobData object has the following members:

Prototype

```
odb_String name() const;
           odb_String analysisCode() const;
           odb_String version() const;
           odb_String dateRun() const;
           odb_String machineName() const;
           odb_String precision() const;
```

name
An odb_String specifying the name of the job.

analysisCode
An odb_Enum::odb_AnalysisCodeEnum specifying the analysis code. Possible values are odb_Enum::ABAQUS_STANDARD, odb_Enum::ABAQUS_EXPLICIT, and odb_Enum::UNKNOWN_ANALYSIS_CODE.

precision
An odb_Enum::odb_PrecisionEnum specifying the precision. Only odb_Enum::SINGLE_PRECISION is currently supported. Possible values are odb_Enum::DOUBLE_PRECISION and odb_Enum::SINGLE_PRECISION.

version
An odb_String specifying the release of the analysis code.

creationTime
An odb_String specifying the date and time at which the analysis was run.

modificationTime
An odb_String specifying the date and time at which the database was last modified.

machineName
An odb_String specifying the name of the machine on which the analysis was run.

productAddOns
An odb_String specifying an odb_Sequence of productAddOns. Possible values are: Possible values are odb_Enum::AQUA, odb_Enum::DESIGN, odb_Enum::BIORID, odb_Enum::CEL, odb_Enum::SOLITER, and odb_Enum::CAVPARALLEL.

OdbAssembly object

The OdbAssembly object has no constructor; it is created automatically when an *Odb* object is created. Abaqus creates the *rootAssembly* member when an *Odb* object is created.

Access

```
odb.rootAssembly()
```

ConnectorOrientation(...)

This method assigns a connector orientation to a connector region.

Prototype

```
void
ConnectorOrientation(const odb_Set& region,
                     const odb_DatumCsys& csys1,
                     odb_Enum::odb_AxisEnum axis1,
                     float angle1,
                     const odb_DatumCsys& csys2,
                     odb_Enum::odb_AxisEnum axis2,
                     float angle2);
```

Required arguments

region An *OdbSet* specifying a region.

csys1 An *OdbDatumCsys* object specifying the first connector node local coordinate system.

axis1 An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation of the first connector node is applied. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3.

angle1 A Float specifying the angle of the additional rotation about the first connector node axis. The default value is 0.0.

csys2 An *OdbDatumCsys* object specifying the second connector node local coordinate system.

axis2 An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation of the second connector node is applied. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3.

angle2 A Float specifying the angle of the additional rotation about the second connector node axis.

Return value

`None.`

Exceptions

If *region* is not an element set: OdbError: Connector orientation assignment requires element set.

ConnectorOrientation(...)

This method assigns a connector orientation to a connector region.

Prototype

```
void
ConnectorOrientation(const odb_Set& region,
                     const odb_DatumCsys& csys1,
                     odb_Enum::odb_AxisEnum axis1,
                     float angle1,
                     bool orient2sameAs1);
```

Required arguments

region An *OdbSet* specifying a region.

csys1 An *OdbDatumCsys* object specifying the first connector node local coordinate system.

axis1 An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation of the first connector node is applied. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3.

angle1 A Float specifying the angle of the additional rotation about the first connector node axis.

Optional arguments

orient2sameAs1 A Boolean specifying whether the same orientation settings should be used for the second node of the connector. The default value is false.

Return value

None.

Exceptions

If *region* is not an element set: OdbError: Connector orientation assignment requires element set.

SectionAssignment(...)

This method is used to assign a section on an assembly or part. Section assignment on the assembly is limited to the connector elements only.

Prototype

```
void
SectionAssignment(const odb_Set& region,
                  const odb_section& section);
```

Required arguments

region	An *OdbSet* specifying a region.
section	A *Section* object.

Optional arguments

None.

Return value

`None.`

Exceptions

If *region* is not an element set:	OdbError: Section assignment requires element set.

`addElements(...)`

This method is used to define elements using nodes defined at the OdbAssembly and/or *OdbInstance* level. For connector elements connected to ground, specify the lone node in the connectivity. The position of the ground node cannot be specified. This is a limitation.



Warning:

Adding elements not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Prototype

```
void
addElements(const odb_SequenceInt& labels,
            const odb_SequenceSequenceInt& connectivity,
            const odb_SequenceString& instanceNames,
            const odb_String& type,
            const odb_String& elementSetName,
            const odb_SectionCategory& sectionCategory);
```

Required arguments

labels	An odb_SequenceInt specifying the element labels.
connectivity	An odb_SequenceSequenceInt specifying the nodal connectivity.
instanceNames	An odb_SequenceString specifying the instanceNames of each node in the nodal connectivity array. If the node is defined at the assembly level, the instance name should be an empty string
type	A String specifying the element type.

Optional arguments

elementSetName	A String specifying a name for this element set. The default value is the empty string.
sectionCategory	A *SectionCategory* object for this element set.

Return value

`None.`

Exceptions

Only certain element types are permitted at the assembly level. e.g., connector elements.	OdbError: Addition of this element type is not permitted at the assembly level
If length of label array does not match connectivity data length:	OdbError: Connectivity array must be provided for all element

`addNodes(...)`

This method adds nodes to the OdbAssembly object using node labels and coordinates.



Warning:

Adding nodes not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Prototype

```
void
addNodes(const odb_SequenceInt& labels,
         const odb_SequenceSequenceFloat& coordinates,
         const odb_String& nodeSetName);
```

Required arguments

labels	An odb_SequenceInt specifying the node labels.
coordinates	An odb_SequenceSequenceFloat specifying the nodal coordinates.

Optional arguments

nodeSetName	A String specifying a name for this node set. The default value is None.

Return value

`None.`

Exceptions

If length of labels does not match length of coordinates:	OdbError: Number of node labels and coordinates does not match
If width of coordinate array does not match assembly dimension:	OdbError: Node location specification does not correspond to part dimensions

sectionAssignments(...)

This method is used to retrieve a section assignment.

Prototype

```
odb_SectionAssignment
sectionAssignments(int index);
```

Required arguments

index	An Int specifying the section assignment.

Optional arguments

None.

Return value

A *SectionAssignment* object.

Exceptions

None.

RigidBody(...)

This method defines an *OdbRigidBody* on the assembly.

Prototype

```
void
RigidBody(const odb_Set& referenceNode,
          odb_Enum::odb_PositionEnum position,
          bool isothermal,
          const odb_Set& elset,
          const odb_Set& pinNodes,
          const odb_Set& tieNodes);
```

Required arguments

referenceNode	An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are odb_Enum::INPUT and odb_Enum::CENTER_OF_MASS. The default value is odb_Enum::INPUT.

isothermal A Boolean specifying an isothermal rigid body. The default value is false. This parameter is used only for a fully coupled thermal stress analysis.

elset An *OdbSet* specifying an element set assigned to the rigid body.

pinNodes An *OdbSet* specifying pin-type nodes assigned to the rigid body.

tieNodes An *OdbSet* specifying tie-type nodes assigned to the rigid body.

Return value

`None.`

Exceptions

If *referenceNode* is not a node set: OdbError: Rigid body definition requires a node set.

Members

The OdbAssembly object can have the following members:

Prototype

```
odb_InstanceRepository& instances();
                odb_SetRepository& nodeSets();
                odb_SetRepository& elementSets();
                odb_SetRepository& surfaces();
                odb_SequenceNode& nodes() const;
                odb_SequenceElement& elements() const;
                odb_DatumCsysRepository& datumCsyses();
                odb_SequenceSectionAssignment sectionAssignments();
                odb_SequenceConnectorOrientation connectorOrientations();
                odb_SequenceRigidBody rigidBodies();
                odb_SequencePretensionSection pretensionSections();
```

instances A repository of *OdbInstance* objects.

nodeSets A repository of *OdbSet* objects specifying node sets.

elementSets A repository of *OdbSet* objects specifying element sets.

surfaces A repository of *OdbSet* objects specifying surfaces.

nodes A sequence of *OdbMeshNode* objects.

elements
A sequence of *OdbMeshElement* objects.

datumCsyses
A repository of *OdbDatumCsys* objects.

sectionAssignments
A sequence of *SectionAssignment* objects.

rigidBodies
A sequence of *OdbRigidBody* objects.

pretensionSections
A sequence of *OdbPretensionSection* objects.

connectorOrientations
A sequence of *ConnectorOrientation* objects.

OdbDatumCsys object

The OdbDatumCsys object contains a coordinate system that can be stored in an output database. You can create the datum coordinate system in the Visualization module during an Abaqus/CAE session and save the datum coordinate system to the output database before you exit Abaqus/CAE. Alternatively, the analysis code can write the datum coordinate system to the output database.

Access

```
odb.rootAssembly().datumCsyses()[name]
```

DatumCsysByThreePoints(...)

This method creates an OdbDatumCsys object using three points. A datum coordinate system created with this method results in a fixed system.

Path

```
odb.rootAssembly().DatumCsysByThreePoints
```

Prototype

```
odb_DatumCsys&
DatumCsysByThreePoints(const odb_String& name,
                 odb_Enum::odb_DatumCsysTypeEnum type,
                 const odb_SequenceFloat& origin,
                 const odb_SequenceFloat& point1,
                 const odb_SequenceFloat& point2);
```

Required arguments

name
An odb_String specifying the repository key.

type
An odb_Enum::odb_DatumCsysTypeEnum specifying the type of coordinate system. Possible values are odb_Enum::CARTESIAN, odb_Enum::CYLINDRICAL, and odb_Enum::SPHERICAL.

origin
An odb_SequenceFloat specifying the coordinates of the origin of the datum coordinate system.

point1
An odb_SequenceFloat specifying the coordinates of a point on the local 1- or r-axis.

point2
An odb_SequenceFloat specifying the coordinates of a point in the 1–2 or r–θ plane.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsysByThreeNodes(...)

This method creates an OdbDatumCsys object using the coordinates of three *OdbMeshNode* objects. A datum coordinate system created with this method results in a system that follows the position of the three nodes. Results, such as those for displacement, are resolved into the orientation of the datum coordinate system without regard to the position of its origin. The last three arguments are given in the form of an *OdbMeshNode* object.

Path

```
odb.rootAssembly().DatumCsysByThreeNodes
```

Prototype

```
odb_DatumCsys&
DatumCsysByThreeNodes(const odb_String& name,
                      odb_Enum::odb_DatumCsysTypeEnum type,
                      const odb_Node& origin,
                      const odb_Node& point1,
                      const odb_Node& point2);
```

Required arguments

name
An odb_String specifying the repository key.

type
An odb_Enum::odb_DatumCsysTypeEnum specifying the type of coordinate system. Possible values are odb_Enum::CARTESIAN, odb_Enum::CYLINDRICAL, and odb_Enum::SPHERICAL.

origin
An *OdbMeshNode* object specifying a node at the origin of the datum coordinate system.

point1
An *OdbMeshNode* object specifying a node on the local 1- or r-axis.

point2
An *OdbMeshNode* object specifying a node in the 1–2 or r–θ plane.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

DatumCsys(...)

This method copies oneOdbDatumCsys object to a new OdbDatumCsys object.

Path

```
odb.rootAssembly().DatumCsys
```

Prototype

```
odb_DatumCsys&
DatumCsys(const odb_String& name,
          const odb_DatumCsys& datumCsys);
```

Required arguments

name An odb_String specifying the repository key.

datumCsys An OdbDatumCsys object specifying the object to be copied.

Optional arguments

None.

Return value

An OdbDatumCsys object.

Exceptions

None.

Members

The OdbDatumCsys object has the following members:

Prototype

```
odb_String name() const;
odb_Enum::odb_DatumCsysTypeEnum type() const;
const float* origin() const;
float origin(int index) const;
const float* xAxis() const;
float xAxis(int index) const;
const float* yAxis() const;
float yAxis(int index) const;
const float* zAxis() const;
float zAxis(int index) const;
```

name An odb_String specifying the repository key.

type An odb_Enum::odb_DatumCsysTypeEnum specifying the type of coordinate system. Possible values are odb_Enum::CARTESIAN, odb_Enum::CYLINDRICAL, and odb_Enum::SPHERICAL.

origin An odb_SequenceFloat specifying the coordinates of the origin of the datum coordinate system.

xAxis An odb_SequenceFloat specifying a point on the *X*-axis.

yAxis An odb_SequenceFloat specifying a point on the *Y*-axis.

zAxis An odb_SequenceFloat specifying a point on the *Z*-axis.

OdbFrame object

The domain of the OdbFrame object is taken from the parent step.

Access

```
odb.steps()[name].frames(i)
```

Frame(...)

This method creates an OdbFrame object and appends it to the frame sequence.

Path

```
odb.steps()[name].Frame
```

Prototype

```
odb_Frame
                Frame(int incrementNumber,
                float frameValue,
                const odb_String& description);
```

Required arguments

incrementNumber An Int specifying the frame increment number within the step. The base frame has normally increment number 0, and the results run from 1. In case of multiple load cases, the same increment number is duplicated for each loadcase.

frameValue A Float specifying the value in units determined by the *domain* member of the Step object. The equivalent in the time domain is *stepTime*; in the frequency domain the equivalent is *frequency*; and in the modal domain the equivalent is *mode*.

Optional arguments

description An odb_String specifying the contents of the frame. The default value is an empty string.

Return value

An OdbFrame object.

Exceptions

None.

Frame(...)

This constructor creates an OdbFrame object in the frequency domain and appends it to the frame sequence. The arguments to the constructor are valid only when *domain*=FREQUENCY or *domain*=MODAL.

Path

```
odb.steps()[name].Frame
```

Prototype

```
odb_Frame
                  Frame(int mode,
                  float frequency,
                  const odb_String& description);
```

Required arguments

mode
An Int specifying the eigenmode. This member is valid only if *domain*=odb_Enum::MODAL.

frequency
A Float specifying the frequency. This member is valid only if *domain*=odb_Enum::FREQUENCY or if the *procedureType* member of the Step object="FREQUENCY". The default value is 0.0.

Optional arguments

description
An odb_String specifying the contents of the frame. The default value is an empty string.

Return value

An OdbFrame object.

Exceptions

None.

Frame(...)

This constructor creates an OdbFrame object for a specific load case and appends it to the frame sequence.

Path

```
odb.steps()[name].Frame
```

Prototype

```
odb_Frame
                  Frame(const odb_LoadCase& loadCase,
                  const odb_String& description,
                  float frequency);
```

Required arguments

loadCase
An *OdbLoadCase* object specifying the load case for the frame.

Optional arguments

description

An odb_String specifying the contents of the frame. The default value is an empty string.

frequency

A Float specifying the frequency. This member is valid only if *domain*=odb_Enum::FREQUENCY or if the *procedureType* member of the Step object="FREQUENCY". The default value is 0.0.

Return value

An OdbFrame object.

Exceptions

None.

Members

The OdbFrame object has members with the same names and descriptions as the arguments to the *Frame* method.

In addition, the OdbFrame object can have the following members:

Prototype

```
odb_Frame associatedFrame();
int cyclicModeNumber();
odb_String description() const;
odb_Enum::odb_DomainEnum domain() const;
odb_FieldOutputRepository& fieldOutputs();
int incrementNumber() const;
float frameValue() const;
float frequency() const;
const odb_LoadCase& loadCase() const;
int mode() const;
```

cyclicModeNumber

An Int specifying the cyclic mode number associated with the data stored on this frame. Only frequency analyses of cyclic symmetry models possess cyclic mode numbers.

domain

An odb_Enum::odb_DomainEnum specifying the domain of the step of which the frame is a member. Possible values are odb_Enum::TIME, odb_Enum::FREQUENCY, and odb_Enum::MODAL.

frequency

A Float specifying the frequency. This member is valid only if *domain*=odb_Enum::FREQUENCY or if the *procedureType* member of the Step object="FREQUENCY". The default value is 0.0.

mode

An Int specifying the eigenmode. This member is valid only if *domain*=odb_Enum::MODAL.

associatedFrame

An OdbFrame object specifying the real or imaginary portion of the data corresponding to this cyclic symmetry mode.

fieldOutputs

A repository of *FieldOutput* objects specifying the key to the *fieldOutputs*repository is a String representing an output variable.

loadCase

An *OdbLoadCase* object specifying the load case for the frame.

OdbInstance object

A part instance is the usage of a part within an assembly.

Access

```
odb.rootAssembly().instances()[name]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()
```

Instance(...)

This method creates an OdbInstance object from an *OdbPart* object.

Path

odb.rootAssembly().Instance

Prototype

```
odb_Instance&
Instance(const odb_String& name,
         const odb_Part& object,
         const odb_SequenceSequenceFloat& localCoordSystem);
```

Required arguments

name An odb_String specifying the instance name.

object An *OdbPart* object.

Optional arguments

localCoordSystem An odb_SequenceSequenceFloat specifying the rotation and translation of the part instance in the global Cartesian coordinate system. The first three sequences specify the new local coordinate system with its center at the origin.

- The first sequence specifies a point on the 1-axis.
- The second sequence specifies a point on the 2-axis.
- The third sequence specifies a point on the 3-axis.

The fourth sequence specifies the translation of the local coordinate system from the origin to its intended location.

For example, the following sequence moves a part 10 units in the *X*-direction with no rotation:

```
localCoordSystem = ((1, 0, 0), (0, 1, 0),
                                (0, 0, 1), (10, 0, 0))
```

The following sequence moves a part 5 units in the *X*-direction with rotation:

```
localCoordSystem = ((0, 1, 0), (1, 0, 0),
                                (0, 0, 1), (5, 0, 0))
```

transforms a part containing the two points

```
Pt1= (1,0,0)
Pt2= (2,0,0)
```

to

```
Pt1 = (0, 6, 0)
Pt2 = (0, 7, 0)
```

Return value

An OdbInstance object.

Exceptions

InvalidNameError.

assignBeamOrientation(...)

This method assigns a beam section orientation to a region of a part instance.

Prototype

```
void
assignBeamOrientation(const odb_Set& region,
                      odb_Enum::odb_OrientationMethodEnum method,
                      const odb_SequenceFloat& vector);
```

Required arguments

region

An *OdbSet* specifying a region on an instance.

method

An odb_Enum::odb_OrientationMethodEnum specifying the assignment method. Only a value of odb_Enum::N1_COSINES is currently supported.

vector

An odb_SequenceFloat specifying the approximate local $\mathbf{n_1}$-direction of the beam cross-section.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignMaterialOrientation(...)

This method assigns a material orientation to a region of a part instance.

Prototype

```
void
assignMaterialOrientation(const odb_Set& region,
              const odb_DatumCsys& localCsys,
              odb_Enum::odb_AxisEnum axis,
              float angle,
              odb_Enum::odb_StackDirectionEnum stackDirection);
```

Required arguments

region
An *OdbSet* specifying a region on an instance.

localCsys
An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis
An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3. The default value is odb_Enum::AXIS_1.

angle
A Float specifying the angle of the additional rotation. The default value is 0.0.

stackDirection
An odb_Enum::odb_StackDirectionEnum specifying the stack or thickness direction of the material. Possible values are odb_Enum::STACK_1, odb_Enum::STACK_2, odb_Enum::STACK_3, and odb_Enum::STACK_ORIENTATION. The default value is odb_Enum::STACK_3.

Return value

`None.`

Exceptions

None.

assignRebarOrientation(...)

This method assigns a rebar reference orientation to a region of a part instance.

Prototype

```
void
assignRebarOrientation(const odb_Set& region,
                       const odb_DatumCsys& localCsys,
                       odb_Enum::odb_AxisEnum axis,
                       float angle);
```

Required arguments

region An *OdbSet* specifying a region on an instance.

localCsys An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3. The default value is odb_Enum::AXIS_1.

angle A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

`None.`

Exceptions

None.

`getElementFromLabel(...)`

This method is used to retrieved an element with a specific label from an instance object.

Prototype

```
odb_Element
getElementFromLabel(int label);
```

Required arguments

label An Int specifying the element label.

Optional arguments

None.

Return value

An *OdbMeshElement* object.

Exceptions

If no element with the specified label exists: OdbError: Invalid element label

getNodeFromLabel(...)

This method is used to retrieved a node with a specific label from an instance object.

Prototype

```
odb_Node
getNodeFromLabel(int label);
```

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An*OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

assignSection(...)

This method is used to assign a section to a region on an instance.

Prototype

```
void
assignSection(const odb_Set& region,
              const odb_section& section);
```

Required arguments

region An *OdbSet* specifying a region on an instance.

section A *Section* object.

Optional arguments

None.

Return value

None.

Exceptions

If *region* is not an element set:	OdbError: Section assignment requires element set.
If the element set is not from the current instance:	OdbError: Section assignment requires element set from this part instance.

AnalyticRigidSurf2DPlanar(...)

This method is used to define a two-dimensional *AnalyticSurface* object on the instance.

Prototype

```
void
AnalyticRigidSurf2DPlanar(const odb_String& name,
          const odb_SequenceAnalyticSurfaceSegment& profile,
          double filletRadius);
```

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius	A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If *OdbPart* associated with the part instance is of type THREE_D:	OdbError: 2D-Planar Analytic Rigid Surface can be defined only if the instance is of type TWO_D_PLANAR or AXISYMMETRIC.

AnalyticRigidSurfExtrude(...)

This method is used to define a three-dimensional cylindrical *AnalyticSurface* on the instance.

Prototype

```
void
AnalyticRigidSurfExtrude(const odb_String& name,
          const odb_SequenceAnalyticSurfaceSegment& profile,
          double filletRadius,
          const odb_SequenceSequenceFloat& localCoordData);
```

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius	A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.
localCoordData	An odb_SequenceSequenceFloat specifying the global coordinates of points used to define the local coordinate system.

Return value

`None.`

Exceptions

If *OdbPart* associated with the part instance is not of type THREE_D:	OdbError: Analytic Rigid Surface of type CYLINDER can be defined only if the instance is of type THREE_D.

`AnalyticRigidSurfRevolve(...)`

This method is used to define a three-dimensional *AnalyticSurface* of revolution on the instance.

Prototype

```
void
AnalyticRigidSurfRevolve(const odb_String& name,
           const odb_SequenceAnalyticSurfaceSegment& profile,
           double filletRadius,
           const odb_SequenceSequenceFloat& localCoordData);
```

Required arguments

name	The name of the analytic surface.
profile	A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius	A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.
localCoordData	An odb_SequenceSequenceFloat specifying the global coordinates of points used to define the local coordinate system.

Return value
None.

Exceptions

If *OdbPart* associated with the part instance is not of type THREE_D:	OdbError: Analytic Rigid Surface of type REVOLUTION can be defined only if the instance is of type THREE_D.

RigidBody(...)

This method defines an *OdbRigidBody* on the instance.

Prototype

```
void
RigidBody(const odb_Set& referenceNode,
          odb_Enum::odb_PositionEnum position,
          bool isothermal,
          const odb_Set& elset,
          const odb_Set& pinNodes,
          const odb_Set& tieNodes);
```

Required arguments

referenceNode	An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position	A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are odb_Enum::INPUT, and odb_Enum::CENTER_OF_MASS. The default value is odb_Enum::INPUT.
isothermal	A Boolean specifying an isothermal rigid body. The default value is false. This parameter is used only for a fully-coupled thermal stress analysis.
elset	An *OdbSet* specifying an element set assigned to the rigid body.
pinNodes	An *OdbSet* specifying pin-type nodes assigned to the rigid body.
tieNodes	An *OdbSet* specifying tie-type nodes assigned to the rigid body.

Return value
None.

Exceptions

If *referenceNode* is not a node set:	OdbError: Rigid body definition requires a node set.

getNodeFromLabel(...)

This method is used to retrieved a node with a specific label from an instance object.

Prototype

```
odb_Node
getNodeFromLabel(int label);
```

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

getNodeFromLabel(...)

This method is used to retrieved a node with a specific label from an instance object.

Prototype

```
odb_Node
getNodeFromLabel(int label);
```

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

nodesLabelsFromNodeTypeFace(...)

Given a sequence of integers defining an element connectivity, this method is used to retrieve the sequence of nodes attached to a specified face of the element based on element type.

Prototype

```
odb_SequenceInt
nodesLabelsFromNodeTypeFace(const odb_SequenceInt& elementNodeLabels,
                const odb_String& elementType,
                odb_Enum::odb_ElementFaceEnumEnum elementFace);
```

Required arguments

elementNodeLabels	A sequence of Ints specifying the node labels defining the connectivity for the entire element.
elementType	A string specifying the type of element whose connectivity is provided.
elementFace	A SymbolicConstant specifying the face of the element for which the node labels are desired.

Optional arguments

None.

Return value

A sequence of Ints specifying the node labels corresponding to the desired element face.

Exceptions

If the specified face is invalid for the element type:	OdbError: Invalid face

Members

The OdbInstance object can have the following members:

Prototype

```
odb_String name() const;
                odb_Enum::odb_DimensionEnum embeddedSpace() const;
                odb_Enum::odb_PartTypeEnum type() const;
                odb_Node nodes(int i) const;
                odb_SequenceNode& nodes() const;
                odb_Element elements(int i) const;
                odb_SequenceElement& elements() const;
                odb_SetRepository& nodeSets() const;
                odb_SetRepository& elementSets() const;
                odb_SetRepository& surfaces() const;
                odb_SequenceSectionAssignment sectionAssignments()\
                    const;
                odb_SequenceBeamOrientation beamOrientations() \
```

```
    const;
odb_SequenceMaterialOrientation materialOrientations() \
    const;
odb_SequenceRebarOrientation rebarOrientations() \
    const;
odb_SequenceRigidBody rigidBodies() const;
bool hasAnalyticSurface() const;
odb_AnalyticSurface analyticSurface() const;
```

name

An odb_String specifying the instance name.

type

An odb_Enum::odb_PartTypeEnum specifying the type of the *OdbPart* object. Only a value of odb_Enum::DEFORMABLE_BODY is currently supported.

embeddedSpace

An odb_Enum::odb_DimensionEnum specifying the dimensionality of the *OdbPart* object. Possible values are odb_Enum::THREE_D, odb_Enum::TWO_D_PLANAR, odb_Enum::AXISYMMETRIC, and odb_Enum::UNKNOWN_DIMENSION.

resultState

An odb_Enum::odb_ResultStateEnum specifying the state of the Instance as modified by the analysis. This member is only present if the Instance is part of the RootAssemblyState tree. Possible values are:

- odb_Enum::PROPAGATED, specifying that the value is the same as the previous frame or the original rootAssembly.
- odb_Enum::MODIFIED, specifying that the geometry of the instance has been changed at this frame.

The default value is odb_Enum::PROPAGATED.

nodes

A sequence of *OdbMeshNode* objects.

elements

A sequence of *OdbMeshElement* objects.

nodeSets

A repository of *OdbSet* objects specifying node sets.

elementSets

A repository of *OdbSet* objects specifying element sets.

surfaces

A repository of *OdbSet* objects specifying surfaces.

sectionAssignments

A sequence of *SectionAssignment* objects.

rigidBodies

A sequence of *OdbRigidBody* objects.

beamOrientations

A sequence of *BeamOrientation* objects.

materialOrientations

A sequence of MaterialOrientation objects.

rebarOrientations

A sequence of *RebarOrientation* objects.

analyticSurface

An *AnalyticSurface* object specifying analytic Surface defined on the instance.

OdbLoadCase object

The OdbLoadCase object describes a load case.

Access

```
odb.steps()[name].frames(i).loadCase()
odb.steps()[name].historyRegions()[name].loadCase()
odb.steps()[name].loadCases()[name]
```

LoadCase(...)

This method creates an OdbLoadCase object.

Path

odb.steps()[*name*].LoadCase

Prototype

```
odb_LoadCase&
LoadCase(const odb_String& name);
```

Required arguments

name An odb_String specifying the name of the OdbLoadCase object.

Optional arguments

None.

Return value

An OdbLoadCase object.

Exceptions

None.

Members

The OdbLoadCase object has members with the same names and descriptions as the arguments to the *LoadCase* method.

OdbMeshElement object

OdbMeshElement objects are created with the `part.addElements` or `rootAssembly.addElements` methods.

Access

```
odb.parts()[name].elements(i)
odb.parts()[name].elementSets()[name].elements(i)
odb.parts()[name].nodeSets()[name].elements(i)
odb.parts()[name].surfaces()[name].elements(i)
odb.rootAssembly().elements(i)
odb.rootAssembly().elementSets()[name].elements(i)
odb.rootAssembly().instances()[name].elements(i)
odb.rootAssembly().instances()[name].elementSets()[name].elements(i)
odb.rootAssembly().instances()[name].nodeSets()[name].elements(i)
odb.rootAssembly().instances()[name].surfaces()[name].elements(i)
odb.rootAssembly().nodeSets()[name].elements(i)
odb.rootAssembly().surfaces()[name].elements(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elements(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elementSets()[name].elements(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodeSets()[name].elements(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.surfaces()[name].elements(i)
```

`getNormal(...)`

This method returns the normal direction for the element face.

Required arguments

faceIndex The value of *faceIndex* is 0 for a shell element and can range from 0 to 5 for a solid element.

Optional arguments

stepName Name of the step.

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the total fime or frequency.

match A SymbolicConstant specifying which frame to return if there is no frame at the exact frame value. Possible values are CLOSEST, BEFORE, AFTER, and EXACT. The default value is CLOSEST.

When *match*=CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

A tuple of 3 floats representing the unit normal vector. If the element face is collapsed such that a normal cannot be computed, a zero-length vector is returned.

Exceptions

If the exact frame is not found:	OdbError: Frame not found.
If the step name is not found:	OdbError: Step is not present in the ODB.
If *frameValue* is not provided and *stepName* is empty:	*frameValue*OdbError: *stepName* should be specified with *frameValue*.

Members

The OdbMeshElement object can have the following members:

Prototype

```
int label() const;
                const int* connectivity(int& numNodes) const ;
                odb_SequenceInt connectivity() const;
                odb_SectionCategory sectionCategory() const;
                odb_String type() const;
                odb_String instanceName() const;
                odb_SequenceString instanceNames() const;
```

label

An Int specifying the element label.

type

An odb_String specifying the element type.

sectionCategory

A *SectionCategory* object specifying the element section properties.

connectivity

A pointer to an array of Ints specifying the element connectivity. For connector elements connected to ground, the other node is repeated in the connectivity data. The position of the ground node cannot be ascertained. This is a limitation. It is important to note the difference with MeshElement object of MDB where the connectivity is node indices instead of node labels.

instanceNames

A pointer to an array of Strings specifying the instance names for nodes in the element connectivity.

instanceName

An odb_String specifying the instance name.

OdbMeshNode object

OdbMeshNode objects are created with the `part.addNodes` method.

Access

```
odb.parts()[name].nodes(i)
odb.parts()[name].nodeSets()[name].nodes(i)
odb.parts()[name].surfaces()[name].nodes(i)
odb.rootAssembly().instances()[name].nodes(i)
odb.rootAssembly().instances()[name].nodeSets()[name].nodes(i)
odb.rootAssembly().instances()[name].surfaces()[name].nodes(i)
odb.rootAssembly().nodes(i)
odb.rootAssembly().nodeSets()[name].nodes(i)
odb.rootAssembly().surfaces()[name].nodes(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodes(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodeSets()[name].nodes(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.surfaces()[name].nodes(i)
```

Members

The OdbMeshNode object has the following members:

Prototype

```
int label() const;
const float* coordinates() const;
```

label

An Int specifying the node label.

coordinates

A pointer to an array of Floats specifying the nodal coordinates in the global Cartesian coordinate system.

OdbPart object

The OdbPart object is similar to the kernel Part object and contains nodes and elements, but not geometry.

Access

```
odb.parts()[name]
```

Part(...)

This method creates an OdbPart object. Nodes and elements are added to this object at a later stage.

Path

```
odb.Part
```

Prototype

```
odb_Part&
Part(const odb_String& name,
     odb_Enum::odb_DimensionEnum embeddedSpace,
     odb_Enum::odb_PartTypeEnum type);
```

Required arguments

name
An odb_String specifying the part name.

embeddedSpace
An odb_Enum::odb_DimensionEnum specifying the dimensionality of the *OdbPart* object. Possible values are odb_Enum::THREE_D, odb_Enum::TWO_D_PLANAR, and odb_Enum::AXISYMMETRIC.

type
An odb_Enum::odb_PartTypeEnum specifying the type of the *OdbPart* object. Possible values are odb_Enum::DEFORMABLE_BODY and odb_Enum::ANALYTIC_RIGID_SURFACE.

Optional arguments

None.

Return value

An OdbPart object.

Exceptions

None.

addElements(...)

This method adds elements to an OdbPart object using element labels and nodal connectivity.



Warning:

Adding elements not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Prototype

```
void
addElements(const odb_SequenceInt& labels,
            const odb_SequenceSequenceInt& connectivity,
            const odb_String& type,
            const odb_String& elementSetName,
            const odb_SectionCategory& sectionCategory);
```

Required arguments

labels
An odb_SequenceInt specifying the element labels.

connectivity
An odb_SequenceSequenceInt specifying the nodal connectivity.

type
A String specifying the element type.

Optional arguments

elementSetName
A String specifying a name for this element set. The default value is the empty string.

sectionCategory
A *SectionCategory* object for this element set.

Return value

None.

Exceptions

None.

addNodes(...)

This method adds nodes to an OdbPart object using node labels and coordinates.



Warning:

Adding nodes not in ascending order of their labels may cause Abaqus/Viewer to plot contours incorrectly.

Prototype

```
void
addNodes(const odb_SequenceInt& labels,
         const odb_SequenceSequenceFloat& coordinates,
         const odb_String& nodeSetName);
```

Required arguments

labels
An odb_SequenceInt specifying the node labels.

coordinates
An odb_SequenceSequenceFloat specifying the nodal coordinates.

Optional arguments

nodeSetName A String specifying a name for this node set. The default value is None.

Return value

`None.`

Exceptions

None.

assignBeamOrientation(...)

This method assigns a beam section orientation to a region of a part instance.

Prototype

```
void
assignBeamOrientation(const odb_Set& region,
                      odb_Enum::odb_OrientationMethodEnum method,
                      const odb_SequenceFloat& vector);
```

Required arguments

region An *OdbSet* specifying a region on an instance.

method An odb_Enum::odb_OrientationMethodEnum specifying the assignment method. Only a value of odb_Enum::N1_COSINES is currently supported.

vector An odb_SequenceFloat specifying the approximate local $\mathbf{n_1}$ -direction of the beam cross-section.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

assignMaterialOrientation(...)

This method assigns a material orientation to a region of a part instance.

Prototype

```
void
assignMaterialOrientation(const odb_Set& region,
                const odb_DatumCsys& localCSys,
                odb_Enum::odb_AxisEnum axis,
```

```
              float angle,
              odb_Enum::odb_StackDirectionEnum stackDirection);
```

Required arguments

region
An *OdbSet* specifying a region on an instance.

localCSys
An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis
An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3. The default value is odb_Enum::AXIS_1.

angle
A Float specifying the angle of the additional rotation. The default value is 0.0.

stackDirection
An odb_Enum::odb_StackDirectionEnum specifying the stack or thickness direction of the material. Possible values are odb_Enum::STACK_1, odb_Enum::STACK_2, odb_Enum::STACK_3, and odb_Enum::STACK_ORIENTATION. The default value is odb_Enum::STACK_3.

Return value
`None.`

Exceptions
None.

`assignRebarOrientation(...)`

This method assigns a rebar reference orientation to a region of a part instance.

Prototype

```
void
assignRebarOrientation(const odb_Set& region,
                       const odb_DatumCsys& localCsys,
                       odb_Enum::odb_AxisEnum axis,
                       float angle);
```

Required arguments

region
An *OdbSet* specifying a region on an instance.

localCsys
An *OdbDatumCsys* object specifying the local coordinate system or `None`, indicating the global coordinate system.

Optional arguments

axis
An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3. The default value is odb_Enum::AXIS_1.

angle
A Float specifying the angle of the additional rotation. The default value is 0.0.

Return value

`None.`

Exceptions

None.

`getElementFromLabel(...)`

This method is used to retrieved an element with a specific label from a part object.

Prototype

```
odb_Element
getElementFromLabel(int label);
```

Required arguments

label
An Int specifying the element label.

Optional arguments

None.

Return value

An *OdbMeshElement* object.

Exceptions

If no element with the specified label exists:
OdbError: Invalid element label

`getNodeFromLabel(...)`

This method is used to retrieved a node with a specific label from a part object.

Prototype

```
odb_Node
getNodeFromLabel(int label);
```

Required arguments

label An Int specifying the node label.

Optional arguments

None.

Return value

An *OdbMeshNode* object.

Exceptions

If no node with the specified label exists: OdbError: Invalid node label

AnalyticRigidSurf2DPlanar(...)

This method is used to define a two-dimensional *AnalyticSurface* object on the part object.

Prototype

```
void
AnalyticRigidSurf2DPlanar(const odb_String& name,
           const odb_SequenceAnalyticSurfaceSegment& profile,
           double filletRadius);
```

Required arguments

name The name of the analytic surface.

profile A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If OdbPart is of type THREE_D: OdbError: 2D-Planar Analytic Rigid Surface can be defined only if the part is of type TWO_D_PLANAR or AXISYMMETRIC.

AnalyticRigidSurfExtrude(...)

This method is used to define a three-dimensional cylindrical *AnalyticSurface* on the part object.

Prototype

```
void
AnalyticRigidSurfExtrude(const odb_String& name,
           const odb_SequenceAnalyticSurfaceSegment& profile,
           double filletRadius);
```

Required arguments

name The name of the analytic surface.

profile A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

None.

Exceptions

If OdbPart is not of type THREE_D: OdbError: Analytic Rigid Surface of type CYLINDER can be defined only if the part is of type THREE_D.

AnalyticRigidSurfRevolve(...)

This method is used to define a three-dimensional *AnalyticSurface* of revolution on the part object.

Prototype

```
void
AnalyticRigidSurfRevolve(const odb_String& name,
           const odb_SequenceAnalyticSurfaceSegment& profile,
           double filletRadius);
```

Required arguments

name The name of the analytic surface.

profile A sequence of *AnalyticSurfaceSegment* objects or an *OdbSequenceAnalyticSurfaceSegment* object.

Optional arguments

filletRadius A Double specifying the radius of curvature to smooth discontinuities between adjoining segments. The default value is 0.0.

Return value

`None.`

Exceptions

If OdbPart is not of type THREE_D: OdbError: Analytic Rigid Surface of type REVOLUTION can be defined only if the part is of type THREE_D.

`RigidBody(...)`

This method defines an *OdbRigidBody* on the part object.

Prototype

```
void
RigidBody(const odb_Set& referenceNode,
          odb_Enum::odb_PositionEnum position,
          bool isothermal,
          const odb_Set& elset,
          const odb_Set& pinNodes,
          const odb_Set& tieNodes);
```

Required arguments

referenceNode An *OdbSet* specifying the reference node assigned to the rigid body.

Optional arguments

position A symbolic constant specify if the location of the reference node is to be defined by the user. Possible values are odb_Enum::INPUT and odb_Enum::CENTER_OF_MASS. The default value is odb_Enum::INPUT.

isothermal A Boolean specifying an isothermal rigid body. The default value is false. This parameter is used only for a fully-coupled thermal stress analysis.

elset An *OdbSet* specifying an element set assigned to the rigid body.

pinNodes An *OdbSet* specifying pin-type nodes assigned to the rigid body.

tieNodes An *OdbSet* specifying tie-type nodes assigned to the rigid body.

Return value

`None.`

Exceptions

If *referenceNode* is not a node set:	OdbError: Rigid body definition requires a node set.

Members

The OdbPart object has members with the same names and descriptions as the arguments to the *Part* method.

In addition, the OdbPart object can have the following members:

Prototype

```
odb_String name() const;
                odb_Enum::odb_DimensionEnum embeddedSpace() const;
                odb_Enum::odb_PartTypeEnum type() const;
                odb_Node& nodes(int index) const;
                odb_SequenceNode& nodes() const;
                odb_Element& elements(int index) const;
                odb_SequenceElement& elements() const;
                odb_SetRepository& nodeSets() const;
                odb_SetRepository& elementSets() const;
                odb_SetRepository& surfaces() const;
                odb_SequenceSectionAssignment sectionAssignments() const;
                odb_SequenceBeamOrientation beamOrientations() const;
                odb_SequenceMaterialOrientation materialOrientations() const;
                odb_SequenceRebarOrientation rebarOrientations() const;
                odb_SequenceRigidBody rigidBodies() const;
                bool hasAnalyticSurface() const;
                odb_AnalyticSurface analyticSurface() const;
```

nodes	A sequence of *OdbMeshNode* objects.
elements	A sequence of *OdbMeshElement* objects.
nodeSets	A repository of *OdbSet* objects specifying node sets.
elementSets	A repository of *OdbSet* objects specifying element sets.
surfaces	A repository of *OdbSet* objects specifying surfaces.
sectionAssignments	A sequence of *SectionAssignment* objects.
beamOrientations	A sequence of *BeamOrientation* objects.
materialOrientations	A sequence of MaterialOrientation objects.
rebarOrientations	A sequence of *RebarOrientation* objects.
rigidBodies	A sequence of *OdbRigidBody* objects.
analyticSurface	An *AnalyticSurface* object specifying analytic Surface defined on the instance.

OdbPretensionSection object

The pretension section object is used to define an assembly load. It associates a pretension node with a pretension section.

Access

```
odb.rootAssembly().pretensionSections(i)
```

Members

The OdbPretensionSection object can have the following members:

Prototype

```
odb_Set node() const;
odb_Set element() const;
odb_Set surface() const;
odb_SequenceFloat normal() const;
float normal(int index) const;
```

node An *OdbSet* object specifying the node set containing the pretension node.

element An *OdbSet* object specifying the element set that defines the pretension section.

surface An *OdbSet* object specifying the surface set that defines the pretension section.

normal An odb_SequenceFloat specifying the components of the normal to the pretension section.

OdbRigidBody object

The Rigid body object is used to bind a set of elements and/or a set of nodes and/or an analytical surface with a reference node.

Access

```
odb.parts()[name].rigidBodies(i)
odb.rootAssembly().instances()[name].rigidBodies(i)
odb.rootAssembly().rigidBodies(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.rigidBodies(i)
```

Members

The OdbRigidBody object can have the following members:

Prototype

```
odb_Enum::odb_PositionEnum position() const;
bool isothermal() const;
odb_Set referenceNode() const;
odb_Set elements() const;
odb_Set tieNodes() const;
odb_Set pinNodes() const;
odb_AnalyticSurface analyticSurface() const;
```

position

An odb_Enum::odb_PositionEnum specifying the specific location of the OdbRigidBody reference node relative to the rest of the rigid body. Possible values are odb_Enum::INPUT and odb_Enum::CENTER_OF_MASS. The default value is odb_Enum::INPUT.

isothermal

A Boolean specifying specify whether the OdbRigidBody can have temperature gradients or be isothermal. This is used only for fully coupled thermal-stress analysis The default value is true.

referenceNode

An *OdbSet* object specifying the reference node set associated with the rigid body.

elements

An *OdbSet* object specifying the element set whose motion is governed by the motion of rigid body reference node.

tieNodes

An *OdbSet* object specifying the node set which have both translational and rotational degrees of freedom associated with the rigid body.

pinNodes

An *OdbSet* object specifying the node set which have only translational degrees of freedom associated with the rigid body.

analyticSurface

An *AnalyticSurface* object specifying the analytic surface whose motion is governed by the motion of rigid body reference node.

OdbSequenceAnalyticSurfaceSegment object

A sequence of *AnalyticSurfaceSegment* describing an analytic surface profile.

Access

```
odb.parts()[name].analyticSurface().segments()
odb.rootAssembly().instances()[name].analyticSurface().segments()
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.analyticSurface().segments()
```

`Start(...)`

This method adds a *AnalyticSurfaceSegment* describing the first segment of the surface profile.

Prototype

```
void
Start(const odb_SequenceFloat& origin);
```

Required arguments

origin An odb_SequenceFloat specifying the coordinates of start point.

Optional arguments

None.

Return value

None.

Exceptions

None.

`Line(...)`

This method adds a *AnalyticSurfaceSegment* describing the line segment of the surface profile.

Prototype

```
void
Line(const odb_SequenceFloat& endPoint);
```

Required arguments

endPoint An odb_SequenceFloat specifying the coordinates of end point.

Optional arguments

None.

Return value
None.

Exceptions
None.

Circle(...)

This method adds a *AnalyticSurfaceSegment* describing a circular segment of the surface profile.

Prototype
```
void
Circle(const odb_SequenceFloat& center,
       const odb_SequenceFloat& endPoint);
```

Required arguments

center	An odb_SequenceFloat specifying the coordinates of center of the circular segment.
endPoint	An odb_SequenceFloat specifying the coordinates of end point of the circular segment.

Optional arguments
None.

Return value
None.

Exceptions
None.

Parabola(...)

This method adds a *AnalyticSurfaceSegment* describing a parabolic segment of the surface profile.

Prototype
```
void
Parabola(const odb_SequenceFloat& middlePoint,
         const odb_SequenceFloat& endPoint);
```

Required arguments

middlePoint	An odb_SequenceFloat specifying the coordinates of middle point of the parabolic segment.
endPoint	An odb_SequenceFloat specifying the coordinates of end point of the parabolic segment.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

Members

The OdbSequenceAnalyticSurfaceSegment object has no members.

OdbSet object

The set objects are used to identify regions of a model.

Access

```
odb.parts()[name].elementSets()[name]
odb.parts()[name].nodeSets()[name]
odb.parts()[name].surfaces()[name]
odb.rootAssembly().elementSets()[name]
odb.rootAssembly().instances()[name].elementSets()[name]
odb.rootAssembly().instances()[name].nodeSets()[name]
odb.rootAssembly().instances()[name].surfaces()[name]
odb.rootAssembly().nodeSets()[name]
odb.rootAssembly().surfaces()[name]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elementSets()[name]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodeSets()[name]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.surfaces()[name]
```

NodeSet(...)

This method creates a node set from an array of *OdbMeshNode* objects (for part instance-level sets)

Path

odb.parts()[*name*].NodeSet

odb.rootAssembly().instances()[*name*].NodeSet

Prototype

```
odb_Set&
NodeSet(const odb_String& name,
        const odb_SequenceNode& nodes);
```

Required arguments

name

An odb_String specifying the name of the set and the repository key.

nodes

A sequence of *OdbMeshNode* objects. For example, for a part:

```
nodes=part1.nodes[1:5]
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

ElementSet(...)

This method creates an element set from an array of *OdbMeshElement* objects (for part instance-level sets).

Path

odb.parts()[*name*].ElementSet

odb.rootAssembly().instances()[*name*].ElementSet

Prototype

```
odb_Set&
ElementSet(const odb_String& name,
           const odb_SequenceElement& elements);
```

Required arguments

name

An odb_String specifying the name of the set and the repository key.

elements

A sequence of *OdbMeshElement* objects. For example, for a part:

```
elements=instance1.elements[1:5]
```

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

InvalidNameError.

NodeSet(...)

This method creates a node set.

Path

odb.parts()[*name*].NodeSet

odb.rootAssembly().instances()[*name*].NodeSet

Prototype

```
odb_Set&
NodeSet(const odb_String& name,
        const odb_SequenceInt& nodeLabels);
```

Required arguments

name	An odb_String specifying the name of the set and the repository key.
nodeLabels	An odb_SequenceInt specifying the node labels.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

NodeSet(...)

This method creates a node set.

Path

odb.rootAssembly().NodeSet

Prototype

```
odb_Set&
NodeSet(const odb_String& name,
        const odb_SequenceString& instanceNames,
        const odb_SequenceSequenceInt& nodeLabels);
```

Required arguments

name	An odb_String specifying the name of the set and the repository key.
instanceNames	An odb_SequenceString specifying the namespaces for *nodeLabels*. The string could be an empty string to denote the nodeLabels at the assembly level.
nodeLabels	An odb_SequenceSequenceInt specifying the node labels.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

ElementSet(...)

This method creates an element set.

Path

odb.parts()[*name*].ElementSet

odb.rootAssembly().instances()[*name*].ElementSet

Prototype

```
odb_Set&
ElementSet(const odb_String& name,
           const odb_SequenceInt& elementLabels);
```

Required arguments

name An odb_String specifying the name of the set and the repository key.

elementLabels An odb_SequenceInt specifying the element labels.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

ElementSet(...)

This method creates an element set.

Path

odb.rootAssembly().ElementSet

Prototype

```
odb_Set&
ElementSet(const odb_String& name,
           const odb_SequenceString& instanceNames,
           const odb_SequenceSequenceInt& elementLabels);
```

Required arguments

name An odb_String specifying the name of the set and the repository key.

instanceNames An odb_SequenceString specifying the namespaces for *elementLabels*. The string could be an empty string to denote the elementLabels at the assembly level.

elementLabels An odb_SequenceSequenceInt specifying the element labels.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

Surface(...)

This method creates a surface set.

Path

odb.parts()[*name*].Surface

odb.rootAssembly().instances()[*name*].Surface

Prototype

```
odb_Set&
Surface(const odb_String& name,
        const odb_SequenceElement& elements,
        const odb_SequenceElementFace& faces);
```

Required arguments

name An odb_String specifying the name of the set and the repository key.

elements An odb_SequenceElement specifying the elements in the surface.

faces An odb_SequenceElementFace specifying the element faces.

The possible face enumerations depend on the type of element, as described in the following table:

Element type	Face enumerations
Solid elements	FACE1, FACE2, FACE3, FACE4, FACE5, FACE6
Three-dimensional shell elements	SIDE1, SIDE2
Two-dimensional elements	FACE1, FACE2, FACE3, FACE4
Wire elements	END, END2

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

Surface(...)

This method creates a surface set.

Path

odb.parts()[*name*].Surface

odb.rootAssembly().instances()[*name*].Surface

Prototype

```
odb_Set&
Surface(const odb_String& name,
        const odb_SequenceInt& elementLabels,
        const odb_SequenceElementFace& faces);
```

Required arguments

name	An odb_String specifying the name of the set and the repository key.
elementLabels	An odb_SequenceInt specifying the element labels.
faces	An odb_SequenceElementFace specifying the element faces. An odb_SequenceSequenceElementFace is required for an assembly set.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

Surface(...)

This method creates a surface set.

Path

odb.rootAssembly().Surface

Prototype

```
odb_Set&
Surface(const odb_String& name,
        const odb_SequenceString& instanceNames,
        const odb_SequenceSequenceInt& elementLabels,
        const odb_SequenceSequenceElementFace& faces);
```

Required arguments

name	An odb_String specifying the name of the set and the repository key.
instanceNames	An odb_SequenceString specifying the namespaces for *elementLabels*. This member is only valid for assembly sets. The string could be an empty string to denote the elementLabels at the assembly level.
elementLabels	An odb_SequenceSequenceInt specifying the element labels.
faces	An odb_SequenceSequenceElementFace specifying the element faces.

Optional arguments

None.

Return value

An OdbSet object.

Exceptions

None.

Members

The OdbSet object can have the following members:

Prototype

```
odb_String name() const;
               odb_SequenceString instanceNames();
               odb_Node nodes(int index);
               const odb_SequenceNode& nodes();
               const odb_SequenceNode& nodes(const odb_String& \
                   instanceName); // For an assembly level set
               odb_Element elements(int index);
               const odb_SequenceElement& elements();
               const odb_SequenceElement& elements(const odb_String& \
                   instanceName); // For an assembly level set
               odb_Enum::odb_ElementFaceEnum faces(int index);
               const odb_SequenceElementFace& faces() ;
               const odb_SequenceElementFace& faces(const odb_String& \
```

```
                    instanceName); // For an assembly level set
                odb_Enum::odb_SetTypeEnum type() const;
```

name
An odb_String specifying the name of the set and the repository key.

instanceNames
An odb_SequenceString specifying the namespaces for the nodes, elements, and faces constituting the set.

nodes
A sequence of *OdbMeshNode* objects.

elements
A sequence of *OdbMeshElement* objects.

faces
An odb_SequenceElementFace specifying the element face.

OdbStep object

An output database contains the same steps of the model database that originated it.

Access

```
odb.steps()[name]
```

Step(...)

This method creates an OdbStep object.

Path

```
odb.Step
```

Prototype

```
odb_Step&
Step(const odb_String& name,
     const odb_String& description,
     odb_Enum::odb_DomainEnum domain,
     double timePeriod,
     const odb_String& previousStepName,
     const odb_String& procedure,
     double totalTime);
```

Required arguments

name

An odb_String specifying the repository key.

description

An odb_String specifying the step description.

domain

An odb_Enum::odb_DomainEnum specifying the domain of the step. Possible values are odb_Enum::TIME, odb_Enum::FREQUENCY, odb_Enum::ARC_LENGTH, and odb_Enum::MODAL.

The type of *OdbFrame* object that can be created for this step is based on the value of the *domain* argument.

Optional arguments

timePeriod

A Double specifying the time period of the step. *timePeriod* is required if *domain*=odb_Enum::TIME; otherwise, this argument is not applicable. The default value is 0.0.

previousStepName

An odb_String specifying the preceding step. If *previousStepName* is the empty string, the last step in the repository is used. If *previousStepName* is not the last step, this will result in a change to the *previousStepName* member of the step that was in that position. A special value 'Initial' refers to the internal initial model step and might be used exclusively for inserting a new step at the first position before any other existing steps. The default value is an empty string.

procedure An odb_String specifying the step procedure. The default value is an empty string.

totalTime A Double specifying the analysis time spend in all the steps previous to this step. The default value is −1.0.

nlgeom A Boolean specifying the *nlgeom* value for all steps. The default value is true.

Return value

An OdbStep object.

Exceptions

If *previousStepName* is invalid: ValueError: previousStepName is invalid.

`getFrame(...)`

This method retrieves an *OdbFrame* object associated with a given frame value.

Prototype

```
odb_Frame
getFrame(double frameValue,
         odb_Enum::odb_MatchEnum match);
```

Required arguments

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the step time or frequency.

Optional arguments

match An odb_Enum::odb_MatchEnum specifying which frame to return if there is no frame at the exact frame value. Possible values are odb_Enum::CLOSEST, odb_Enum::BEFORE, odb_Enum::AFTER, and odb_Enum::EXACT. The default value is odb_Enum::CLOSEST.

When *match*=odb_Enum::CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=odb_Enum::EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

getFrame(...)

This method retrieves an *OdbFrame* object associated with a given load case.

Prototype

```
odb_Frame
getFrame(const odb_LoadCase& loadCase);
```

Required arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

Optional arguments

None.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

getFrame(...)

This method retrieves an *OdbFrame* object associated with a given load case and frame value.

Prototype

```
odb_Frame
getFrame(const odb_LoadCase& loadCase,
         double frameValue,
         odb_Enum::odb_MatchEnum match);
```

Required arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

frameValue A Double specifying the value at which the frame is required. *frameValue* can be the step time or frequency.

Optional arguments

match An odb_Enum::odb_MatchEnum specifying which frame to return if there is no frame at the exact frame value. Possible values are odb_Enum::CLOSEST, odb_Enum::BEFORE, odb_Enum::AFTER, and odb_Enum::EXACT. The default value is odb_Enum::CLOSEST.

When *match*=odb_Enum::CLOSEST, Abaqus returns the closest frame. If the frame value requested is exactly halfway between two frames, Abaqus returns the frame after the value.

When *match*=odb_Enum::EXACT, Abaqus raises an exception if the exact frame value does not exist.

Return value

An *OdbFrame* object.

Exceptions

If the *OdbFrame* object is not found: OdbError: Frame not found.

getHistoryRegion(...)

This method retrieves a *HistoryRegion* object associated with a HistoryPoint in the model.

Prototype

```
odb_HistoryRegion
getHistoryRegion(const odb_HistoryPoint& point,
                 const odb_LoadCase& loadCase);
```

Required arguments

point A *HistoryPoint* object specifying the point in the model.

Optional arguments

loadCase An *OdbLoadCase* object specifying a load case in the step.

Return value

A *HistoryRegion* object.

Exceptions

If a *HistoryRegion* object is not found: OdbError: HistoryRegion not found.

setDefaultDeformedField(...)

This method sets the default deformed field variable in a step.

Prototype

```
void
setDefaultDeformedField(const odb_FieldOutput& field);
```

Required arguments

field A *FieldOutput* object specifying the default deformed field variable for visualization.

Optional arguments

None.

Return value

None.

Exceptions

None.

setDefaultField(...)

This method sets the default field variable in a step.

Prototype

```
void
setDefaultField(const odb_FieldOutput& field);
```

Required arguments

field A *FieldOutput* object specifying the default field variable for visualization.

Optional arguments

None.

Return value

None.

Exceptions

None.

Members

The OdbStep object has members with the same names and descriptions as the arguments to the *Step* method.

In addition, the OdbStep object can have the following members:

Prototype

```
odb_String name() const;
odb_String description() const;
odb_Enum::odb_DomainEnum domain() const;
const odb_LoadCaseRepository& loadCases();
float timePeriod() const;
float totalTime() const;
odb_String previousStepName() const;
odb_String procedure() const;
int number() const;
bool nlgeom() const;
odb_Frame frames(int frameNo);
odb_SequenceFrame& frames();
odb_HistoryRegionRepository& historyRegions();
```

```
double mass() const;
odb_SequenceDouble massCenter() const;
odb_SequenceDouble inertiaAboutCenter() const;
odb_SequenceDouble inertiaAboutOrigin() const;
double acousticMass() const;
odb_SequenceDouble acousticMassCenter() const;
odb_SequenceNodalDofs eliminatedNodalDofs() const;
```

number
An Int specifying the step number.

nlgeom
A Boolean specifying whether geometric nonlinearity can occur in this step.

mass
A Double specifying the current value of the mass of the model. This does not include the mass of the acoustic media if any present.

acousticMass
A Double specifying the current value of the mass of the acoustic media of the model.

frames
A sequence of *OdbFrame* objects.

historyRegions
A repository of *HistoryRegion* objects.

loadCases
A repository of *OdbLoadCase* objects.

massCenter
An odb_SequenceDouble specifying the coordinates of the center of mass.

inertiaAboutCenter
An odb_SequenceDouble specifying the moments and products of inertia about the center of mass. For 3-D models inertia quantities are written in the following order: I(XX), I(YY), I(ZZ), I(XY), I(XZ), and I(YZ). For 2-D models only I(ZZ) and I(XY) are outputted.

inertiaAboutOrigin
An odb_SequenceDouble specifying the moments and products of inertia about the origin of the global coordinate system. For 3-D models inertia quantities are written in the following order: I(XX), I(YY), I(ZZ), I(XY), I(XZ), and I(YZ). For 2-D models only I(ZZ) and I(XY) are outputted.

acousticMassCenter
An odb_SequenceDouble specifying the coordinates of the center of mass of the acoustic media.

RebarOrientation object

The RebarOrientation object represents the orientation of the rebar reference.

Access

```
odb.parts()[name].rebarOrientations(i)
odb.rootAssembly().instances()[name].rebarOrientations(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.rebarOrientations(i)
```

Members

The RebarOrientation object can have the following members:

Prototype

```
odb_Set region() const;
odb_DatumCsys csys() const;
odb_Enum::odb_AxisEnum axis() const;
float angle() const;
```

axis

An odb_Enum::odb_AxisEnum specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. Possible values are odb_Enum::AXIS_1, odb_Enum::AXIS_2, and odb_Enum::AXIS_3.

angle

A Float specifying the angle of the additional rotation.

region

An *OdbSet* object specifying a region for which the rebar orientation is defined.

csys

An *OdbDatumCsys* object specifying a datum coordinates system.

SectionCategory object

The SectionCategory object is used to group regions of the model with like sections.
Section definitions that contain the same number of section points or integration points are grouped together.

To access data for a particular section definition, use the individual *Section* objects in the output database. For more information, see *Beam Section profile commands* and *Section commands*.

Access

```
odb.parts()[name].elements(i).sectionCategory()
odb.parts()[name].elementSets()[name].elements(i).sectionCategory()
odb.parts()[name].nodeSets()[name].elements(i).sectionCategory()
odb.parts()[name].surfaces()[name].elements(i).sectionCategory()
odb.rootAssembly().elements(i).sectionCategory()
odb.rootAssembly().elementSets()[name].elements(i).sectionCategory()
odb.rootAssembly().instances()[name].elements(i).sectionCategory()
odb.rootAssembly().instances()[name].elementSets()[name].elements(i)\
.sectionCategory()
odb.rootAssembly().instances()[name].nodeSets()[name].elements(i)\
.sectionCategory()
odb.rootAssembly().instances()[name].surfaces()[name].elements(i)\
.sectionCategory()
odb.rootAssembly().nodeSets()[name].elements(i).sectionCategory()
odb.rootAssembly().surfaces()[name].elements(i).sectionCategory()
odb.sectionCategories()[name]
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elements(i).sectionCategory()
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elementSets()[name].elements(i).sectionCategory()
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodeSets()[name].elements(i).sectionCategory()
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.surfaces()[name].elements(i).sectionCategory()
```

SectionCategory(...)

This method creates a SectionCategory object.

Path

odb.SectionCategory

Prototype

```
odb_SectionCategory& SectionCategory(const odb_String&
name, const odb_String& description);
```

Required arguments

name An odb_String specifying the name of the category.

description An odb_String specifying the description of the category.

Optional arguments

None.

Return value

A SectionCategory object.

Exceptions

None.

Members

The SectionCategory object has members with the same names and descriptions as the arguments to the *SectionCategory* method.

In addition, the SectionCategory object can have the following member:

Prototype

```
odb_String name() const; odb_String description() const; const
odb_SectionPoint& sectionPoints(int index) const; const
odb_SequenceSectionPoint& sectionPoints() const;
```

sectionPoints

A sequence of *SectionPoint* objects.

SectionPoint object

The SectionPoint object describes the location of a section point within a section category.

Access

```
odb.parts()[name].elements(i).sectionCategory().sectionPoints(i)
odb.parts()[name].elementSets()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.parts()[name].nodeSets()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.parts()[name].surfaces()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.rootAssembly().elements(i).sectionCategory().sectionPoints(i)
odb.rootAssembly().elementSets()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.rootAssembly().instances()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.rootAssembly().instances()[name].elementSets()[name].elements(i)\
.sectionCategory().sectionPoints(i)
odb.rootAssembly().instances()[name].nodeSets()[name].elements(i)\
.sectionCategory().sectionPoints(i)
odb.rootAssembly().instances()[name].surfaces()[name].elements(i)\
.sectionCategory().sectionPoints(i)
odb.rootAssembly().nodeSets()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.rootAssembly().surfaces()[name].elements(i).sectionCategory()\
.sectionPoints(i)
odb.sectionCategories()[name].sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].locations(i)\
.sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elements(i).sectionCategory().sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.elementSets()[name].elements(i).sectionCategory().sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.nodeSets()[name].elements(i).sectionCategory().sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.surfaces()[name].elements(i).sectionCategory().sectionPoints(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).sectionPoint()
```

SectionPoint(...)

This method creates a SectionPoint object.

Path

odb.sectionCategories()[*name*].SectionPoint

Prototype

```
odb_SectionPoint& SectionPoint(int number, const
odb_String& description);
```

Required arguments

number	An Int specifying the number of the section point. See *Beam Elements* and *Shell Elements* for the numbering convention.
description	An odb_String specifying the description of the section point.

Optional arguments

None.

Return value

A SectionPoint object.

Exceptions

None.

Members

The SectionPoint object has members with the same names and descriptions as the arguments to the *SectionPoint* method.

Prototype

```
int
number() const; const odb_String description() const;
```

SectorDefinition object

The SectorDefinition object describes the number of symmetry sectors and axis of symmetry for a cyclic symmetry model.

Access

```
odb.sectorDefinition()
```

Members

The SectorDefinition object has the following members:

Prototype

```
int numSectors();
const odb_SequenceSequenceFloat symmetryAxis();
```

numSectors

An Int specifying the number of sectors in the cyclic symmetry model.

symmetryAxis

An odb_SequenceSequenceFloat specifying the coordinates of two points on the axis of symmetry.

UserData object

The UserData object contains user-defined XY data. The UserData object has no constructor; it is created automatically when an *Odb* object is created.

Access

```
odb.userData()
```

XYData(...)

This method creates an XYData object from a sequence of *X*–*Y* data pairs.

Path

odb.userData().XYData

Prototype

```
odb_UserData&
XYData(const odb_String& name,
       const odb_SequenceSequenceDouble& data,
       const odb_String& sourceDescription,
       const odb_String& contentDescription,
       const odb_String& positionDescription,
       const odb_String& legendLabel,
       const odb_String& xValuesLabel,
       const odb_String& yValuesLabel,
       const odb_QuantityType& axis1QuantityType,
       const odb_QuantityType& axis2QuantityType);
```

Required arguments

name
An odb_String specifying the repository key.

data
An odb_SequenceSequenceDouble specifying the *X*–*Y* data pairs.

Optional arguments

sourceDescription
An odb_String specifying the source of the *X*–*Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription
An odb_String specifying the content of the *X*–*Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription
An odb_String specifying additional information about the *X*–*Y* data (e.g., “for whole model”). The default value is an empty string.

legendLabel
An odb_String specifying the label to be used in the legend. The default value is the name of the XYData object.

xValuesLabel An odb_String specifying the label for the X-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

yValuesLabel An odb_String specifying the label for the Y-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

axis1QuantityType A QuantityType object specifying the QuantityType object associated to the X -axis1- values.

axis2QuantityType A QuantityType object specifying the QuantityType object associated to the Y -axis2- values.

Return value

An XYData object.

Exceptions

InvalidNameError.

Members

The UserData object can have the following members:

Prototype

```
odb_UserXYDataRepository& xyDataObjects();
odb_UserXYData& xyDataObjects(const odb_String& xyName);
```

name An odb_String specifying the repository key.

sourceDescription An odb_String specifying the source of the *X–Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription An odb_String specifying the content of the *X–Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription An odb_String specifying additional information about the *X–Y* data (e.g., “for whole model”). The default value is an empty string.

xValuesLabel An odb_String specifying the label for the X-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

yValuesLabel An odb_String specifying the label for the Y-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

axis1QuantityType A QuantityType object specifying the QuantityType object associated to the X -axis1- values.

axis2QuantityType

A QuantityType object specifying the QuantityType object associated to the Y -axis2-values.

legendLabel

An odb_String specifying the label to be used in the legend. The default value is the name of the XYData object.

xyDataObjects

A repository of *UserXYData* objects.

data

An odb_SequenceSequenceDouble specifying the *X*–*Y* data pairs.

UserXYData object

The UserXYData object stores a sequence of pairs and information about the axes.

Access

```
Api.userData().xyDataObjects()[name]
```

addData(...)

This method replaces the contents of the *data* member of the *XYData(...)* object.

Prototype

```
void
addData(const odb_SequenceSequenceDouble& data);
```

Required arguments

data An odb_SequenceSequenceDouble specifying the *X*–*Y* data pairs.

Optional arguments

None.

Return value

None.

Exceptions

None.

getData()

Returns the contents of the *data* member of the *XYData(...)* object.

Prototype

```
odb_SequenceSequenceFloat
getData();
```

Arguments

None.

Return value

An odb_SequenceSequenceFloat.

Exceptions

None.

Members

The UserXYData object can have the following members:

Prototype

```
odb_String name() const;
odb_String sourceDescription() const;
odb_String contentDescription() const;
odb_String positionDescription() const;
odb_String xAxisLabel() const;
odb_String yAxisLabel() const;
odb_String legendLabel() const;
odb_String description() const;
odb_SequenceSequenceDouble data() const;
```

name
An odb_String specifying the repository key.

sourceDescription
An odb_String specifying the source of the *X–Y* data (e.g., “Entered from keyboard”, “Taken from ASCII file”, “Read from an ODB”, etc.). The default value is an empty string.

contentDescription
An odb_String specifying the content of the *X–Y* data (e.g., “field 1 vs. field 2”). The default value is an empty string.

positionDescription
An odb_String specifying additional information about the *X–Y* data (e.g., “for whole model”). The default value is an empty string.

xAxisLabel
An odb_String specifying the label for the X-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

yAxisLabel
An odb_String specifying the label for the Y-values. This value may be overridden if the *X–Y* data are combined with other *X–Y* data. The default value is an empty string.

legendLabel
An odb_String specifying the label to be used in the legend. The default value is the name of the XYData object.

description
An odb_String specifying the complete description of the XYData object.

data
An odb_SequenceSequenceDouble specifying the *X–Y* data pairs.

Top level commands

The Odb commands do the following:

- Determine if an output database (`.odb`) file needs to be upgraded to the current release.
- Determine the extreme value for an output variable over a number of fields; for example, over a number of load cases.
- Open an existing output database file and create a new Odb object.
- Upgrade an output database file to the current release and write the upgraded output database to a new file.
- Fetch the active output database from inside a C++ user subroutine during an analysis run.
- Determine the actual subtype of a base Section, Profile, or Amplitude object.
- Cast a base Section, Profile, or Amplitude object to the proper subtype.

isUpgradeRequiredForOdb

This method determines if an output database file needs to be upgraded to the current release.

Prototype

```
bool isUpgradeRequiredForOdb(const odb_String& upgradeRequiredOdbPath);
```

Required arguments

upgradeRequiredOdbPath	An odb_String specifying the path to an output database file to test. The test determines if the output database needs to be upgraded to the current release.

Optional arguments

None.

Return value

A Boolean indicating the result of the test. A value of True indicates that the output database needs to be upgraded to the current release.

Exceptions

None.

maxEnvelope

Retrieve the maximum value of an output variable over a number of fields.

Prototype

```
odb_SequenceFieldOutput maxEnvelope(
    const odb_SequenceFieldOutput & inputFields);
odb_SequenceFieldOutput maxEnvelope(
    const odb_SequenceFieldOutput & inputFields,
    odb_Enum::odb_InvariantEnum invariant);
odb_SequenceFieldOutput maxEnvelope(
```

```
    const odb_SequenceFieldOutput & inputFields,
    const odb_String& componentLabel);
```

Required arguments

Argument
An odb_SequenceFieldoutput object containing all the fieldOutput objects from which the maximum value will be computed.
A SymbolicConstant specifying the invariant or component label to be used when comparing vectors or tensors. Possible values are • odb_Enum::MAGNITUDE • odb_Enum::MISES • odb_Enum::TRESCA • odb_Enum::PRESS • odb_Enum::INV3 • odb_Enum::MAX_PRINCIPAL • odb_Enum::MID_PRINCIPAL • odb_Enum::MIN_PRINCIPAL You must provide either this argument or the following argument if the field is a vector or tensor.
An odb_String specifying the component of the tensor or vector to be used for selecting the maximum value.

Optional arguments

None.

Return value

An odb_SequenceFieldOutput object. The first fieldOutput object contains the maximum value. The second fieldOutput object contains the index of the field containing the maximum value. The index follows the order in which fields are positioned in the list of fieldOutput objects provided as the argument to the function.

Exceptions

OdbError

TypeError This function takes no keyword arguments.

minEnvelope

Retrieve the minimum value of an output variable over a number of fields.

Prototype

```
odb_SequenceFieldOutput minEnvelope(
    const odb_SequenceFieldOutput & inputFields);
odb_SequenceFieldOutput minEnvelope(
    const odb_SequenceFieldOutput & inputFields,
    odb_Enum::odb_InvariantEnum invariant);
odb_SequenceFieldOutput minEnvelope(
    const odb_SequenceFieldOutput & inputFields,
    const odb_String& componentLabel);
```

Required arguments

Argument
An odb_SequenceFieldoutput object containing all the fieldOutput objects from which the maximum value will be computed.
A SymbolicConstant specifying the invariant or component label to be used when comparing vectors or tensors. Possible values are • odb_Enum::MAGNITUDE • odb_Enum::MISES • odb_Enum::TRESCA • odb_Enum::PRESS • odb_Enum::INV3 • odb_Enum::MAX_PRINCIPAL • odb_Enum::MID_PRINCIPAL • odb_Enum::MIN_PRINCIPAL You must provide either this argument or the following argument if the field is a vector or tensor.
An odb_String specifying the component of the tensor or vector to be used for selecting the minimum value.

Optional arguments

None.

Return value

An odb_SequenceFieldOutput object. The first fieldOutput object contains the minimum value. The second fieldOutput object contains the index of the field containing the minimum value. The index follows the order in which fields are positioned in the list of fieldOutput objects provided as the argument to the function.

Exceptions

OdbError

TypeError This function takes no keyword arguments.

openOdb

This method opens an existing output database (`.odb`) file and creates a new Odb object.

Prototype

```
odb_Odb& openOdb(const odb_String& path);
```

Required arguments

path A odb_String specifying the path to an existing output database (`.odb`) file.

Optional arguments

readOnly A Boolean specifying whether the file will permit only read access or both read and write access. The initial value is False, indicating that both read and write access will be permitted.

readInternalSets A Boolean specifying whether the file will permit access to sets specified as Internal in the database. The initial value is False, indicating that internal sets will not be read.

Return value

An Odb object.

Exceptions

If the output database was generated by a previous release of Abaqus and needs upgrading:	OdbError: The database is from a previous release of Abaqus. Run `abaqus upgrade -job <newFilename> -odb <oldFileName>` to upgrade it.
If the output database was generated by a newer release of Abaqus, and the installation of Abaqus needs upgrading:	OdbError: Abaqus installation must be upgraded before this output database can be opened.

openOdb

This method opens an existing output database (`.odb`) file and creates a new Odb object.

Prototype

```
odb_Odb& openOdb(const odb_String& name,
const odb_String& path, bool readOnly, bool checkPhase, bool
readInternalSets, STRMODE_DECL);
```

Required arguments

name A odb_String specifying the repository key.

Optional arguments

path A odb_String specifying the path to an existing output database (`.odb`) file.

readOnly A Boolean specifying whether the file will permit only read access or both read and write access. The initial value is False, indicating that both read and write access will be permitted.

checkPhase A Boolean specifying whether an error message will be printed if another process is modifying the database. The initial value is True, indicating an error message will be printed.

readInternalSets
A Boolean specifying whether the file will permit access to sets specified as Internal in the database. The initial value is False, indicating that internal sets will not be read.

STRMODE_DECL
A constant specifying whether the current locale encoding should be retained. If the ODB path uses only ASCII characters or if the path is known to be encoded in UTF-8, this argument need not be defined.

Return value

An Odb object.

Exceptions

If the output database was generated by a previous release of Abaqus and needs upgrading:	OdbError: The database is from a previous release of Abaqus. Run `abaqus upgrade -job <newFilename> -odb <oldFileName>` to upgrade it.
If the output database was generated by a newer release of Abaqus, and the installation of Abaqus needs upgrading:	OdbError: Abaqus installation must be upgraded before this output database can be opened.
If the file is not a valid database:	AbaqusException: Cannot open file <*filename*>.

upgradeOdb

This method upgrades an existing Odb object to the current release and writes the upgraded version of the Odb object to a file. In addition, Abaqus/CAE writes information about the status of the upgrade to a log (`*.log`) file.

Prototype

```
void upgradeOdb(const odb_String& existingOdbPath,
const odb_String& upgradedOdbPath);
```

Required arguments

existingOdbPath
An odb_String specifying the path to the file containing the output database to be upgraded.

upgradedOdbPath
An odb_String specifying the path to the file that will contain the upgraded output database.

Optional arguments

None.

Return value

`None.`

Exceptions

If the output database upgrade fails:	OdbError: cannot convert database

getActiveOdb

This method is intended to be used inside an Abaqus/Standard or Abaqus/Explicit user subroutine written in C++. It fetches the object that is in use by the analysis product and enables a user to modify it inside the user subroutine. This method will throw an exception if more than one output database is currently open.

Prototype

```
odb_Odb& getActiveOdb();
```

Arguments

None.

Return value

An object.

Exceptions

If more than one output database file is open:	OdbError: The getActiveOdb method can only be called if a single odb file is open.
If no output database files are open:	OdbError: No odb files are open.

odb_IsA

This method takes a type and a base object and returns a boolean indicating whether the subtype of the supplied base object matches the type argument. This method is useful when fetching base Section, Profile, or Amplitude objects from their containers. If, for example, the user wants to determine if the base odb_Section object returned by the odb_SectionContainer is actually of type odb_BeamSection, he would call this method with the odb_BeamSection as the type and the odb_Section object as the base object.

Prototype

```
bool odb_IsA(TYPE, const BASETYPE& obj);
```

Required arguments

TYPE	The potential type of the base class object.
obj	The base object being tested for the TYPE.

Optional arguments

None.

Return value

A Boolean.

Exceptions

None.

odb_dynamicCast

This method casts a supplied Section, Profile, or Amplitude to the specified subtype. This method is useful for retrieving the base objects out of the appropriate containers and casting them to the subtype to access the data members for that particular subtype. This method will throw an odb_BaseException if the cast is not possible.

Prototype

```
const TYPE& returnObj = odb_dynamicCast(TYPE, const BASETYPE& obj);
```

Required arguments

TYPE	The potential type of the base class object.
obj	The base object being case to TYPE.

Optional arguments

None.

Return value

An object of type TYPE.

Exceptions

If the cast is not possible:	Bad cast: From BASETYPE To TYPE.

Property and Section commands

The Property commands are used to create and manage reinforcements and to assign properties to a part or part instance. The Section commands are used to create sections and profiles with their associated properties and behavior.
The various section objects are all derived from the Section object. The various profile objects are all derived from the Profile object. See *Property commands* for the property assignment commands.

In this section:

- *SectionAssignment object*
- *Section object*
- *AcousticInfiniteSection object*
- *AcousticInterfaceSection object*
- *BeamSection object*
- *CohesiveSection object*
- *CompositeShellSection object*
- *CompositeSolidSection object*
- *ConnectorSection object*
- *EulerianSection object*
- *GasketSection object*
- *GeneralStiffnessSection object*
- *GeometryShellSection object*
- *HomogeneousShellSection object*
- *HomogeneousSolidSection object*
- *LayerProperties object*
- *MembraneSection object*
- *MPCSection object*
- *PEGSection object*
- *RebarLayers object*
- *SectionLayer object*
- *ShellSection object*
- *SolidSection object*
- *SurfaceSection object*
- *TransverseShearBeam object*
- *TransverseShearShell object*
- *TrussSection object*

SectionAssignment object

The SectionAssignment object is used to specify a section assignment on an assembly or part. Section assignments on the assembly are limited to connector elements only.

Access

```
odb.parts()[name].sectionAssignments(i)
odb.rootAssembly().instances()[name].sectionAssignments(i)
odb.rootAssembly().sectionAssignments(i)
odb.steps()[name].frames(i).fieldOutputs()[name].values(i).instance()\
.sectionAssignments(i)
```

Members

The SectionAssignment object has members with the same names and descriptions as the arguments to the *SectionAssignment* method.

In addition, the SectionAssignment object has the following member:

Prototype

```
odb_Set region() const;
           odb_String sectionName() const;
           double offset() const;
```

suppressed

A Boolean specifying whether the section assignment is suppressed or not. The default value is false.

Section object

The Section object defines the properties of a section. The Section object is the abstract base type for other Section objects. The Section object has no explicit constructor. The methods and members of the Section object are common to all objects derived from the Section.

Access

```
sectionApi.sections()[name]
```

Members

The Section object has the following member:

Prototype

```
odb_String name() const;
```

name An odb_String specifying the repository key.

AcousticInfiniteSection object

The AcousticInfiniteSection object defines the properties of an acoustic section.

The AcousticInfiniteSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

AcousticInfiniteSection(...)

This method creates an AcousticInfiniteSection object.

Path

```
sectionApi.AcousticInfiniteSection
```

Prototype

```
odb_AcousticInfiniteSection&
AcousticInfiniteSection(const odb_String& name,
                        const odb_String& material,
                        double thickness,
                        int order);
```

Required arguments

name An odb_String specifying the repository key.

material An odb_String specifying the name of the material.

Optional arguments

thickness A Double specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

order An Int specifying the number of ninth-order polynomials that will be used to resolve the variation of the acoustic field in the infinite direction. Possible values are 0 <*order*≤ 10. The default value is 10.

Return value

An AcousticInfiniteSection object.

Exceptions

InvalidNameError and RangeError.

Members

The AcousticInfiniteSection object has members with the same names and descriptions as the arguments to the *AcousticInfiniteSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

AcousticInterfaceSection object

The AcousticInterfaceSection object defines the properties of an acoustic section.

The AcousticInterfaceSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

AcousticInterfaceSection(...)

This method creates an AcousticInterfaceSection object.

Path

```
sectionApi.AcousticInterfaceSection
```

Prototype

```
odb_AcousticInterfaceSection&
AcousticInterfaceSection(const odb_String& name,
                         double thickness);
```

Required arguments

name

An odb_String specifying the repository key.

Optional arguments

thickness

A Double specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

Return value

An AcousticInterfaceSection object.

Exceptions

InvalidNameError and RangeError.

Members

The AcousticInterfaceSection object has members with the same names and descriptions as the arguments to the *AcousticInterfaceSection* method.

Corresponding analysis keywords

- **INTERFACE*

BeamSection object

The BeamSection object defines the properties of a beam section.

The BeamSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

BeamSection(...)

This method creates a BeamSection object.

Path

```
sectionApi.BeamSection
```

Prototype

```
odb_BeamSection&
BeamSection(const odb_String& name,
            const odb_String& integration,
            const odb_String& profile,
            double poissonRatio,
            bool thermalExpansion,
            bool temperatureDependency,
            int dependencies,
            odb_Union density,
            odb_Union referenceTemperature,
            const odb_String& temperatureVar,
            double alphaDamping,
            double betaDamping,
            double compositeDamping,
            bool useFluidInertia,
            const odb_String& submerged,
            odb_Union fluidMassDensity,
            odb_Union crossSectionRadius,
            double lateralMassCoef,
            double axialMassCoef,
            double massOffsetX,
            double massOffsetY,
            const odb_String& material,
            const odb_SequenceSequenceDouble& table,
            const odb_SequenceSequenceDouble& outputPts,
            const odb_SequenceDouble& centroid,
            const odb_SequenceDouble& shearCenter);
```

Required arguments

name
An odb_String specifying the repository key.

integration
An odb_String specifying the integration method for the section. Possible values are "BEFORE_ANALYSIS" and "DURING_ANALYSIS".

profile
An odb_String specifying the name of the profile.

Optional arguments

poissonRatio
A Double specifying the Poisson's ratio of the section. The default value is 0.0.

thermalExpansion
A Boolean specifying whether to use thermal expansion data. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

density
The string "NONE" or a Double specifying the density of the section. The default value is "NONE".

referenceTemperature
The string "NONE" or a Double specifying the reference temperature of the section. The default value is "NONE".

temperatureVar
An odb_String specifying the temperature variation for the section. Possible values are "LINEAR" and "INTERPOLATED". The default value is "LINEAR".

alphaDamping
A Double specifying the α_R factor to create mass proportional damping in direct-integration dynamics. The default value is 0.0.

betaDamping
A Double specifying the β_R factor to create stiffness proportional damping in direct-integration dynamics. The default value is 0.0.

compositeDamping
A Double specifying the fraction of critical damping to be used in calculating composite damping factors for the modes (for use in modal dynamics). The default value is 0.0.

useFluidInertia
A Boolean specifying whether added mass effects will be simulated. The default value is false.

submerged
An odb_String specifying whether the section is either full submerged or half submerged. This argument applies only when *useFluidInertia* = True. Possible values are "FULLY" and "HALF". The default value is "FULLY".

fluidMassDensity
The string "NONE" or a Double specifying the mass density of the fluid. This argument applies only when *useFluidInertia* = True and must be specified in that case. The default value is "NONE".

crossSectionRadius
The string "NONE" or a Double specifying the radius of the cylindrical cross-section. This argument applies only when *useFluidInertia* = True and must be specified in that case. The default value is "NONE".

lateralMassCoef
A Double specifying the added mass coefficient, C_A, for lateral motions of the beam. This argument applies only when*useFluidInertia* = True. The default value is 1.0.

axialMassCoef

A Double specifying the added mass coefficient, $C_{(A-E)}$, for motions along the axis of the beam. This argument affects only the term added to the free end(s) of the beam, and applies only when *useFluidInertia* = True. The default value is 0.0.

massOffsetX

A Double specifying the local 1-coordinate of the center of the cylindrical cross-section with respect to the beam cross-section. This argument applies only when *useFluidInertia* = True. The default value is 0.0.

massOffsetY

A Double specifying the local 2-coordinate of the center of the cylindrical cross-section with respect to the beam cross-section. This argument applies only when *useFluidInertia* = True. The default value is 0.0.

material

An odb_String specifying the name of the material. The default value is an empty string. The material is required when *integration* is "DURING_ANALYSIS".

table

An odb_SequenceSequenceDouble specifying the items described below. The default value is an empty sequence.

outputPts

An odb_SequenceSequenceDouble specifying the positions at which output is requested. The default value is an empty sequence.

centroid

An odb_SequenceDouble specifying the *X*–*Y* coordinates of the centroid. The default value is (0.0, 0.0).

shearCenter

An odb_SequenceDouble specifying the *X*–*Y* coordinates of the shear center. The default value is (0.0, 0.0).

Table data

The table data specify the following:

- E, the Young's modulus of the section.
- G, the torsional shear modulus of the section.
- Thermal expansion coefficient, if using thermal expansion.
- Temperature, if the data depend on temperature.
- Value of the first field variable, if the data depend on field variables.
- Value of the second field variable.
- Etc.

Return value

A BeamSection object.

Exceptions

None.

Members

The BeamSection object has members with the same names and descriptions as the arguments to the *BeamSection* method.

In addition, the BeamSection object can have the following member:

Prototype

```
odb_TransverseShearBeam beamTransverseShear() const;
```

beamTransverseShear

A *TransverseShearBeam* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **BEAM GENERAL SECTION*
- **BEAM SECTION*
- **BEAM FLUID INERTIA*
- **CENTROID*
- **DAMPING*
- **SHEAR CENTER*
- **SECTION POINTS*

CohesiveSection object

The CohesiveSection object defines the properties of a cohesive section.

The CohesiveSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

CohesiveSection(...)

This method creates a CohesiveSection object.

Path

```
sectionApi.CohesiveSection
```

Prototype

```
odb_CohesiveSection&
CohesiveSection(const odb_String& name,
                const odb_String& response,
                const odb_String& material,
                const odb_String& initialThicknessType,
                double initialThickness,
                odb_Union outOfPlaneThickness);
```

Required arguments

name

An odb_String specifying the repository key.

response

An odb_String specifying the geometric assumption that defines the constitutive behavior of the cohesive elements. Possible values are "TRACTION_SEPARATION", "CONTINUUM", and "GASKET".

material

An odb_String specifying the name of the material.

Optional arguments

initialThicknessType

An odb_String specifying the method used to compute the initial thickness. Possible values are:

- "SOLVER_DEFAULT", specifying that Abaqus will use the analysis product default
- "GEOMETRY", specifying that Abaqus will compute the thickness from the nodal coordinates of the elements.
- "SPECIFY", specifying that Abaqus will use the value given for *initialThickness*

The default value is "SOLVER_DEFAULT".

initialThickness

A Double specifying the initial thickness for the section. The *initialThickness* argument applies only when *initialThicknessType*="SPECIFY". The default value is 1.0.

outOfPlaneThickness The string "NONE" or a Double specifying the out-of-plane thickness for the section. The default value is "NONE".

Return value

A CohesiveSection object.

Exceptions

RangeError and InvalidNameError.

Members

The CohesiveSection object has members with the same names and descriptions as the arguments to the *CohesiveSection* method.

Corresponding analysis keywords

- **COHESIVE SECTION*

CompositeShellSection object

The CompositeShellSection object defines the properties of a composite shell section.

The CompositeShellSection object is derived from the *GeometryShellSection* object.

Access

```
sectionApi.sections()[name]
```

CompositeShellSection(...)

This method creates a CompositeShellSection object.

Path

```
sectionApi.CompositeShellSection
```

Prototype

```
odb_CompositeShellSection& CompositeShellSection(const odb_String& name,
           const kseO_SequenceSectionLayer& layup,
           bool symmetric,
           odb_String layupName,
           odb_String thicknessType,
           bool preIntegrate,
           odb_String poissonDefinition,
           double poisson,
           odb_String integrationRule,
           odb_String temperature,
           odb_String simplification,
           odb_Union nTemp,
           odb_Union thicknessModulus,
           bool useDensity,
           double density,
           odb_String thicknessField,
           odb_String nodalThicknessField);
```

Required arguments

name An odb_String specifying the repository key.

layup A sequence of *SectionLayer* objects specifying the shell cross-section.

Optional arguments

symmetric A Boolean specifying whether or not the layup should be made symmetric by the analysis. The default value is false.

layupName An odb_String specifying the layup name for this section. The default value is an empty string.

thicknessType An odb_String specifying the distribution used for defining the thickness of the elements. Possible values are "UNIFORM", "ANALYTICAL_FIELD", "DISCRETE_FIELD",

"NODAL_ANALYTICAL_FIELD", and "NODAL_DISCRETE_FIELD". The default value is "UNIFORM".

preIntegrate

A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (true) or integrated during the analysis (false). The default value is false.

poissonDefinition

An odb_String specifying whether to use the default value for the Poisson's ratio. Possible values are:

- "DEFAULT", specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- "VALUE", specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is "DEFAULT".

poisson

A Double specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*="VALUE". The default value is 0.5.

integrationRule

An odb_String specifying the shell section integration rule. Possible values are "SIMPSON" and "GAUSS". The default value is "SIMPSON".

temperature

An odb_String specifying the mode used for temperature and field variable input across the section thickness. Possible values are "GRADIENT" and "POINTWISE". The default value is "GRADIENT".

idealization

An odb_String specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to true. Possible values are "NO_IDEALIZATION", "SMEAR_ALL_LAYERS", "MEMBRANE", and "BENDING". The default value is "NO_IDEALIZATION".

nTemp

The string "NONE" or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*="POINTWISE". The default value is "NONE".

thicknessModulus

The string "NONE" or a Double specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is "NONE".

useDensity

A Boolean specifying whether or not to use the value of *density*. The default value is false.

density

A Double specifying the value of density to apply to this section. The default value is 0.0.

thicknessField

An odb_String specifying the name of the AnalyticalField or DiscreteField object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*="ANALYTICAL_FIELD" or *thicknessType*="DISCRETE_FIELD". The default value is an empty string.

nodalThicknessField

An odb_String specifying the name of the AnalyticalField or DiscreteField object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument

applies only when *thicknessType*="NODAL_ANALYTICAL_FIELD" or *thicknessType*="NODAL_DISCRETE_FIELD". The default value is an empty string.

Return value

A CompositeShellSection object.

Exceptions

None.

Members

The CompositeShellSection object has members with the same names and descriptions as the arguments to the *CompositeShellSection* method.

In addition, the CompositeShellSection object can have the following members:

Prototype

```
odb_RebarLayers rebarLayers() const;
odb_TransverseShearShell transverseShear() const;
```

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

transverseShear

A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

CompositeSolidSection object

The CompositeSolidSection object defines the properties of a composite solid section.

The CompositeSolidSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

CompositeSolidSection(...)

This method creates a CompositeSolidSection object.

Path

```
sectionApi.CompositeSolidSection
```

Prototype

```
odb_CompositeSolidSection&
CompositeSolidSection(const odb_String& name,
                      const odb_SequenceSectionLayer& layup,
                      bool symmetric,
                      const odb_String& layupName);
```

Required arguments

name
An odb_String specifying the repository key.

layup
A sequence of *SectionLayer* objects specifying the solid cross-section.

Optional arguments

symmetric
A Boolean specifying whether or not the layup should be made symmetric by the analysis. The default value is false.

layupName
An odb_String specifying the layup name for this section. The default value is an empty string.

Return value

A CompositeSolidSection object.

Exceptions

None.

Members

The CompositeSolidSection object has members with the same names and descriptions as the arguments to the *CompositeSolidSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

ConnectorSection object

A ConnectorSection object describes the connection type and the behavior of a connector.

The ConnectorSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

ConnectorSection(...)

This method creates a ConnectorSection object.

Path

```
sectionApi.ConnectorSection
```

Prototype

```
odb_ConnectorSection&
ConnectorSection(const odb_String& name,
  const odb_String& assembledType,
  const odb_String& rotationalType,
  const odb_String& translationalType,
  const odb_String& integration,
  odb_Union u1ReferenceLength,
  odb_Union u2ReferenceLength,
  odb_Union u3ReferenceLength,
  odb_Union ur1ReferenceAngle,
  odb_Union ur2ReferenceAngle,
  odb_Union ur3ReferenceAngle,
  odb_Union massPerLength,
  odb_Union contactAngle,
  double materialFlowFactor,
  bool regularize,
  bool defaultTolerance,
  double regularization,
  const odb_String& extrapolation,
  const odb_SequenceConnectorBehaviorOption& behaviorOptions);
```

Required arguments

name

An odb_String specifying the repository key.

Optional arguments

assembledType

An odb_String specifying the assembled connection type. Possible values are:

- "NONE"
- "BEAM"
- "BUSHING"
- "CVJOINT"
- "CYLINDRICAL"

- "HINGE"
- "PLANAR"
- "RETRACTOR"
- "SLIPRING"
- "TRANSLATOR"
- "UJOINT"
- "WELD"

The default value is "NONE".

You cannot include the *assembledType* argument if *translationalType* or *rotationalType* are given a value other than "NONE". At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given a value other than "NONE".

rotationalType

An odb_String specifying the basic rotational connection type. Possible values are:

- "NONE"
- "ALIGN"
- "CARDAN"
- "CONSTANT_VELOCITY"
- "EULER"
- "FLEXION_TORSION"
- "FLOW_CONVERTER"
- "PROJECTION_FLEXION_TORSION"
- "REVOLUTE"
- "ROTATION"
- "ROTATION_ACCELEROMETER"
- "UNIVERSAL"

The default value is "NONE".

You cannot include the *rotationalType* argument if *assembledType* is given a value other than "NONE". At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given an value other than "NONE".

translationalType

An odb_String specifying the basic translational connection type. Possible values are:

- "NONE"
- "ACCELEROMETER"
- "AXIAL"
- "CARTESIAN"
- "JOIN"
- "LINK"
- "PROJECTION_CARTESIAN"
- "RADIAL_THRUST"

- "SLIDE_PLANE"
- "SLOT"

The default value is "NONE".

You cannot include the *translationalType* argument if *assembledType* is given a value other than "NONE". At least one of the arguments *assembledType*, *translationalType*, or *rotationalType* must be given an value other than "NONE".

integration

An odb_String specifying the time integration scheme to use for analysis. This argument is applicable only to an Abaqus/Explicit analysis. Possible values are "UNSPECIFIED", "IMPLICIT", and "EXPLICIT". The default value is "UNSPECIFIED".

u1ReferenceLength

The string "NONE" or a Double specifying the reference length associated with constitutive response for the first component of relative motion. The default value is "NONE".

u2ReferenceLength

The string "NONE" or a Double specifying the reference length associated with constitutive response for the second component of relative motion. The default value is "NONE".

u3ReferenceLength

The string "NONE" or a Double specifying the reference length associated with constitutive response for the third component of relative motion. The default value is "NONE".

ur1ReferenceAngle

The string "NONE" or a Double specifying the reference angle in degrees associated with constitutive response for the fourth component of relative motion. The default value is "NONE".

ur2ReferenceAngle

The string "NONE" or a Double specifying the reference angle in degrees associated with constitutive response for the fifth component of relative motion. The default value is "NONE".

ur3ReferenceAngle

The string "NONE" or a Double specifying the reference angle in degrees associated with constitutive response for the sixth component of relative motion. The default value is "NONE".

massPerLength

The string "NONE" or a Double specifying the mass per unit reference length of belt material. This argument is applicable only when *assembledType*="SLIPRING", and must be specified in that case. The default value is "NONE".

contactAngle

The string "NONE" or a Double specifying the contact angle made by the belt wrapping around node b. This argument is applicable only to an Abaqus/Explicit analysis, and only when *assembledType*="SLIPRING". The default value is "NONE".

materialFlowFactor

A Double specifying the scaling factor for material flow at node b. This argument is applicable only when *assembledType*="RETRACTOR" or *rotationalType*="FLOW_CONVERTER". The default value is 1.0.

regularize

A Boolean specifying whether or not all tabular data associated with the *behaviorOptions* will be regularized. This argument is applicable only for an Abaqus/Explicit analysis. The default value is true.

defaultTolerance A Boolean specifying whether or not the default regularization tolerance will be used for all tabular data associated with the *behaviorOptions*. This argument is applicable only for an Abaqus/Explicit analysis and only if *regularize*=true. The default value is true.

regularization A Double specifying the regularization increment to be used for all tabular data associated with the *behaviorOptions*. This argument is applicable only for an Abaqus/Explicit analysis and only if *regularize*=true and *defaultTolerance*=false. The default value is 0.03.

extrapolation An odb_String specifying the extrapolation technique to be used for all tabular data associated with the *behaviorOptions*. Possible values are "CONSTANT" and "LINEAR". The default value is "CONSTANT".

behaviorOptions A sequence of *ConnectorBehaviorOption* objects.

Return value

A ConnectorSection object.

Exceptions

InvalidNameError and RangeError.

getVertices()

This method returns a sequence consisting of tuples of coordinates of the connector's endpoints.

Path

```
sectionApi.getVertices
```

Prototype

```
odb_ConnectorSection&
getVertices();
```

Arguments

None.

Return value

A sequence of tuples of floats.

Exceptions

None.

Members

The ConnectorSection object has members with the same names and descriptions as the arguments to the *ConnectorSection* method.

Corresponding analysis keywords

- **CONNECTOR SECTION*
- **CONNECTOR BEHAVIOR*
- **CONNECTOR CONSTITUTIVE REFERENCE*

EulerianSection object

The EulerianSection object defines the properties of an Eulerian section.

The EulerianSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

EulerianSection(...)

This method creates an EulerianSection object.

Path

```
sectionApi.EulerianSection
```

Prototype

```
odb_EulerianSection&
EulerianSection(const odb_String& name,
                const odb_StringRepository& data);
```

Required arguments

name An odb_String specifying the repository key.

data A repository of String objects specifying a dictionary mapping Material instance names to Material names. Internally the specified mapping gets sorted on Material instance name.

Optional arguments

None.

Return value

An EulerianSection object.

Exceptions

None.

Members

The EulerianSection object has members with the same names and descriptions as the arguments to the *EulerianSection* method.

Corresponding analysis keywords

- **EULERIAN SECTION*

GasketSection object

The GasketSection object defines the properties of a gasket section.

The GasketSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

GasketSection(...)

This method creates a GasketSection object.

Path

```
sectionApi.GasketSection
```

Prototype

```
odb_GasketSection&
GasketSection(const odb_String& name,
              const odb_String& material,
              double crossSection,
              double initialGap,
              odb_Union initialThickness,
              double initialVoid,
              odb_Union stabilizationStiffness);
```

Required arguments

name
An odb_String specifying the repository key.

material
An odb_String specifying the name of the material of which the gasket is made or material that defines gasket behavior.

Optional arguments

crossSection
A Double specifying the cross-sectional area, width, or out-of-plane thickness, if applicable, depending on the gasket element type. The default value is 1.0.

initialGap
A Double specifying the initial gap. The default value is 0.0.

initialThickness
The string "DEFAULT" or a Double specifying the initial gasket thickness. If "DEFAULT" is specified, the initial thickness is determined using nodal coordinates. The default value is "DEFAULT".

initialVoid
A Double specifying the initial void. The default value is 0.0.

stabilizationStiffness
The string "DEFAULT" or a Double specifying the default stabilization stiffness used in all but link elements to stabilize gasket elements that are not supported at all nodes, such as those that extend outside neighboring components. If "DEFAULT" is specified, a value

is used equal to 10^{-9} times the initial compressive stiffness in the thickness direction. The default value is "DEFAULT".

Return value

A GasketSection object.

Exceptions

InvalidNameError and ValueError.

Members

The GasketSection object has members with the same names and descriptions as the arguments to the *GasketSection* method.

Corresponding analysis keywords

- **GASKET SECTION*

GeneralStiffnessSection object

The GeneralStiffnessSection object defines the properties of a shell section via the stiffness matrix.

The GeneralStiffnessSection object is derived from the *ShellSection* object.

Access

```
sectionApi.sections()[name]
```

GeneralStiffnessSection(...)

This method creates a GeneralStiffnessSection object.

Path

```
sectionApi.GeneralStiffnessSection
```

Prototype

```
odb_GeneralStiffnessSection&
GeneralStiffnessSection(const odb_String& name,
                const odb_SequenceDouble& stiffnessMatrix,
                odb_Union referenceTemperature,
                bool applyThermalStress,
                bool temperatureDependency,
                int dependencies,
                const odb_String& poissonDefinition,
                double poisson,
                bool useDensity,
                double density,
                const odb_SequenceDouble& thermalStresses,
                const odb_SequenceSequenceDouble& scalingData);
```

Required arguments

name
An odb_String specifying the repository key.

stiffnessMatrix
An odb_SequenceDouble specifying the stiffness matrix for the section in the order D11, D12, D22, D13, D23, D33, ..., D66. Twenty-one entries must be given.

Optional arguments

referenceTemperature
The string "NONE" or a Float specifying the reference temperature for thermal expansion. The default value is "NONE".

applyThermalStress
A Boolean specifying whether or not the section stiffness varies with thermal stresses. The default value is false.

temperatureDependency
A Boolean specifying whether the data depend on temperature. The default value is false.

dependencies
An Int specifying the number of field variable dependencies. The default value is 0.

poissonDefinition An odb_String specifying whether to use the default value for the Poisson's ratio. Possible values are:

- "DEFAULT", specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- "VALUE", specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is "DEFAULT".

poisson A Double specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*="VALUE". The default value is 0.5.

useDensity A Boolean specifying whether or not to use the value of *density*. The default value is false.

density A Double specifying the value of density to apply to this section. The default value is 0.0.

thermalStresses An odb_SequenceDouble specifying the generalized stress values caused by a unit temperature rise. Six entries must be given if the value of *applyThermalStress* is set to True. The default value is ("").

scalingData An odb_SequenceSequenceDouble specifying the scaling factors for given temperatures and/or field data. Each row should contain (Y, alpha, T, F1,...,Fn). The default value is an empty sequence.

Return value

A GeneralStiffnessSection object.

Exceptions

None.

Members

The GeneralStiffnessSection object has members with the same names and descriptions as the arguments to the *GeneralStiffnessSection* method.

In addition, the GeneralStiffnessSection object can have the following members:

Prototype

```
odb_RebarLayers rebarLayers() const;
odb_TransverseShearShell transverseShear() const;
```

rebarLayers A *RebarLayers* object specifying reinforcement properties.

transverseShear A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL GENERAL SECTION*

GeometryShellSection object

The GeometryShellSection object defines the properties of a geometry shell section. The GeometryShellSection object has no explicit constructor and no methods. The GeometryShellSection object is an abstract base type.

The GeometryShellSection object is derived from the *ShellSection* object.

Access

```
sectionApi.sections()[name]
```

Members

The GeometryShellSection object can have the following members:

Prototype

```
odb_String name() const;
odb_String thicknessType() const;
bool preIntegrate() const;
odb_String poissonDefinition() const;
double poisson() const;
odb_String integrationRule() const;
odb_String temperature() const;
odb_String idealization() const;
odb_Union nTemp() const;
odb_Union thicknessModulus() const;
bool useDensity() const;
double density() const;
odb_String thicknessField() const;
odb_RebarLayers rebarLayers() const;
odb_String nodalThicknessField() const;
odb_TransverseShearShell transverseShear() const;
```

name

An odb_String specifying the repository key.

thicknessType

An odb_String specifying the distribution used for defining the thickness of the elements. Possible values are "UNIFORM", "ANALYTICAL_FIELD", "DISCRETE_FIELD", "NODAL_ANALYTICAL_FIELD", and "NODAL_DISCRETE_FIELD". The default value is "UNIFORM".

preIntegrate

A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (true) or integrated during the analysis (false). The default value is false.

poissonDefinition

An odb_String specifying whether to use the default value for the Poisson's ratio. Possible values are:

- "DEFAULT", specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- "VALUE", specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is "DEFAULT".

poisson

A Double specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*="VALUE". The default value is 0.5.

integrationRule

An odb_String specifying the shell section integration rule. Possible values are "SIMPSON" and "GAUSS". The default value is "SIMPSON".

temperature

An odb_String specifying the mode used for temperature and field variable input across the section thickness. Possible values are "GRADIENT" and "POINTWISE". The default value is "GRADIENT".

idealization

An odb_String specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to true. Possible values are "NO_IDEALIZATION", "SMEAR_ALL_LAYERS", "MEMBRANE", and "BENDING". The default value is "NO_IDEALIZATION".

nTemp

The string "NONE" or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*="POINTWISE". The default value is "NONE".

thicknessModulus

The string "NONE" or a Double specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is "NONE".

useDensity

A Boolean specifying whether or not to use the value of *density*. The default value is false.

density

A Double specifying the value of density to apply to this section. The default value is 0.0.

thicknessField

An odb_String specifying the name of the AnalyticalField object or DiscreteField object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*="ANALYTICAL_FIELD" or *thicknessType*="DISCRETE_FIELD". The default value is an empty string.

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

nodalThicknessField

An odb_String specifying the name of the AnalyticalField object or DiscreteField object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*="NODAL_ANALYTICAL_FIELD" or *thicknessType*="NODAL_DISCRETE_FIELD". The default value is an empty string.

transverseShear

A *TransverseShearShell* object specifying the transverse shear stiffness properties.

HomogeneousShellSection object

The HomogeneousShellSection object defines the properties of a shell section.

The HomogeneousShellSection object is derived from the *GeometryShellSection* object.

Access

```
sectionApi.sections()[name]
```

HomogeneousShellSection(...)

This method creates a HomogeneousShellSection object.

Path

```
sectionApi.HomogeneousShellSection
```

Prototype

```
odb_HomogeneousShellSection&
         HomogeneousShellSection(const odb_String& name,
           double thickness,
           const odb_String& material,
           int numIntPts,
           odb_String thicknessType,
           bool preIntegrate,
           odb_String poissonDefinition,
           double poisson,
           odb_String integrationRule,
           odb_String temperature,
           odb_String simplification,
           odb_Union nTemp,
           odb_Union thicknessModulus,
           bool useDensity,
           double density,
           odb_String thicknessField,
           odb_String nodalThicknessField);
```

Required arguments

name
An odb_String specifying the repository key.

thickness
A Double specifying the thickness of the section. The *thickness* argument applies only when *thicknessType*="UNIFORM".

material
An odb_String specifying the name of the section material.

Optional arguments

numIntPts
An Int specifying the number of integration points to be used through the section. Possible values are *numIntPts*> 0. The default value is 5.

To use the default settings of the analysis products, set *numIntPts* to 5 if *integrationRule*="SIMPSON" or set *numIntPts* to 7 if *integrationRule*="GAUSS".

thicknessType

An odb_String specifying the distribution used for defining the thickness of the elements. Possible values are "UNIFORM", "ANALYTICAL_FIELD", "DISCRETE_FIELD", "NODAL_ANALYTICAL_FIELD", and "NODAL_DISCRETE_FIELD". The default value is "UNIFORM".

preIntegrate

A Boolean specifying whether the shell section properties are specified by the user prior to the analysis (true) or integrated during the analysis (false). The default value is false.

poissonDefinition

An odb_String specifying whether to use the default value for the Poisson's ratio. Possible values are:

- "DEFAULT", specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- "VALUE", specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is "DEFAULT".

poisson

A Double specifying the Poisson's ratio. Possible values are $-1.0 \leq poisson \leq 0.5$. This argument is valid only when *poissonDefinition*="VALUE". The default value is 0.5.

integrationRule

An odb_String specifying the shell section integration rule. Possible values are "SIMPSON" and "GAUSS". The default value is "SIMPSON".

temperature

An odb_String specifying the mode used for temperature and field variable input across the section thickness. Possible values are "GRADIENT" and "POINTWISE". The default value is "GRADIENT".

idealization

An odb_String specifying the mechanical idealization used for the section calculations. This member is only applicable when *preIntegrate* is set to true. Possible values are "NO_IDEALIZATION", "SMEAR_ALL_LAYERS", "MEMBRANE", and "BENDING". The default value is "NO_IDEALIZATION".

nTemp

The string "NONE" or an Int specifying the number of temperature points to be input. This argument is valid only when *temperature*="POINTWISE". The default value is "NONE".

thicknessModulus

The string "NONE" or a Double specifying the effective thickness modulus. This argument is relevant only for continuum shells and must be used in conjunction with the argument *poisson*. The default value is "NONE".

useDensity

A Boolean specifying whether or not to use the value of *density*. The default value is false.

density

A Double specifying the value of density to apply to this section. The default value is 0.0.

thicknessField

An odb_String specifying the name of the AnalyticalField object or the DiscreteField object used to define the thickness of the shell elements. The *thicknessField* argument applies only

when *thicknessType*="ANALYTICAL_FIELD" or *thicknessType*="DISCRETE_FIELD". The default value is an empty string.

nodalThicknessField

An odb_String specifying the name of the AnalyticalField object or the DiscreteField object used to define the thickness of the shell elements at each node. The *nodalThicknessField* argument applies only when *thicknessType*="NODAL_ANALYTICAL_FIELD" or *thicknessType*="NODAL_DISCRETE_FIELD". The default value is an empty string.

Return value

A HomogeneousShellSection object.

Exceptions

None.

Members

The HomogeneousShellSection object has members with the same names and descriptions as the arguments to the *HomogeneousShellSection* method.

In addition, the HomogeneousShellSection object can have the following members:

Prototype

```
odb_RebarLayers rebarLayers() const;
odb_TransverseShearShell transverseShear() const;
```

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

transverseShear

A *TransverseShearShell* object specifying the transverse shear stiffness properties.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

HomogeneousSolidSection object

The HomogeneousSolidSection object defines the properties of a solid section.

The HomogeneousSolidSection object is derived from the *SolidSection* object.

Access

```
sectionApi.sections()[name]
```

HomogeneousSolidSection(...)

This method creates a HomogeneousSolidSection object.

Path

```
sectionApi.HomogeneousSolidSection
```

Prototype

```
odb_HomogeneousSolidSection&
HomogeneousSolidSection(const odb_String& name,
                        const odb_String& material,
                        double thickness);
```

Required arguments

name An odb_String specifying the repository key.

material An odb_String specifying the name of the material.

Optional arguments

thickness A double specifying the thickness of the section. Possible values are `None` or greater than zero. The default value is 1.0.

Return value

A HomogeneousSolidSection object.

Exceptions

InvalidNameError and RangeError.

Members

The HomogeneousSolidSection object has members with the same names and descriptions as the arguments to the *HomogeneousSolidSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

LayerProperties object

The LayerProperties object defines the properties of a layer of reinforcement for membrane, shell, and surface sections.

Access

```
sectionApi.sections()[name].rebarLayers().layerTable(i)
```

LayerProperties(...)

This method creates a LayerProperties object.

Path

LayerProperties

Prototype

```
odb_LayerProperties&
LayerProperties(double barArea,
                odb_Union orientationAngle,
                const odb_String& layerName,
                const odb_String& material,
                double barSpacing,
                double layerPosition,
                double spacingAngle,
                double extensionRatio,
                double radius);
```

Required arguments

barArea
A Double specifying the area per bar.

orientationAngle
An odb_Union specifying the orientation of the rebar. A Float specifies the angular orientation; a String specifies an orientation name.

layerName
An odb_String specifying the name of the rebar layer.

material
An odb_String specifying the name of the rebar material.

Optional arguments

barSpacing
A Double specifying the spacing of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to "CONSTANT". The default value is 0.0.

layerPosition
A Double specifying the position of the rebar from the middle surface of the shell. *layerPosition* applies only for homogeneous shell sections and composite shell sections. The default value is 0.0.

spacingAngle A Double specifying the spacing angle of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to "ANGULAR". The default value is 0.0.

extensionRatio A Double specifying the extension ratio for the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to "LIFT_EQUATION". The default value is 0.0.

radius A Double specifying the radius of the rebar. This argument is only valid if the *rebarSpacing* argument on the parent *RebarLayers* object is set to "LIFT_EQUATION". The default value is 0.0.

Return value

A LayerProperties object.

Exceptions

None.

Members

The LayerProperties object has members with the same names and descriptions as the arguments to the *LayerProperties* method.

Corresponding analysis keywords

- **REBAR LAYER*

MembraneSection object

The MembraneSection object defines the properties of a membrane section.

The MembraneSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

MembraneSection(...)

This method creates a MembraneSection object.

Path

```
sectionApi.MembraneSection
```

Prototype

```
odb_MembraneSection&
MembraneSection(const odb_String& name,
                const odb_String& material,
                double thickness,
                const odb_String& thicknessType,
                const odb_String& poissonDefinition,
                double poisson,
                const odb_String& thicknessField);
```

Required arguments

name An odb_String specifying the repository key.

material An odb_String specifying the name of the material.

Optional arguments

thickness A Double specifying the thickness for the section. Possible values are *thickness*> 0.0. The default value is 1.0.

thicknessType An odb_String specifying the distribution used for defining the thickness of the elements. Possible values are "UNIFORM", "ANALYTICAL_FIELD", and "DISCRETE_FIELD". The default value is "UNIFORM".

poissonDefinition An odb_String specifying whether to use the default value for the Poisson's ratio. Possible values are:

- "DEFAULT", specifying that the default value for the Poisson's ratio is 0.5 in an Abaqus/Standard analysis and is obtained from the material definition in an Abaqus/Explicit analysis.
- "VALUE", specifying that the Poisson's ratio used in the analysis is the value provided in *poisson*.

The default value is "DEFAULT".

poisson

A Double specifying the section Poisson's ratio. Possible values are $-1.0 \le poisson \le 0.5$. This argument is valid only when *poissonDefinition*="VALUE". The default value is 0.5.

thicknessField

An odb_String specifying the name of the AnalyticalField object or the DiscreteField object used to define the thickness of the shell elements. The *thicknessField* argument applies only when *thicknessType*="ANALYTICAL_FIELD" or *thicknessType*="DISCRETE_FIELD". The default value is an empty string.

Return value

A MembraneSection object.

Exceptions

RangeError and InvalidNameError.

Members

The MembraneSection object has members with the same names and descriptions as the arguments to the *MembraneSection* method.

In addition, the MembraneSection object can have the following member:

Prototype

```
odb_RebarLayers rebarLayers() const;
```

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

Corresponding analysis keywords

- **MEMBRANE SECTION*

MPCSection object

The MPCSection object defines the properties of a multi-point constraint section.

The MPCSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

MPCSection(...)

This method creates a MPCSection object.

Path

```
sectionApi.MPCSection
```

Prototype

```
odb_MPCSection&
MPCSection(const odb_String& name,
           const odb_String& mpcType,
           const odb_String& userMode,
           int userType);
```

Required arguments

name
An odb_String specifying the repository key.

mpcType
An odb_String specifying the MPC type of the section. Possible values are "BEAM_MPC", "ELBOW_MPC", "PIN_MPC", "LINK_MPC", "TIE_MPC", and "USER_DEFINED".

Optional arguments

userMode
An odb_String specifying the mode of the MPC when it is user-defined. Possible values are "DOF_MODE" and "NODE_MODE". The default value is "DOF_MODE".

The *userMode* argument applies only when *mpcType*="USER_DEFINED".

userType
An Int specifying to differentiate between different constraint types in a user-defined MPCSection. The default value is 0.

The *userType* argument applies only when *mpcType*="USER_DEFINED".

Return value

A MPCSection object.

Exceptions

RangeError and InvalidNameError.

Members

The MPCSection object has members with the same names and descriptions as the arguments to the *MPCSection* method.

Corresponding analysis keywords

- **MPC*

PEGSection object

The PEGSection object defines the properties of a solid section.

The PEGSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

PEGSection(...)

This method creates a PEGSection object.

Path

```
sectionApi.PEGSection
```

Prototype

```
odb_PEGSection&
PEGSection(const odb_String& name,
           const odb_String& material,
           double thickness,
           double wedgeAngle1,
           double wedgeAngle2);
```

Required arguments

name An odb_String specifying the repository key.

material An odb_String specifying the name of the material.

Optional arguments

thickness A Double specifying the thickness of the section. Possible values are *thickness*> 0.0. The default value is 1.0.

wedgeAngle1 A Double specifying the value of the x component of the angle between the bounding planes, $\Delta\phi_x$. The default value is 0.0.

wedgeAngle2 A Double specifying the value of the y component of the angle between the bounding planes, $\Delta\phi_y$. The default value is 0.0.

Return value

A PEGSection object.

Exceptions

InvalidNameError and RangeError.

Members

The PEGSection object has members with the same names and descriptions as the arguments to the *PEGSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

RebarLayers object

The RebarLayers object defines the rebar properties of a section.

Access

```
sectionApi.sections()[name].rebarLayers()
```

RebarLayers(...)

This method creates a RebarLayers object.

Path

```
sectionApi.sections()[name].RebarLayers
```

Prototype

```
odb_RebarLayers&
RebarLayers(const odb_String& rebarSpacing,
            const odb_SequenceLayerProperties& layerTable);
```

Required arguments

rebarSpacing
An odb_String specifying the type of rebar geometry. Possible values are "CONSTANT", "ANGULAR", and "LIFT_EQUATION".

layerTable
A sequence of *LayerProperties* objects specifying the layers of reinforcement.

Optional arguments

None.

Return value

A RebarLayers object.

Exceptions

None.

Members

The RebarLayers object has members with the same names and descriptions as the arguments to the *RebarLayers* method.

Corresponding analysis keywords

- **REBAR LAYER*

SectionLayer object

The SectionLayer object defines the material layer in a composite shell.

Access

```
sectionApi.sections()[name].layup(i)
```

SectionLayer(...)

This method creates a SectionLayer object.

Path

```
SectionLayer
```

Prototype

```
odb_SectionLayer&
SectionLayer(double thickness,
             const odb_String& material,
             odb_Union orientAngle,
             int numIntPts,
             const odb_String& axis,
             float angle,
             const odb_String& additionalRotationType,
             const odb_String& plyName,
             const odb_DatumCsys& orientation,
             const odb_String& additionalRotationField);
```

Required arguments

thickness

A Double specifying the thickness of the section layer.

material

An odb_String specifying the name of the section layer material.

Optional arguments

orientAngle

An odb_Union specifying the relative orientation of the section layer. A Float specifies the angular orientation; a String specifies a user-subroutine orientation name. If a String is specified, a user-subroutine orientation is used, otherwise the Float value is used as an angular orientation. The default value is 0.0.

numIntPts

An Int specifying the number of integration points to be used through the section. This argument is valid only if the *preIntegrate* argument on the parent *CompositeShellSection* object is set to true. The default value is 3.

axis

An odb_String specifying the axis of a cylindrical or spherical datum coordinate system about which an additional rotation is applied. For shells this axis is also the shell normal. This only applies if a valid reference is provided for the orientation. Possible values are "AXIS_1", "AXIS_2", and "AXIS_3". The default value is "AXIS_3".

angle A Float specifying the angle of the additional rotation. This only applies if a valid reference is provided for the orientation. The default value is 0.0.

additionalRotationType An odb_String specifying the method used to describe the additional rotation when a valid orientation is specified. Possible values are "ROTATION_NONE", "ROTATION_ANGLE", and "ROTATION_FIELD". The default value is "ROTATION_NONE".

plyName An odb_String specifying the ply identifier for this section layer. The default value is "".

orientation An *OdbDatumCsys object* specifying a coordinate system reference for the relative orientation of this layer. If this reference is valid it is used as the relative orientation of the layer, otherwise the *orientAngle* is used as described.

additionalRotationField An odb_String specifying the name of the field specifying the additional rotation. The default value is "".

Return value

A SectionLayer object.

Exceptions

None.

Members

The SectionLayer object has members with the same names and descriptions as the arguments to the *SectionLayer* method.

Corresponding analysis keywords

- **SHELL SECTION*
- **SHELL GENERAL SECTION*

ShellSection object

The ShellSection object defines the properties of a shell section. The ShellSection object is derived from the *Section* object. The ShellSection object has no explicit constructor and no methods or members.

The ShellSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

Members

The ShellSection object can have the following members:

Prototype

```
odb_String name() const;
odb_TransverseShearShell transverseShear() const;
```

name	An odb_String specifying the repository key.
transverseShear	A *TransverseShearShell* object specifying the transverse shear stiffness properties.

SolidSection object

The SolidSection object defines the properties of a solid section. The SolidSection object has no explicit constructor, no members, and no methods. The SolidSection object is an abstract base type.

The SolidSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

Members

The SolidSection object has the following member:

Prototype

```
odb_String name() const;
```

name An odb_String specifying the repository key.

SurfaceSection object

The SurfaceSection object defines the properties of a surface section.

The SurfaceSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

SurfaceSection(...)

This method creates a SurfaceSection object.

Path

```
sectionApi.SurfaceSection
```

Prototype

```
odb_SurfaceSection&
SurfaceSection(const odb_String& name,
               bool useDensity,
               double density);
```

Required arguments

name An odb_String specifying the repository key.

Optional arguments

useDensity A Boolean specifying whether or not to use the value of *density*. The default value is false.

density A Double specifying the value of density to apply to this section. The default value is 0.0.

Return value

A SurfaceSection object.

Exceptions

RangeError and InvalidNameError.

Members

The SurfaceSection object has members with the same names and descriptions as the arguments to the *SurfaceSection* method.

In addition, the SurfaceSection object can have the following member:

Prototype

```
odb_RebarLayers rebarLayers() const;
```

rebarLayers

A *RebarLayers* object specifying reinforcement properties.

Corresponding analysis keywords

- **SURFACE SECTION*

TransverseShearBeam object

The TransverseShearBeam object defines the transverse shear stiffness properties of a beam section.

Access

```
sectionApi.sections()[name].beamTransverseShear()
```

TransverseShearBeam(...)

This method creates a TransverseShearBeam object.

Path

```
sectionApi.sections()[name].TransverseShearBeam
```

Prototype

```
odb_TransverseShearBeam&
TransverseShearBeam(const odb_String& scfDefinition,
                    odb_Union k23,
                    odb_Union k13,
                    odb_Union slendernessCompensation);
```

Required arguments

scfDefinition
An odb_String specifying how slenderness compensation factor of the section is given. Possible values are "ANALYSIS_DEFAULT", "COMPUTED", and "VALUE".

Optional arguments

k23
The string "NONE" or a Double specifying the k23 shear stiffness of the section. The default value is "NONE".

k13
The string "NONE" or a Double specifying the k13 shear stiffness of the section. The default value is "NONE".

slendernessCompensation
The string "COMPUTED" or a Double specifying the slenderness compensation factor of the section. The default value is 0.25.

Return value

A TransverseShearBeam object.

Exceptions

None.

Members

The TransverseShearBeam object has members with the same names and descriptions as the arguments to the *TransverseShearBeam* method.

Corresponding analysis keywords

- **TRANSVERSE SHEAR STIFFNESS*

TransverseShearShell object

The TransverseShearShell object defines the transverse shear stiffness properties of a shell section.

Access

```
sectionApi.sections()[name].transverseShear()
```

TransverseShearShell(...)

This method creates a TransverseShearShell object.

Path

```
sectionApi.sections()[name].TransverseShearShell
```

Prototype

```
odb_TransverseShearShell&
TransverseShearShell(double k11,
                     double k22,
                     double k12);
```

Required arguments

k11
A Double specifying the shear stiffness of the section in the first direction.

k22
A Double specifying the shear stiffness of the section in the second direction.

k12
A Double specifying the coupling term in the shear stiffness of the section.

Optional arguments

None.

Return value

A TransverseShearShell object.

Exceptions

None.

Members

The TransverseShearShell object has members with the same names and descriptions as the arguments to the *TransverseShearShell* method.

Corresponding analysis keywords

- **TRANSVERSE SHEAR STIFFNESS*

TrussSection object

The TrussSection object defines the properties of a truss section.

The TrussSection object is derived from the *Section* object.

Access

```
sectionApi.sections()[name]
```

TrussSection(...)

This method creates a TrussSection object.

Path

```
sectionApi.TrussSection
```

Prototype

```
odb_TrussSection&
TrussSection(const odb_String& name,
             const odb_String& material,
             double area);
```

Required arguments

name An odb_String specifying the repository key.

material An odb_String specifying the name of the material.

Optional arguments

area A Double specifying the cross-sectional area for the section. Possible values are *area*> 0. The default value is 1.0.

Return value

A TrussSection object.

Exceptions

RangeError and InvalidNameError.

Members

The TrussSection object has members with the same names and descriptions as the arguments to the *TrussSection* method.

Corresponding analysis keywords

- **SOLID SECTION*

Infrastructure commands

The generic objects described in this chapter are used in the construction of specific Abaqus objects.

In this section:

- *BaseException object*
- *Repository object*
- *RepositoryIterator object*
- *Sequence object*
- *String object*
- *Union object*

BaseException object

The odb_BaseException object catches all exceptions thrown in the output database

Access

```
catch(odb_BaseException& exc)
```

`UserReport()`

This method returns a description of the error condition that generated the exception.

Prototype

```
odb_String UserReport() const;
```

Arguments

None.

Return value

`None.`

Exceptions

None.

Repository object

Repositories are provided to store objects retrieved by name. Both the repositories and the content of the repositories are created by the API; the user can only retrieve objects from repositories. Iterators are provided to navigate the repositories.

operator[](...)

This method accesses an item in an odb_Repository object. This method should be used with caution because it does not perform any checking of the index range.

Prototype

```
type [odb_string name] const;
```

Required arguments

name A char* or an odb_String specifying the object to be retrieved.

Optional arguments

None.

Return value

The object of type *type* saved in the repository under the given name.

Exceptions

None.

get/constGet(...)

This method accesses an item in an odb_Repository object.

Prototype

```
type get(const char* name);
const type& constGet(const char* name) const;
```

Required arguments

name A char* or an odb_String specifying the object to be retrieved.

Optional arguments

None.

Return value

The object of type *type* saved in the repository under the given name.

Exceptions

None.

isMember(...)

This method checks whether an object with a given name is stored in the repository.

Prototype

```
bool isMember(const odb_String& name) const;
bool isMember(const char* name) const;
```

Required arguments

name A char* or an odb_String specifying the name under which the object is stored.

Optional arguments

None.

Return value

A Boolean specifying whether the object is stored in the repository.

Exceptions

None.

size(...)

This method returns the number of items in an `odb_Repository` object.

Prototype

```
int size() const;
```

Arguments

None.

Return value

An Int indicating the number of items in the repository.

Exceptions

None.

RepositoryIterator object

Iterators are provided to navigate the repositories. Examples of using an iterator can be found in the example problem section.

odb_TypeRepositoryIT(...)

This method constructs an iterator for a repository.

Path

odb_*Type*RepositoryIT

Prototype

```
odb_TypeRepositoryIT(const odb_TypeRepository& repository);
```

Required arguments

repository	The repository over which to iterate.

Optional arguments

None.

Return value

An Iterator object.

Exceptions

None.

first(...)

This method sets the iterator to the first item in the repository.

Prototype

```
first();
```

Arguments

None.

Return value

None.

Exceptions

None.

next(...)

This method sets the iterator to the next item in the repository.

Prototype

```
next();
```

Arguments

None.

Return value

None.

Exceptions

None.

isDone(...)

This method checks whether the iterator is at the end of the repository.

Prototype

```
bool isDone();
```

Arguments

None.

Return value

A Boolean specifying whether the end of the repository has been reached.

Exceptions

None.

currentKey(...)

This method returns the name of the item to which the iterator is currently pointing.

Prototype

```
odb_String currentKey();
```

Arguments

None.

Return value

An odb_String specifying the name of the item that will be returned by the `currentValue` method.

Exceptions

None.

currentValue(...)

This method returns the item in the repository to which the iterator is currently pointing.

Prototype

```
const type& currentValue();
```

Arguments

None.

Return value

An item of the class *type* to which the iterator is currently pointing.

Exceptions

None.

Sequence object

An odb_Sequence object is a repository used to hold an ordered list of objects of a specific type. Data are added using the `append` method. The odb_Sequence object will automatically allocate memory to contain the appended object; the user may, however, control the efficiency of the memory allocation process through the constructor.

odb_Sequence*Type*(...)

This method creates an empty odb_Sequence object of objects of type *Type*.

Path

odb_Sequence*Type*

Prototype

```
odb_SequenceType();
```

Required arguments

None.

Optional arguments

The optional arguments are mutually exclusive.

intialSize An Int specifying the amount of memory to be allocated when the object is created. The default value is 0.

instance Applicable only for odb_SequenceElement. An *OdbInstance* object specifying the instance to which the elements belong.

assembly Applicable only for odb_SequenceElement. An *OdbAssembly* object specifying the assembly to which the elements belong.

Return value

An empty odb_Sequence object.

Exceptions

None.

append(...)

This method adds items to an odb_Sequence object.

Prototype

```
void append(Type item);
```

Required arguments

item

An item of type *Type* to be appended to the odb_Sequence object, where *Type* must be one of the following:

- Float
- Int
- String
- SectionLayer
- LayerProperties
- odbMeshElement
- FieldOutput

For example, an odb_SequenceInt object accepts an argument of type Int.

Optional arguments

None.

Return value

`None.`

Exceptions

None.

constGet/get(...)

This method accesses an item in an odb_Sequence object.

Prototype

```
Type get(int index);
const Type& constGet(int index) const;
```

Required arguments

index

An Int specifying the index of the item to be accessed from the odb_Sequence object.

Optional arguments

None.

Return value

An item of the *Type* stored in the odb_Sequence object.

Exceptions

None.

isMember(...)

This method checks whether an object is stored in the sequence.

Prototype

```
bool isMember(const Type& item) const;
```

Required arguments

item An object of type *Type*.

Optional arguments

None.

Return value

A Boolean specifying whether the object is stored in the sequence.

Exceptions

None.

operator[](...)

This method accesses an item in an odb_Sequence object.

Prototype

```
const Type& [int index] const;
```

Required arguments

index An Int specifying the index of the item to be accessed from the odb_Sequence object.

Optional arguments

None.

Return value

An item of the *Type* stored in the odb_Sequence object.

Exceptions

None.

size(...)

This method returns the number of items in an odb_Sequence object.

Prototype

```
int size() const;
```

Arguments

None.

Return value

An Int indicating the number of items in the sequence.

Exceptions

None.

String object

An odb_String object is used to specify a character string. The odb_String object provides a convenient means of storing and passing strings, along with a simple interface to append and modify the data stored in the string.

odb_String(...)

This constructor creates an odb_String object from a char*. Any method that accepts an odb_String argument will also accept a char*.

Prototype

```
odb_String(const char* string)
odb_String()
```

Required arguments

A char* to initialize the odb_String.

Optional arguments

None.

Return value

An odb_String object.

Exceptions

None.

append(...)

This method adds characters to an odb_String object.

Prototype

```
void append(const odb_String& string);
```

Required arguments

string An odb_String to be appended to the contents of odb_String.

Optional arguments

None.

Return value

None.

Exceptions

None.

cStr(...)

This method returns a char* to the string held by an odb_String object.

Prototype

```
char* cStr();
```

Arguments

None.

Return value

A char* to the string held by the odb_String object.

Exceptions

None.

length(...)

This method returns the number of characters of an odb_String object.

Prototype

```
int length();
```

Arguments

None.

Return value

An Int denoting the number of characters.

Exceptions

None.

Union object

An odb_Union object is used to determine the type of variable used when an object allows multiple variable types. The odb_Union object can have a value that is either a SymbolicConstant, Int, Float, Double, Boolean, or String.

type(...)

This method returns the current type of the odb_Union object.

Prototype

```
odb_UnionType type();
```

Arguments

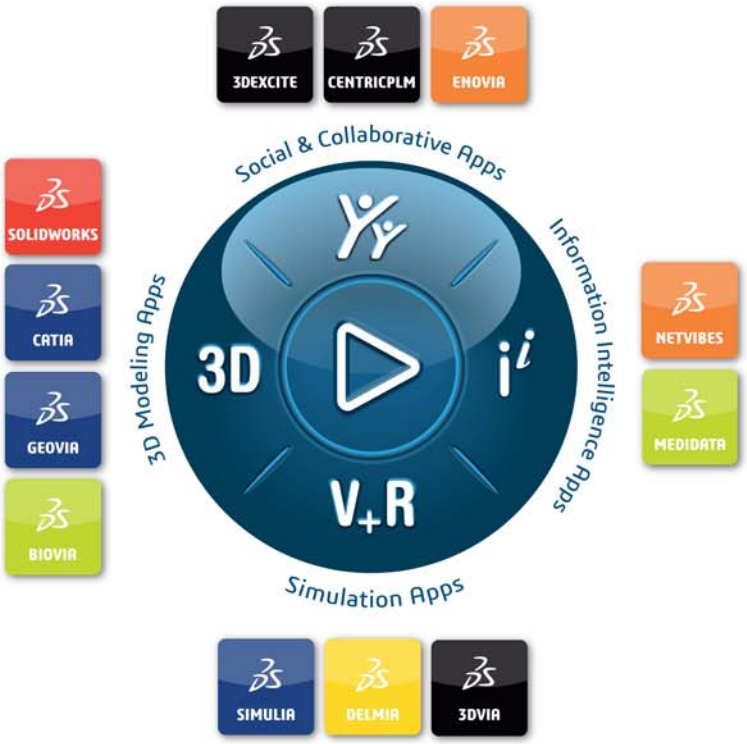
None.

Return value

The type of the odb_Union object. Possible values are odb_UNION_INT, odb_UNION_FLOAT, odb_UNION_DOUBLE, odb_UNION_BOOL, and odb_UNION_STRING.

Exceptions

odbException



Our **3D**EXPERIENCE® platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes is a catalyst for human progress. We provide business and people with collaborative virtual environments to imagine sustainable innovations. By creating virtual twin experiences of the real world with our **3DEXPERIENCE** platform and applications, our customers can redefine the creation, production and life-cycle-management processes of their offer and thus have a meaningful impact to make the world more sustainable. The beauty of the Experience Economy is that it is a human-centered economy for the benefit of all – consumers, patients and citizens.

Dassault Systèmes brings value to more than 300,000 customers of all sizes, in all industries, in more than 150 countries. For more information, visit **www.3ds.com**.

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes
17F, Foxconn Building,
No. 1366, Lujiazui Ring Road
Pilot Free Trade Zone, Shanghai 200120
China

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

DASSAULT SYSTEMES

©2025 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries.