

## GMAT TAT-C Project

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	AbsoluteDate Class Reference . . . . .	9
5.1.1	Detailed Description . . . . .	10
5.1.2	Constructor & Destructor Documentation . . . . .	10
5.1.2.1	AbsoluteDate() [1/2] . . . . .	10
5.1.2.2	AbsoluteDate() [2/2] . . . . .	10
5.1.2.3	~AbsoluteDate() . . . . .	10
5.1.3	Member Function Documentation . . . . .	11
5.1.3.1	Advance() . . . . .	11
5.1.3.2	Clone() . . . . .	11
5.1.3.3	GetGregorianDate() . . . . .	11
5.1.3.4	GetJulianDate() . . . . .	12
5.1.3.5	GregorianToJulianDate() . . . . .	12
5.1.3.6	operator=() . . . . .	12

5.1.3.7	SetGregorianDate()	13
5.1.3.8	SetJulianDate()	13
5.1.4	Member Data Documentation	13
5.1.4.1	currentDate	14
5.1.4.2	DAYS_PER_MONTH	14
5.1.4.3	JD_1900	14
5.2	Attitude Class Reference	14
5.2.1	Detailed Description	15
5.2.2	Constructor & Destructor Documentation	15
5.2.2.1	Attitude() [1/2]	15
5.2.2.2	Attitude() [2/2]	15
5.2.2.3	~Attitude()	15
5.2.3	Member Function Documentation	16
5.2.3.1	Clone()	16
5.2.3.2	InertialToReference()	16
5.2.3.3	operator=()	16
5.3	ConicalSensor Class Reference	17
5.3.1	Detailed Description	18
5.3.2	Constructor & Destructor Documentation	18
5.3.2.1	ConicalSensor() [1/2]	18
5.3.2.2	ConicalSensor() [2/2]	18
5.3.2.3	~ConicalSensor()	20
5.3.3	Member Function Documentation	20
5.3.3.1	CheckTargetVisibility()	20
5.3.3.2	GetFieldOfView()	20
5.3.3.3	operator=()	21
5.3.3.4	SetFieldOfView()	21
5.3.4	Member Data Documentation	21
5.3.4.1	fieldOfView	21
5.4	CoverageChecker Class Reference	22

5.4.1	Detailed Description	23
5.4.2	Constructor & Destructor Documentation	23
5.4.2.1	CoverageChecker() [1/2]	23
5.4.2.2	CoverageChecker() [2/2]	24
5.4.2.3	~CoverageChecker()	24
5.4.3	Member Function Documentation	24
5.4.3.1	AccumulateCoverageData() [1/2]	24
5.4.3.2	AccumulateCoverageData() [2/2]	25
5.4.3.3	CheckGridFeasibility() [1/2]	25
5.4.3.4	CheckGridFeasibility() [2/2]	25
5.4.3.5	CheckPointCoverage()	26
5.4.3.6	CreateNewPOIReport()	26
5.4.3.7	GetEarthFixedSatState()	26
5.4.3.8	operator=()	27
5.4.3.9	ProcessCoverageData()	27
5.4.3.10	SetComputePOIGeometryData()	27
5.4.4	Member Data Documentation	28
5.4.4.1	bfState	28
5.4.4.2	BODY_RADIUS	28
5.4.4.3	bodyUnit	28
5.4.4.4	centralBody	28
5.4.4.5	computePOIGeometryData	28
5.4.4.6	coverageEnd	29
5.4.4.7	coverageStart	29
5.4.4.8	dateData	29
5.4.4.9	discreteEventData	29
5.4.4.10	feasibilityTest	29
5.4.4.11	numEventsPerPoint	29
5.4.4.12	pointArray	30
5.4.4.13	pointGroup	30

5.4.4.14	ptPos	30
5.4.4.15	rangeVec	30
5.4.4.16	sc	30
5.4.4.17	timeldx	30
5.4.4.18	timeSeriesData	31
5.5	CustomSensor Class Reference	31
5.5.1	Detailed Description	32
5.5.2	Constructor & Destructor Documentation	32
5.5.2.1	CustomSensor() [1/2]	33
5.5.2.2	CustomSensor() [2/2]	33
5.5.2.3	~CustomSensor()	33
5.5.3	Member Function Documentation	33
5.5.3.1	CheckRegionVisibility()	33
5.5.3.2	CheckTargetMaxExcursionCoordinates()	34
5.5.3.3	CheckTargetVisibility()	34
5.5.3.4	ComputeExternalPoints()	34
5.5.3.5	Max()	34
5.5.3.6	Min()	34
5.5.3.7	operator=()	36
5.5.3.8	PointsToSegments()	36
5.5.3.9	RegionIsFullyContained()	36
5.5.3.10	Sort() [1/2]	36
5.5.3.11	Sort() [2/2]	37
5.5.4	Member Data Documentation	37
5.5.4.1	clockAngleVec	37
5.5.4.2	coneAngleVec	37
5.5.4.3	externalPointArray	37
5.5.4.4	maxXExcursion	37
5.5.4.5	maxYExcursion	37
5.5.4.6	minXExcursion	38

5.5.4.7	minYExcursion	38
5.5.4.8	numFOVPoints	38
5.5.4.9	numTestPoints	38
5.5.4.10	segmentArray	38
5.5.4.11	xProjectionCoordArray	38
5.5.4.12	yProjectionCoordArray	38
5.6	Earth Class Reference	39
5.6.1	Detailed Description	40
5.6.2	Constructor & Destructor Documentation	40
5.6.2.1	Earth() [1/2]	40
5.6.2.2	Earth() [2/2]	40
5.6.2.3	~Earth()	40
5.6.3	Member Function Documentation	40
5.6.3.1	ComputeGMT()	40
5.6.3.2	Convert()	41
5.6.3.3	FixedToTopo()	41
5.6.3.4	FixedToTopocentric()	42
5.6.3.5	GeocentricToGeodeticLat()	42
5.6.3.6	GetBodyFixedState()	43
5.6.3.7	GetEarthSunDistRaDec()	43
5.6.3.8	GetInertialToFixedRotation()	43
5.6.3.9	GetRadius()	44
5.6.3.10	GetSunPositionInBodyCoords()	44
5.6.3.11	InertialToBodyFixed()	44
5.6.3.12	operator=()	45
5.6.4	Member Data Documentation	45
5.6.4.1	flattening	45
5.6.4.2	J2	45
5.6.4.3	lastRotationTime	46
5.6.4.4	mu	46

5.6.4.5	radius	46
5.6.4.6	rotationResult	46
5.7	IntervalEventReport Class Reference	46
5.7.1	Detailed Description	47
5.7.2	Constructor & Destructor Documentation	47
5.7.2.1	IntervalEventReport() [1/2]	47
5.7.2.2	IntervalEventReport() [2/2]	48
5.7.2.3	~IntervalEventReport()	48
5.7.3	Member Function Documentation	48
5.7.3.1	AddPOIEvent()	48
5.7.3.2	GetEndDate()	49
5.7.3.3	GetPOIEvents()	49
5.7.3.4	GetPOIIndex()	49
5.7.3.5	GetStartDate()	49
5.7.3.6	operator=()	49
5.7.3.7	SetAllPOIEvents()	50
5.7.3.8	SetEndDate()	50
5.7.3.9	SetPOIIndex()	50
5.7.3.10	SetStartDate()	51
5.7.4	Member Data Documentation	51
5.7.4.1	discretePOIEvents	51
5.7.4.2	endDate	51
5.7.4.3	poiIndex	51
5.7.4.4	startDate	52
5.8	KeyValueStatistics Class Reference	52
5.8.1	Detailed Description	52
5.8.2	Constructor & Destructor Documentation	52
5.8.2.1	KeyValueStatistics() [1/2]	53
5.8.2.2	KeyValueStatistics() [2/2]	53
5.8.2.3	~KeyValueStatistics()	53



5.8.3	Member Function Documentation	53
5.8.3.1	GetAvgValue()	53
5.8.3.2	GetMaxValue()	54
5.8.3.3	GetMinValue()	54
5.8.3.4	operator=()	54
5.8.4	Member Data Documentation	54
5.8.4.1	avgValue	54
5.8.4.2	maxValue	55
5.8.4.3	minValue	55
5.9	LinearAlgebra Class Reference	55
5.9.1	Detailed Description	55
5.9.2	Member Function Documentation	55
5.9.2.1	LineSegmentIntersect()	55
5.10	NadirPointingAttitude Class Reference	57
5.10.1	Detailed Description	58
5.10.2	Constructor & Destructor Documentation	58
5.10.2.1	NadirPointingAttitude() [1/2]	58
5.10.2.2	NadirPointingAttitude() [2/2]	58
5.10.2.3	~NadirPointingAttitude()	59
5.10.3	Member Function Documentation	59
5.10.3.1	InertialToReference()	59
5.10.3.2	operator=()	59
5.10.4	Member Data Documentation	60
5.10.4.1	centralBodyFixedPos	60
5.10.4.2	centralBodyFixedVel	60
5.10.4.3	R_fixed_to_nadir	60
5.10.4.4	R_fixed_to_nadir_transposed	60
5.10.4.5	xHat	60
5.10.4.6	yHat	60
5.10.4.7	zHat	61

5.11 OrbitState Class Reference . . . . .	61
5.11.1 Detailed Description . . . . .	62
5.11.2 Constructor & Destructor Documentation . . . . .	62
5.11.2.1 OrbitState() [1/2] . . . . .	62
5.11.2.2 OrbitState() [2/2] . . . . .	62
5.11.2.3 ~OrbitState() . . . . .	62
5.11.3 Member Function Documentation . . . . .	62
5.11.3.1 Clone() . . . . .	62
5.11.3.2 ConvertCartesianToKeplerian() . . . . .	63
5.11.3.3 ConvertKeplerianToCartesian() . . . . .	63
5.11.3.4 GetCartesianState() . . . . .	63
5.11.3.5 GetKeplerianState() . . . . .	64
5.11.3.6 operator=() . . . . .	64
5.11.3.7 SetCartesianState() . . . . .	64
5.11.3.8 SetGravityParameter() . . . . .	65
5.11.3.9 SetKeplerianState() . . . . .	65
5.11.3.10 SetKeplerianVectorState() . . . . .	65
5.11.4 Member Data Documentation . . . . .	66
5.11.4.1 currentState . . . . .	66
5.11.4.2 mu . . . . .	66
5.12 PointGroup Class Reference . . . . .	66
5.12.1 Detailed Description . . . . .	67
5.12.2 Constructor & Destructor Documentation . . . . .	67
5.12.2.1 PointGroup() [1/2] . . . . .	67
5.12.2.2 PointGroup() [2/2] . . . . .	67
5.12.2.3 ~PointGroup() . . . . .	68
5.12.3 Member Function Documentation . . . . .	68
5.12.3.1 AccumulatePoints() . . . . .	68
5.12.3.2 AddHelicalPointsByAngle() . . . . .	68
5.12.3.3 AddHelicalPointsByNumPoints() . . . . .	69

5.12.3.4	AddUserDefinedPoints()	69
5.12.3.5	CheckHasPoints()	69
5.12.3.6	ComputeHelicalPoints()	69
5.12.3.7	ComputeTestPoints()	70
5.12.3.8	GetLatAndLon()	70
5.12.3.9	GetLatLonVectors()	70
5.12.3.10	GetNumPoints()	71
5.12.3.11	GetPointPositionVector()	71
5.12.3.12	operator=()	71
5.12.3.13	SetLatLonBounds()	72
5.12.4	Member Data Documentation	72
5.12.4.1	coords	72
5.12.4.2	lat	72
5.12.4.3	latLower	72
5.12.4.4	latUpper	73
5.12.4.5	lon	73
5.12.4.6	lonLower	73
5.12.4.7	lonUpper	73
5.12.4.8	numPoints	73
5.12.4.9	numRequestedPoints	73
5.13	Propagator Class Reference	74
5.13.1	Detailed Description	76
5.13.2	Constructor & Destructor Documentation	76
5.13.2.1	Propagator() [1/2]	76
5.13.2.2	Propagator() [2/2]	76
5.13.2.3	~Propagator()	77
5.13.3	Member Function Documentation	77
5.13.3.1	ComputeArgumentOfPeriapsisRate()	77
5.13.3.2	ComputeDragEffects()	77
5.13.3.3	ComputeMeanMotionRate()	78

5.13.3.4	ComputeOrbitRates()	78
5.13.3.5	ComputePeriapsisAltitude()	78
5.13.3.6	ComputeRightAscensionNodeRate()	78
5.13.3.7	GetApplyDrag()	79
5.13.3.8	GetPropStartEnd()	79
5.13.3.9	MeanMotion()	79
5.13.3.10	operator=()	79
5.13.3.11	Propagate()	80
5.13.3.12	PropagateOrbitalElements()	80
5.13.3.13	SemiParameter()	80
5.13.3.14	SetApplyDrag()	81
5.13.3.15	SetOrbitState()	81
5.13.3.16	SetPhysicalConstants()	81
5.13.4	Member Data Documentation	82
5.13.4.1	AOP	82
5.13.4.2	applyDrag	82
5.13.4.3	argPeriapsisRate	82
5.13.4.4	densityModel	82
5.13.4.5	ECC	82
5.13.4.6	eqRadius	83
5.13.4.7	INC	83
5.13.4.8	J2	83
5.13.4.9	lastDragUpdateEpoch	83
5.13.4.10	MA	83
5.13.4.11	meanMotion	83
5.13.4.12	meanMotionRate	84
5.13.4.13	mu	84
5.13.4.14	MU_FOR_EARTH	84
5.13.4.15	orbitPeriod	84
5.13.4.16	propEnd	84

5.13.4.17 propStart . . . . .	84
5.13.4.18 RAAN . . . . .	85
5.13.4.19 refJd . . . . .	85
5.13.4.20 rightAscensionNodeRate . . . . .	85
5.13.4.21 sc . . . . .	85
5.13.4.22 semiLatusRectum . . . . .	85
5.13.4.23 SMA . . . . .	85
5.13.4.24 TA . . . . .	86
5.14 RectangularSensor Class Reference . . . . .	86
5.14.1 Detailed Description . . . . .	87
5.14.2 Constructor & Destructor Documentation . . . . .	87
5.14.2.1 RectangularSensor() [1/2] . . . . .	87
5.14.2.2 RectangularSensor() [2/2] . . . . .	88
5.14.2.3 ~RectangularSensor() . . . . .	88
5.14.3 Member Function Documentation . . . . .	88
5.14.3.1 CheckTargetVisibility() . . . . .	88
5.14.3.2 GetAngleHeight() . . . . .	89
5.14.3.3 GetAngleWidth() . . . . .	89
5.14.3.4 operator=() . . . . .	89
5.14.3.5 SetAngleHeight() . . . . .	89
5.14.3.6 SetAngleWidth() . . . . .	90
5.14.4 Member Data Documentation . . . . .	90
5.14.4.1 angleHeight . . . . .	90
5.14.4.2 angleWidth . . . . .	90
5.15 Sensor Class Reference . . . . .	91
5.15.1 Detailed Description . . . . .	92
5.15.2 Constructor & Destructor Documentation . . . . .	92
5.15.2.1 Sensor() [1/2] . . . . .	92
5.15.2.2 Sensor() [2/2] . . . . .	93
5.15.2.3 ~Sensor() . . . . .	93

5.15.3	Member Function Documentation	93
5.15.3.1	CheckTargetMaxExcursionAngle()	93
5.15.3.2	CheckTargetVisibility()	93
5.15.3.3	ComputeBodyToSensorMatrix()	95
5.15.3.4	ConeClockArraysToStereographic()	95
5.15.3.5	ConeClocktoRADEC()	95
5.15.3.6	ConeClockToStereographic()	96
5.15.3.7	GetBodyToSensorMatrix()	96
5.15.3.8	operator=()	96
5.15.3.9	RADECtoUnitVec()	96
5.15.3.10	SetSensorBodyOffsetAngles()	97
5.15.3.11	UnitVecToStereographic()	97
5.15.4	Member Data Documentation	97
5.15.4.1	eulerSeq1	97
5.15.4.2	eulerSeq2	97
5.15.4.3	eulerSeq3	98
5.15.4.4	maxExcursionAngle	98
5.15.4.5	offsetAngle1	98
5.15.4.6	offsetAngle2	98
5.15.4.7	offsetAngle3	98
5.15.4.8	R_SB	98
5.16	SensorElement Struct Reference	99
5.16.1	Detailed Description	99
5.16.2	Member Data Documentation	99
5.16.2.1	index	99
5.16.2.2	value	99
5.17	Spacecraft Class Reference	99
5.17.1	Detailed Description	101
5.17.2	Constructor & Destructor Documentation	101
5.17.2.1	Spacecraft() [1/2]	102

5.17.2.2	Spacecraft() [2/2]	102
5.17.2.3	~Spacecraft()	102
5.17.3	Member Function Documentation	103
5.17.3.1	AddSensor()	103
5.17.3.2	CanInterpolate()	103
5.17.3.3	CheckTargetVisibility() [1/2]	103
5.17.3.4	CheckTargetVisibility() [2/2]	105
5.17.3.5	ComputeNadirToBodyMatrix()	105
5.17.3.6	GetBodyFixedToInertial()	106
5.17.3.7	GetCartesianState()	106
5.17.3.8	GetCartesianStateAtEpoch()	106
5.17.3.9	GetDragArea()	107
5.17.3.10	GetDragCoefficient()	107
5.17.3.11	GetJulianDate()	107
5.17.3.12	GetOrbitEpoch()	107
5.17.3.13	GetOrbitState()	108
5.17.3.14	GetTotalMass()	108
5.17.3.15	HasSensors()	108
5.17.3.16	InertialToConeClock()	108
5.17.3.17	Interpolate()	109
5.17.3.18	operator=()	109
5.17.3.19	SetAttitude()	109
5.17.3.20	SetBodyNadirOffsetAngles()	110
5.17.3.21	SetDragArea()	110
5.17.3.22	SetDragCoefficient()	110
5.17.3.23	SetOrbitState()	111
5.17.3.24	SetTotalMass()	111
5.17.3.25	TimeToInterpolate()	112
5.17.4	Member Data Documentation	112
5.17.4.1	attitude	112

5.17.4.2	<a href="#">dragArea</a>	112
5.17.4.3	<a href="#">dragCoefficient</a>	112
5.17.4.4	<a href="#">eulerSeq1</a>	113
5.17.4.5	<a href="#">eulerSeq2</a>	113
5.17.4.6	<a href="#">eulerSeq3</a>	113
5.17.4.7	<a href="#">interpolator</a>	113
5.17.4.8	<a href="#">numSensors</a>	113
5.17.4.9	<a href="#">offsetAngle1</a>	113
5.17.4.10	<a href="#">offsetAngle2</a>	113
5.17.4.11	<a href="#">offsetAngle3</a>	114
5.17.4.12	<a href="#">orbitEpoch</a>	114
5.17.4.13	<a href="#">orbitState</a>	114
5.17.4.14	<a href="#">R_BN</a>	114
5.17.4.15	<a href="#">sensorList</a>	114
5.17.4.16	<a href="#">totalMass</a>	114
5.18	<a href="#">TATCException Class Reference</a>	115
5.18.1	<a href="#">Detailed Description</a>	115
5.18.2	<a href="#">Constructor &amp; Destructor Documentation</a>	115
5.18.2.1	<a href="#">TATCException() [1/2]</a>	116
5.18.2.2	<a href="#">TATCException() [2/2]</a>	116
5.19	<a href="#">VisibilityReport Class Reference</a>	116
5.19.1	<a href="#">Detailed Description</a>	117
5.19.2	<a href="#">Constructor &amp; Destructor Documentation</a>	117
5.19.2.1	<a href="#">VisibilityReport() [1/2]</a>	118
5.19.2.2	<a href="#">VisibilityReport() [2/2]</a>	118
5.19.2.3	<a href="#">~VisibilityReport()</a>	118
5.19.3	<a href="#">Member Function Documentation</a>	118
5.19.3.1	<a href="#">GetEndDate()</a>	118
5.19.3.2	<a href="#">GetStartDate()</a>	119
5.19.3.3	<a href="#">operator=()</a>	119



5.19.3.4	SetEndDate()	119
5.19.3.5	SetStartDate()	119
5.19.4	Member Data Documentation	120
5.19.4.1	endDate	120
5.19.4.2	startDate	120
5.20	VisiblePOIReport Class Reference	120
5.20.1	Detailed Description	122
5.20.2	Constructor & Destructor Documentation	122
5.20.2.1	VisiblePOIReport() [1/2]	122
5.20.2.2	VisiblePOIReport() [2/2]	122
5.20.2.3	~VisiblePOIReport()	122
5.20.3	Member Function Documentation	122
5.20.3.1	GetObsAzimuth()	123
5.20.3.2	GetObsRange()	123
5.20.3.3	GetObsZenith()	123
5.20.3.4	GetPOIIndex()	123
5.20.3.5	GetSunAzimuth()	124
5.20.3.6	GetSunZenith()	124
5.20.3.7	operator=()	124
5.20.3.8	SetObsAzimuth()	124
5.20.3.9	SetObsRange()	125
5.20.3.10	SetObsZenith()	125
5.20.3.11	SetPOIIndex()	125
5.20.3.12	SetSunAzimuth()	126
5.20.3.13	SetSunZenith()	126
5.20.4	Member Data Documentation	126
5.20.4.1	obsAzimuth	126
5.20.4.2	obsRange	126
5.20.4.3	obsZenith	127
5.20.4.4	poiIndex	127
5.20.4.5	sunAzimuth	127
5.20.4.6	sunZenith	127

<b>6</b>	<b>File Documentation</b>	<b>129</b>
6.1	src/AbsoluteDate.cpp File Reference	129
6.2	src/AbsoluteDate.hpp File Reference	129
6.3	src/Attitude.cpp File Reference	130
6.4	src/Attitude.hpp File Reference	130
6.5	src/ConicalSensor.cpp File Reference	131
6.6	src/ConicalSensor.hpp File Reference	132
6.7	src/CoverageChecker.cpp File Reference	133
6.8	src/CoverageChecker.hpp File Reference	133
6.9	src/CustomSensor.cpp File Reference	134
6.9.1	Function Documentation	134
6.9.1.1	CompareSensorElements()	135
6.10	src/CustomSensor.hpp File Reference	135
6.10.1	Function Documentation	136
6.10.1.1	CompareSensorElements()	136
6.11	src/Earth.cpp File Reference	137
6.12	src/Earth.hpp File Reference	137
6.13	src/IntervalEventReport.cpp File Reference	138
6.14	src/IntervalEventReport.hpp File Reference	139
6.15	src/KeyValueStatistics.cpp File Reference	141
6.16	src/KeyValueStatistics.hpp File Reference	141
6.17	src/LinearAlgebra.cpp File Reference	142
6.18	src/LinearAlgebra.hpp File Reference	142
6.19	src/NadirPointingAttitude.cpp File Reference	143
6.20	src/NadirPointingAttitude.hpp File Reference	144
6.21	src/OrbitState.cpp File Reference	145
6.22	src/OrbitState.hpp File Reference	145
6.23	src/PointGroup.cpp File Reference	146
6.24	src/PointGroup.hpp File Reference	146
6.25	src/Propagator.cpp File Reference	147
6.26	src/Propagator.hpp File Reference	148
6.27	src/RectangularSensor.cpp File Reference	149
6.28	src/RectangularSensor.hpp File Reference	149
6.29	src/Sensor.cpp File Reference	150
6.30	src/Sensor.hpp File Reference	150
6.31	src/Spacecraft.cpp File Reference	151
6.32	src/Spacecraft.hpp File Reference	152
6.33	src/TATCException.cpp File Reference	152
6.34	src/TATCException.hpp File Reference	153
6.35	src/VisibilityReport.cpp File Reference	154
6.36	src/VisibilityReport.hpp File Reference	154
6.37	src/VisiblePOIReport.cpp File Reference	155
6.38	src/VisiblePOIReport.hpp File Reference	156





# Chapter 1

## Todo List

**Member `Attitude::InertialToReference`** (`const Rvector6 &centralBodyState`)

is this misnamed?

**Member `CoverageChecker::sc`**

Should this be an array of spacecraft?

**Member `NadirPointingAttitude::InertialToReference`** (`const Rvector6 &centralBodyState`)

is this misnamed?

**Member `Propagator::sc`**

should this be an array of sc?

**Member `Spacecraft::AddSensor`** (`Sensor *sensor`)

- check for sensor already on list!!

**Member `Spacecraft::InertialToConeClock`** (`const Rvector3 &viewVec`, `Real &cone`, `Real &clock`)

- do we need to buffer states here as well??



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

AbsoluteDate . . . . .	9
Attitude . . . . .	14
NadirPointingAttitude . . . . .	57
BaseException	
TATCException . . . . .	115
CoverageChecker . . . . .	22
Earth . . . . .	39
IntervalEventReport . . . . .	46
KeyValueStatistics . . . . .	52
LinearAlgebra . . . . .	55
OrbitState . . . . .	61
PointGroup . . . . .	66
Propagator . . . . .	74
Sensor . . . . .	91
ConicalSensor . . . . .	17
CustomSensor . . . . .	31
RectangularSensor . . . . .	86
SensorElement . . . . .	99
Spacecraft . . . . .	99
VisibilityReport . . . . .	116
VisiblePOIReport . . . . .	120





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">AbsoluteDate</a>	9
<a href="#">Attitude</a>	14
<a href="#">ConicalSensor</a>	17
<a href="#">CoverageChecker</a>	22
<a href="#">CustomSensor</a>	31
<a href="#">Earth</a>	39
<a href="#">IntervalEventReport</a>	46
<a href="#">KeyValueStatistics</a>	52
<a href="#">LinearAlgebra</a>	55
<a href="#">NadirPointingAttitude</a>	57
<a href="#">OrbitState</a>	61
<a href="#">PointGroup</a>	66
<a href="#">Propagator</a>	74
<a href="#">RectangularSensor</a>	86
<a href="#">Sensor</a>	91
<a href="#">SensorElement</a>	99
<a href="#">Spacecraft</a>	99
<a href="#">TATCException</a>	115
<a href="#">VisibilityReport</a>	116
<a href="#">VisiblePOIRreport</a>	120



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">AbsoluteDate.cpp</a>	129
src/ <a href="#">AbsoluteDate.hpp</a>	129
src/ <a href="#">Attitude.cpp</a>	130
src/ <a href="#">Attitude.hpp</a>	130
src/ <a href="#">ConicalSensor.cpp</a>	131
src/ <a href="#">ConicalSensor.hpp</a>	132
src/ <a href="#">CoverageChecker.cpp</a>	133
src/ <a href="#">CoverageChecker.hpp</a>	133
src/ <a href="#">CustomSensor.cpp</a>	134
src/ <a href="#">CustomSensor.hpp</a>	135
src/ <a href="#">Earth.cpp</a>	137
src/ <a href="#">Earth.hpp</a>	137
src/ <a href="#">IntervalEventReport.cpp</a>	138
src/ <a href="#">IntervalEventReport.hpp</a>	139
src/ <a href="#">KeyValueStatistics.cpp</a>	141
src/ <a href="#">KeyValueStatistics.hpp</a>	141
src/ <a href="#">LinearAlgebra.cpp</a>	142
src/ <a href="#">LinearAlgebra.hpp</a>	142
src/ <a href="#">NadirPointingAttitude.cpp</a>	143
src/ <a href="#">NadirPointingAttitude.hpp</a>	144
src/ <a href="#">OrbitState.cpp</a>	145
src/ <a href="#">OrbitState.hpp</a>	145
src/ <a href="#">PointGroup.cpp</a>	146
src/ <a href="#">PointGroup.hpp</a>	146
src/ <a href="#">Propagator.cpp</a>	147
src/ <a href="#">Propagator.hpp</a>	148
src/ <a href="#">RectangularSensor.cpp</a>	149
src/ <a href="#">RectangularSensor.hpp</a>	149
src/ <a href="#">Sensor.cpp</a>	150
src/ <a href="#">Sensor.hpp</a>	150
src/ <a href="#">Spacecraft.cpp</a>	151
src/ <a href="#">Spacecraft.hpp</a>	152
src/ <a href="#">TATCException.cpp</a>	152
src/ <a href="#">TATCException.hpp</a>	153
src/ <a href="#">VisibilityReport.cpp</a>	154
src/ <a href="#">VisibilityReport.hpp</a>	154
src/ <a href="#">VisiblePOIReport.cpp</a>	155
src/ <a href="#">VisiblePOIReport.hpp</a>	156



## Chapter 5

# Class Documentation

### 5.1 AbsoluteDate Class Reference

```
#include <AbsoluteDate.hpp>
```

#### Public Member Functions

- [AbsoluteDate](#) ()  
*class construction/destruction*
- [AbsoluteDate](#) (const [AbsoluteDate](#) &copy)
- [AbsoluteDate](#) & [operator=](#) (const [AbsoluteDate](#) &copy)
- virtual [~AbsoluteDate](#) ()
- virtual void [SetGregorianDate](#) (Integer year, Integer month, Integer day, Integer hour, Integer minute, Real second)  
*Set the Gregorian date.*
- virtual void [SetJulianDate](#) (Real jd)  
*Set the Julian Date.*
- virtual Real [GetJulianDate](#) () const  
*Return the Julian Date.*
- virtual Rvector6 [GetGregorianDate](#) ()  
*Return the Gregorian date.*
- virtual void [Advance](#) (Real stepInSec)  
*Advance the time by stepInSec seconds.*
- virtual [AbsoluteDate](#) \* [Clone](#) () const  
*Clone the [AbsoluteDate](#).*

#### Protected Member Functions

- Real [GregorianToJulianDate](#) (Integer year, Integer month, Integer day, Integer hour, Integer minute, Real second)  
*Compute the Julian date from the input Gregorian date.*

## Protected Attributes

- Real [currentDate](#)  
*Current date in Julian Day format.*

## Static Protected Attributes

- static const Integer [DAYS\\_PER\\_MONTH](#) [12]  
*<static> days-per-month constant*
- static const Real [JD\\_1900](#) = 2415019.5  
*<static> Julian date of 1900 constant*

### 5.1.1 Detailed Description

Definition of the [AbsoluteDate](#) class. This class represents an epoch.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 [AbsoluteDate\(\)](#) [1/2]

```
AbsoluteDate::AbsoluteDate ( )
```

class construction/destruction

Default constructor

#### 5.1.2.2 [AbsoluteDate\(\)](#) [2/2]

```
AbsoluteDate::AbsoluteDate (
    const AbsoluteDate & copy )
```

Copy constructor

Parameters

<i>copy</i>	the object to copy
-------------	--------------------

#### 5.1.2.3 [~AbsoluteDate\(\)](#)

```
AbsoluteDate::~~AbsoluteDate ( ) [virtual]
```

Destructor

### 5.1.3 Member Function Documentation

#### 5.1.3.1 Advance()

```
void AbsoluteDate::Advance (
    Real stepInSec ) [virtual]
```

Advance the time by *stepInSec* seconds.

Advances the date by the number of seconds specified.

##### Parameters

<i>stepInSec</i>	stepsize (seconds)
------------------	--------------------

#### 5.1.3.2 Clone()

```
AbsoluteDate * AbsoluteDate::Clone ( ) const [virtual]
```

Clone the [AbsoluteDate](#).

This method returns a clone of the [AbsoluteDate](#).

##### Returns

clone of the [AbsoluteDate](#).

#### 5.1.3.3 GetGregorianDate()

```
Rvector6 AbsoluteDate::GetGregorianDate ( ) [virtual]
```

Return the Gregorian date.

Returns the Gregorian date.

##### Returns

the Gregorian date

#### 5.1.3.4 GetJulianDate()

```
Real AbsoluteDate::GetJulianDate ( ) const [virtual]
```

Return the Julian Date.

Returns the Julian date.

##### Returns

the Julian date

#### 5.1.3.5 GregorianToJulianDate()

```
Real AbsoluteDate::GregorianToJulianDate (
    Integer year,
    Integer month,
    Integer day,
    Integer hour,
    Integer minute,
    Real second ) [protected]
```

Compute the Julian date from the input Gregorian date.

Converts a Gregorian date to a Julian date.

##### Parameters

<i>year</i>	the year
<i>month</i>	the month
<i>day</i>	the day
<i>hour</i>	the hour
<i>minute</i>	the minute
<i>second</i>	the seconds

##### Returns

the Julian date

#### 5.1.3.6 operator=()

```
AbsoluteDate & AbsoluteDate::operator= (
    const AbsoluteDate & copy )
```

The operator= for the AbsolutDate class.



## Parameters

<i>copy</i>	the object to copy
-------------	--------------------

## 5.1.3.7 SetGregorianDate()

```
void AbsoluteDate::SetGregorianDate (
    Integer year,
    Integer month,
    Integer day,
    Integer hour,
    Integer minute,
    Real second ) [virtual]
```

Set the Gregorian date.

Sets the Gregorian date.

## Parameters

<i>year</i>	the year
<i>month</i>	the month
<i>day</i>	the day
<i>hour</i>	the hour
<i>minute</i>	the minute
<i>second</i>	the seconds

## 5.1.3.8 SetJulianDate()

```
void AbsoluteDate::SetJulianDate (
    Real jd ) [virtual]
```

Set the Julian Date.

Sets the Julian date.

## Parameters

<i>jd</i>	the Julian date
-----------	-----------------

## 5.1.4 Member Data Documentation

#### 5.1.4.1 currentDate

```
Real AbsoluteDate::currentDate [protected]
```

Current date in Julian Day format.

#### 5.1.4.2 DAYS\_PER\_MONTH

```
const Integer AbsoluteDate::DAYS_PER_MONTH [static], [protected]
```

**Initial value:**

```
=  
{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
```

<static> days-per-month constant

Implementation of the [AbsoluteDate](#) class

#### 5.1.4.3 JD\_1900

```
const Real AbsoluteDate::JD_1900 = 2415019.5 [static], [protected]
```

<static> Julian date of 1900 constant

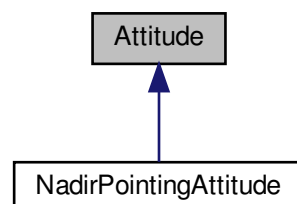
The documentation for this class was generated from the following files:

- [src/AbsoluteDate.hpp](#)
- [src/AbsoluteDate.cpp](#)

## 5.2 Attitude Class Reference

```
#include <Attitude.hpp>
```

Inheritance diagram for Attitude:



## Public Member Functions

- [Attitude](#) ()  
*class construction/destruction*
- [Attitude](#) (const [Attitude](#) &copy)
- [Attitude](#) & [operator=](#) (const [Attitude](#) &copy)
- virtual [~Attitude](#) ()
- virtual [Attitude](#) \* [Clone](#) () const  
*Clone the [Attitude](#).*
- virtual Rmatrix33 [InertialToReference](#) (const Rvector6 &centralBodyState)  
*Converts the inertial-to-reference matrix.*

### 5.2.1 Detailed Description

Definition of the [Attitude](#) class. This base class models the spacecraft attitude state.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 [Attitude\(\)](#) [1/2]

```
Attitude::Attitude ( )
```

class construction/destruction

Implementation of the base [Attitude](#) class Default constructor.

#### 5.2.2.2 [Attitude\(\)](#) [2/2]

```
Attitude::Attitude (
    const Attitude & copy )
```

Copy constructor.

Parameters

<i>copy</i>	the <a href="#">Attitude</a> object to copy
-------------	---

#### 5.2.2.3 [~Attitude\(\)](#)

```
Attitude::~~Attitude ( ) [virtual]
```

Destructor.

## 5.2.3 Member Function Documentation

### 5.2.3.1 Clone()

```
Attitude * Attitude::Clone ( ) const [virtual]
```

Clone the [Attitude](#).

This method returns a clone of the [Attitude](#).

#### Returns

clone of the [Attitude](#).

### 5.2.3.2 InertialToReference()

```
Rmatrix33 Attitude::InertialToReference (
    const Rvector6 & centralBodyState ) [virtual]
```

Converts the inertial-to-reference matrix.

**Todo** is this misnamed?

This method computes the matrix that converts from inertial to the reference frame, given the input central body state

#### Parameters

<i>centralBodyState</i>	central body state
-------------------------	--------------------

#### Returns

matrix from inertial to reference

#### Note

This method is expected to be implemented in child classes.

Reimplemented in [NadirPointingAttitude](#).

### 5.2.3.3 operator=()

```
Attitude & Attitude::operator= (
    const Attitude & copy )
```

The operator= for [Attitude](#).

## Parameters

<i>copy</i>	the <a href="#">Attitude</a> object to copy
-------------	---

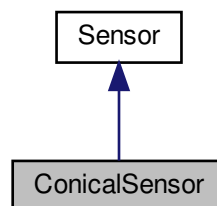
The documentation for this class was generated from the following files:

- [src/Attitude.hpp](#)
- [src/Attitude.cpp](#)

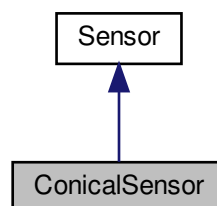
## 5.3 ConicalSensor Class Reference

```
#include <ConicalSensor.hpp>
```

Inheritance diagram for ConicalSensor:



Collaboration diagram for ConicalSensor:



## Public Member Functions

- [ConicalSensor](#) (Real fov)  
*class construction/destruction*
- [ConicalSensor](#) (const [ConicalSensor](#) &copy)
- [ConicalSensor](#) & [operator=](#) (const [ConicalSensor](#) &copy)
- virtual [~ConicalSensor](#) ()
- virtual void [SetFieldOfView](#) (Real fov)  
*Set and Get the field-of-view.*
- virtual Real [GetFieldOfView](#) ()
- virtual bool [CheckTargetVisibility](#) (Real viewConeAngle, Real viewClockAngle=0.0)

## Protected Attributes

- Real [fieldOfView](#)  
*Field-of-View (radians)*

## Additional Inherited Members

### 5.3.1 Detailed Description

Definition of the Conical [Sensor](#) class. This class models a conical sensor.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 [ConicalSensor\(\)](#) [1/2]

```
ConicalSensor::ConicalSensor (
    Real fov )
```

class construction/destruction

Implementation of the [ConicalSensor](#) class Constructor

#### Parameters

<i>fov</i>	field-of-view for the sensor (radians), sensor half-angle
------------	---

#### 5.3.2.2 [ConicalSensor\(\)](#) [2/2]

```
ConicalSensor::ConicalSensor (
    const ConicalSensor & copy )
```

Copy constructor

**Parameters**

<i>copy</i>	object to copy
-------------	----------------

**5.3.2.3 ~ConicalSensor()**

```
ConicalSensor::~~ConicalSensor ( ) [virtual]
```

**Destructor****5.3.3 Member Function Documentation****5.3.3.1 CheckTargetVisibility()**

```
bool ConicalSensor::CheckTargetVisibility (
    Real viewConeAngle,
    Real viewClockAngle = 0.0 ) [virtual]
```

Check the target visibility given the input cone and clock angles: determines whether or not the point is in the sensor FOV.

Determines whether or not the point is in the sensor FOV

**Parameters**

<i>viewConeAngle</i>	the view cone angle
<i>viewClockAngle</i>	the view clock angle <unused for="" this="" class>="">

**Returns**

true if the point is in the sensor FOV; false otherwise

Implements [Sensor](#).

**5.3.3.2 GetFieldOfView()**

```
Real ConicalSensor::GetFieldOfView ( ) [virtual]
```

Returns the field-of-view for the [ConicalSensor](#)

**Returns**

field-of-view (radians)



#### 5.3.3.3 operator=()

```
ConicalSensor & ConicalSensor::operator= (
    const ConicalSensor & copy )
```

The operator= for the [ConicalSensor](#)

##### Parameters

<i>copy</i>	object to copy
-------------	----------------

#### 5.3.3.4 SetFieldOfView()

```
void ConicalSensor::SetFieldOfView (
    Real fov ) [virtual]
```

Set and Get the field-of-view.

Sets the field-of-view for the [ConicalSensor](#)

##### Parameters

<i>fov</i>	field-of-view (radians)
------------	-------------------------

### 5.3.4 Member Data Documentation

#### 5.3.4.1 fieldOfView

```
Real ConicalSensor::fieldOfView [protected]
```

Field-of-View (radians)

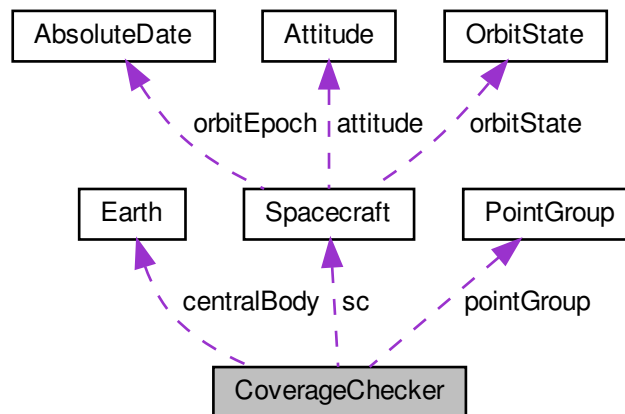
The documentation for this class was generated from the following files:

- [src/ConicalSensor.hpp](#)
- [src/ConicalSensor.cpp](#)

## 5.4 CoverageChecker Class Reference

```
#include <CoverageChecker.hpp>
```

Collaboration diagram for CoverageChecker:



### Public Member Functions

- [CoverageChecker](#) ([PointGroup](#) \*ptGroup, [Spacecraft](#) \*sat)  
*class construction/destruction*
- [CoverageChecker](#) (const [CoverageChecker](#) &copy)
- [CoverageChecker](#) & operator= (const [CoverageChecker](#) &copy)
- virtual [~CoverageChecker](#) ()
- virtual IntegerArray [CheckPointCoverage](#) (const Rvector6 &theState, Real theTime)  
*Check the point coverage and return the resulting index array.*
- virtual IntegerArray [AccumulateCoverageData](#) ()  
*Accumulate the coverage data at the current propagated time.*
- virtual IntegerArray [AccumulateCoverageData](#) (Real atTime)  
*Accumulate the coverage data at the input time.*
- virtual std::vector< [IntervalEventReport](#) > [ProcessCoverageData](#) ()  
*Process the coverage data, create reports.*
- virtual [IntervalEventReport](#) [CreateNewPOIReport](#) (Real startJd, Real endJd, Integer poiIdx)  
*Create a new POI report.*
- virtual void [SetComputePOIGeometryData](#) (bool flag)  
*Set the flag indicating whether or not to compute POI geometry data.*

### Protected Member Functions

- virtual Rvector6 [GetEarthFixedSatState](#) (Real jd, const Rvector6 &scCartState)  
*Get the [Earth](#) Fixed state at the input time for the input cartesian state.*
- virtual bool [CheckGridFeasibility](#) (Integer ptIdx, const Rvector3 &bodyFixedState)
- virtual void [CheckGridFeasibility](#) (const Rvector3 &bodyFixedState)  
*Check the grid feasibility for all points for the input body fixed state.*

## Protected Attributes

- [PointGroup](#) \* [pointGroup](#)  
*the points to use for coverage*
- [Spacecraft](#) \* [sc](#)  
*The spacecraft object.*
- [Earth](#) \* [centralBody](#)  
*the central body; the model of [Earth](#)'s properties & rotation*
- Integer [timelIdx](#)  
*the number of accumulated propagation data points // ???*
- `std::vector< IntegerArray >` [timeSeriesData](#)  
*times when points are visible*
- `std::vector< std::vector< VisiblePOIRreport > >` [discreteEventData](#)  
*discrete event data*
- RealArray [dateData](#)  
*the date of each propagation point*
- IntegerArray [numEventsPerPoint](#)  
*the number of propagation times when each point was visible*
- `std::vector< Rvector3 * >` [pointArray](#)  
*array of all points*
- `std::vector< bool >` [feasibilityTest](#)  
*feasibility values for each point*
- bool [computePOIGeometryData](#)  
*flag indicating if observer and sun geometry should be computed*
- Real [coverageStart](#)  
*Start time of the coverage.*
- Real [coverageEnd](#)  
*End time of the coverage.*
- Rvector3 [rangeVec](#)
- Rvector3 [bfState](#)
- Rvector3 [bodyUnit](#)
- Rvector3 [ptPos](#)

## Static Protected Attributes

- static const Real [BODY\\_RADIUS](#) = 6378.1363  
*<static const>=""> body radius*

### 5.4.1 Detailed Description

Definition of the coverage checker class. This class checks for point coverage and generates reports.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 CoverageChecker() [1/2]

```
CoverageChecker::CoverageChecker (
    PointGroup * ptGroup,
    Spacecraft * sat )
```

class construction/destruction

Constructor



**5.4.3.2 AccumulateCoverageData()** [2/2]

```
IntegerArray CoverageChecker::AccumulateCoverageData (
    Real atTime ) [virtual]
```

Accumulate the coverage data at the input time.

Accumulates the coverage data after the propagation update

**Returns**

array of indexes

**5.4.3.3 CheckGridFeasibility()** [1/2]

```
bool CoverageChecker::CheckGridFeasibility (
    Integer ptIdx,
    const Rvector3 & bodyFixedState ) [protected], [virtual]
```

Check the grid feasibility for the input point with the input body fixed state

Checks the grid feasibility

**Parameters**

<i>ptIdx</i>	point index
<i>bodyFixedState</i>	input body fixed state

**Returns**

output feasibility flag

**5.4.3.4 CheckGridFeasibility()** [2/2]

```
void CoverageChecker::CheckGridFeasibility (
    const Rvector3 & bodyFixedState ) [protected], [virtual]
```

Check the grid feasibility for all points for the input body fixed state.

Checks the grid feasibility

**Parameters**

<i>bodyFixedState</i>	input body fixed state
-----------------------	------------------------

#### 5.4.3.5 CheckPointCoverage()

```
IntegerArray CoverageChecker::CheckPointCoverage (
    const Rvector6 & theState,
    Real theTime ) [virtual]
```

Check the point coverage and return the resulting index array.

Checks the point coverage.

##### Returns

array of indexes

#### 5.4.3.6 CreateNewPOIReport()

```
IntervalEventReport CoverageChecker::CreateNewPOIReport (
    Real startJd,
    Real endJd,
    Integer poiIdx ) [virtual]
```

Create a new POI report.

Creates a new report of coverage data.

##### Parameters

<i>startJd</i>	start Julian date for the reportSetComputePOIGeometryData
<i>endJd</i>	end Julian date for the report
<i>poiIndex</i>	POI index for the created report

##### Returns

report of coverage

#### 5.4.3.7 GetEarthFixedSatState()

```
Rvector6 CoverageChecker::GetEarthFixedSatState (
    Real jd,
    const Rvector6 & scCartState ) [protected], [virtual]
```

Get the [Earth](#) Fixed state at the input time for the input cartesian state.

Returns the Earth-Fixed state at the specified time

**Parameters**

<i>jd</i>	Julian date
-----------	-------------

**Returns**

earth-fixed state at the input time

**5.4.3.8 operator=()**

```
CoverageChecker & CoverageChecker::operator= (
    const CoverageChecker & copy )
```

The operator= for the [CoverageChecker](#) object

**Parameters**

<i>copy</i>	the object to copy
-------------	--------------------

**5.4.3.9 ProcessCoverageData()**

```
std::vector< IntervalEventReport > CoverageChecker::ProcessCoverageData ( ) [virtual]
```

Process the coverate data, create reports.

Returns an array of reports of coverage

**Returns**

array of reports of coverage

**5.4.3.10 SetComputePOIGeometryData()**

```
void CoverageChecker::SetComputePOIGeometryData (
    bool flag ) [virtual]
```

Set the flag indicating whether or not to compute POI geometry data.

Sets the flag indicating whether or not to compute the POI Geometry data

## Parameters

<i>flag</i>	compute the POI geometry data?
-------------	--------------------------------

## 5.4.4 Member Data Documentation

### 5.4.4.1 bfState

`Rvector3 CoverageChecker::bfState` [protected]

### 5.4.4.2 BODY\_RADIUS

`const Real CoverageChecker::BODY_RADIUS = 6378.1363` [static], [protected]

<static const>=""> body radius

Implementation of the [CoverageChecker](#) class

### 5.4.4.3 bodyUnit

`Rvector3 CoverageChecker::bodyUnit` [protected]

### 5.4.4.4 centralBody

`Earth* CoverageChecker::centralBody` [protected]

the central body; the model of [Earth](#)'s properties & rotation

### 5.4.4.5 computePOIGeometryData

`bool CoverageChecker::computePOIGeometryData` [protected]

flag indicating if observer and sun geometry should be computed



#### 5.4.4.6 coverageEnd

Real CoverageChecker::coverageEnd [protected]

End time of the coverage.

#### 5.4.4.7 coverageStart

Real CoverageChecker::coverageStart [protected]

Start time of the coverage.

#### 5.4.4.8 dateData

RealArray CoverageChecker::dateData [protected]

the date of each propagation point

#### 5.4.4.9 discreteEventData

std::vector<std::vector<VisiblePOIRreport> > CoverageChecker::discreteEventData [protected]

discrete event data

#### 5.4.4.10 feasibilityTest

std::vector<bool> CoverageChecker::feasibilityTest [protected]

feasibility values for each point

#### 5.4.4.11 numEventsPerPoint

IntegerArray CoverageChecker::numEventsPerPoint [protected]

the number of propagation times when each point was visible

#### 5.4.4.12 pointArray

```
std::vector<Rvector3*> CoverageChecker::pointArray [protected]
```

array of all points

#### 5.4.4.13 pointGroup

```
PointGroup* CoverageChecker::pointGroup [protected]
```

the points to use for coverage

#### 5.4.4.14 ptPos

```
Rvector3 CoverageChecker::ptPos [protected]
```

#### 5.4.4.15 rangeVec

```
Rvector3 CoverageChecker::rangeVec [protected]
```

local Rvectors used for Grid Feasibility calculations (for performance)

#### 5.4.4.16 sc

```
Spacecraft* CoverageChecker::sc [protected]
```

The spacecraft object.

**Todo** Should this be an array of spacecraft?

#### 5.4.4.17 timeIdx

```
Integer CoverageChecker::timeIdx [protected]
```

the number of accumulated propagation data points // ???

## 5.4.4.18 timeSeriesData

```
std::vector<IntegerArray> CoverageChecker::timeSeriesData [protected]
```

times when points are visible

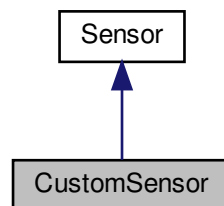
The documentation for this class was generated from the following files:

- [src/CoverageChecker.hpp](#)
- [src/CoverageChecker.cpp](#)

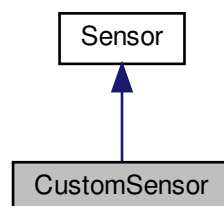
## 5.5 CustomSensor Class Reference

```
#include <CustomSensor.hpp>
```

Inheritance diagram for CustomSensor:



Collaboration diagram for CustomSensor:



## Public Member Functions

- [CustomSensor](#) (const Rvector &coneAngleVecIn, const Rvector &clockAngleVecIn)  
*class construction/destruction*
- [CustomSensor](#) (const [CustomSensor](#) &copy)
- [CustomSensor](#) & operator= (const [CustomSensor](#) &copy)
- virtual [~CustomSensor](#) ()
- bool [CheckTargetVisibility](#) (Real viewConeAngle, Real viewClockAngle)
- bool [CheckRegionVisibility](#) (const Rvector &coneAngleVec, const Rvector &clockAngleVec)

## Protected Member Functions

- bool [CheckTargetMaxExcursionCoordinates](#) (Real xCoord, Real yCoord)  
*class hidden methods used by constructor*
- Rmatrix [PointsToSegments](#) (const Rvector &xCoords, const Rvector &yCoords)
- void [ComputeExternalPoints](#) ()
- bool [RegionsFullyContained](#) (std::vector< IntegerArray > &adjacency)  
*helper methods for checkRegionVisibility()*
- void [Sort](#) (Rvector &v, bool ascending=true)  
*rVector utilities*
- void [Sort](#) (Rvector &v, IntegerArray &indices, bool ascending=true)
- Real [Max](#) (const Rvector &v)
- Real [Min](#) (const Rvector &v)

## Protected Attributes

- Integer [numFOVPoints](#)  
*data computed from constructor inputs*
- Rvector [coneAngleVec](#)
- Rvector [clockAngleVec](#)
- Rvector [xProjectionCoordArray](#)  
*stereographic projection of the numFOVpoints cone & clock angles*
- Rvector [yProjectionCoordArray](#)
- Rmatrix [segmentArray](#)
- Integer [numTestPoints](#)  
*test points computed in [ComputeExternalPoints\(\)](#)*
- Rmatrix [externalPointArray](#)
- Real [maxXExcursion](#)  
*maximum and minimum values for x and y values in stereographic projection*
- Real [minXExcursion](#)
- Real [maxYExcursion](#)
- Real [minYExcursion](#)

### 5.5.1 Detailed Description

Implementation of the [CustomSensor](#) class

### 5.5.2 Constructor & Destructor Documentation

## 5.5.2.1 CustomSensor() [1/2]

```
CustomSensor::CustomSensor (
    const Rvector & coneAngleVecIn,
    const Rvector & clockAngleVecIn )
```

class construction/destruction

### Constructor

coneAngleVec and clockAngleVec contain pairs of angles that describe the sensor FOV. coneAngleVec[0] is paired with clockAngleVec[0], coneAngleVec[1] is paired with clockAngleVec[1] and so on. The last point in each arrays should be the same as the first point to ensure FOV closure.

#### Parameters

<i>coneAngleVec</i>	array of cone angles measured from +Z sensor axis (rad) if xP,yP,zP is a UNIT vector describing a FOV point, then the cone angle for the point is $\pi/2 - \text{asin}(zP)$ ;
<i>clockAngleVec</i>	array of clock angles (right ascensions) rad measured clockwise from the + X-axis. if xP,yP,zP is a UNIT vector describing a FOV point, then the clock angle for the point is $\text{atan2}(y,x)$ ;

## 5.5.2.2 CustomSensor() [2/2]

```
CustomSensor::CustomSensor (
    const CustomSensor & copy )
```

Copy constructor

#### Parameters

<i>copy</i>	object to copy
-------------	----------------

## 5.5.2.3 ~CustomSensor()

```
CustomSensor::~CustomSensor ( ) [virtual]
```

## 5.5.3 Member Function Documentation

## 5.5.3.1 CheckRegionVisibility()

```
bool CustomSensor::CheckRegionVisibility (
    const Rvector & coneAngleVec,
    const Rvector & clockAngleVec )
```

### 5.5.3.2 CheckTargetMaxExcursionCoordinates()

```
bool CustomSensor::CheckTargetMaxExcursionCoordinates (
    Real xCoord,
    Real yCoord ) [protected]
```

class hidden methods used by constructor

### 5.5.3.3 CheckTargetVisibility()

```
bool CustomSensor::CheckTargetVisibility (
    Real viewConeAngle,
    Real viewClockAngle ) [virtual]
```

visibility methods Check the target visibility given the input cone and clock angles: determines whether or not the point is in the sensor FOV.

Implements [Sensor](#).

### 5.5.3.4 ComputeExternalPoints()

```
void CustomSensor::ComputeExternalPoints ( ) [protected]
```

### 5.5.3.5 Max()

```
Real CustomSensor::Max (
    const Rvector & v ) [protected]
```

returns maximum value from an Rvector

#### Parameters

<i>v</i>	vector containing real values to select maximum from
----------	--

#### Returns

maximum value contained in vector

### 5.5.3.6 Min()

```
Real CustomSensor::Min (
    const Rvector & v ) [protected]
```

returns minimum value from an Rvector

**Parameters**

<i>v</i>	vector containing real values to select minimum from
----------	--

**Returns**

minimum value contained in vector

**5.5.3.7 operator=()**

```
CustomSensor & CustomSensor::operator= (
    const CustomSensor & copy )
```

operator= for CustomSensor

**Parameters**

<i>copy</i>	object to copy
-------------	----------------

**5.5.3.8 PointsToSegments()**

```
Rmatrix CustomSensor::PointsToSegments (
    const Rvector & xCoords,
    const Rvector & yCoords ) [protected]
```

**5.5.3.9 RegionIsFullyContained()**

```
bool CustomSensor::RegionIsFullyContained (
    std::vector< IntegerArray > & adjacency ) [protected]
```

helper methods for checkRegionVisibility()

**5.5.3.10 Sort()** [1/2]

```
void CustomSensor::Sort (
    Rvector & v,
    bool ascending = true ) [protected]
```

rVector utilities



#### 5.5.3.11 Sort() [2/2]

```
void CustomSensor::Sort (  
    Rvector & v,  
    IntegerArray & indices,  
    bool ascending = true ) [protected]
```

### 5.5.4 Member Data Documentation

#### 5.5.4.1 clockAngleVec

```
Rvector CustomSensor::clockAngleVec [protected]
```

#### 5.5.4.2 coneAngleVec

```
Rvector CustomSensor::coneAngleVec [protected]
```

#### 5.5.4.3 externalPointArray

```
Rmatrix CustomSensor::externalPointArray [protected]
```

#### 5.5.4.4 maxXExcursion

```
Real CustomSensor::maxXExcursion [protected]
```

maximum and minimum values for x and y values in stereographic projection

#### 5.5.4.5 maxYExcursion

```
Real CustomSensor::maxYExcursion [protected]
```

#### 5.5.4.6 minXExcursion

`Real CustomSensor::minXExcursion [protected]`

#### 5.5.4.7 minYExcursion

`Real CustomSensor::minYExcursion [protected]`

#### 5.5.4.8 numFOVPoints

`Integer CustomSensor::numFOVPoints [protected]`

data computed from constructor inputs

#### 5.5.4.9 numTestPoints

`Integer CustomSensor::numTestPoints [protected]`

test points computed in [ComputeExternalPoints\(\)](#)

#### 5.5.4.10 segmentArray

`Rmatrix CustomSensor::segmentArray [protected]`

#### 5.5.4.11 xProjectionCoordArray

`Rvector CustomSensor::xProjectionCoordArray [protected]`

stereographic projection of the numFOVpoints cone & clock angles

#### 5.5.4.12 yProjectionCoordArray

`Rvector CustomSensor::yProjectionCoordArray [protected]`

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

- [src/CustomSensor.hpp](#)
- [src/CustomSensor.cpp](#)

## 5.6 Earth Class Reference

```
#include <Earth.hpp>
```

### Public Member Functions

- [Earth](#) ()  
*class construction/destruction*
- [Earth](#) (const [Earth](#) &copy)
- [Earth](#) & [operator=](#) (const [Earth](#) &copy)
- virtual [~Earth](#) ()
- virtual Rmatrix33 [GetInertialToFixedRotation](#) (Real jd)  
*Get the inertial-to-fixed rotation matrix.*
- virtual Real [ComputeGMT](#) (Real jd)  
*Compute the Greenwich Mean Time.*
- virtual Rvector3 [GetBodyFixedState](#) (Rvector3 inertialState, Real jd)  
*Get the body-fixed state.*
- virtual Rvector3 [Convert](#) (const Rvector3 &origValue, const std::string &fromType, const std::string &toType)  
*Convert between body-fixed representations.*
- virtual Rvector3 [InertialToBodyFixed](#) (const Rvector3 &inertialVector, Real jd, const std::string &toType)  
*Convert the input vector from inertial to body-fixed.*
- virtual Rvector3 [FixedToTopocentric](#) (const Rvector3 &inertialVector, const Real lat, const Real lon)  
*Convert the input vecgor from body-fixed to topocentric.*
- virtual Rvector3 [GetSunPositionInBodyCoords](#) (Real jd, const std::string &toType)  
*Get the Sun position in body coordinates.*
- virtual Rmatrix33 [FixedToTopo](#) (Real gdLat, Real gdLon)  
*Compute the body-fixed-to-topocentric rotaiton matrix.*
- virtual Real [GeocentricToGeodeticLat](#) (Real gcLat)  
*Convert geocentric latitude to geodetic latitude.*
- virtual void [GetEarthSunDistRaDec](#) (Real jd, Rvector3 &rSun, Real &rtAsc, Real &decl)  
*Get the Earth-Sun distance.*
- Real [GetRadius](#) ()  
*Get the mean equatorial radius.*

### Protected Attributes

- Real [J2](#)  
*J2 term for [Earth](#).*
- Real [mu](#)  
*Gravitational parameter of the [Earth](#).*
- Real [radius](#)  
*Equatorial radius of the [Earth](#).*
- Real [flattening](#)  
*Flattening of the [Earth](#).*
- Rmatrix33 [rotationResult](#)  
*Local class data for performance.*
- Real [lastRotationTime](#)  
*Save the last computd rotation time, for performance.*

### 5.6.1 Detailed Description

Definition of the [Earth](#) class. This class is a simple model of the [Earth](#).

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 [Earth\(\)](#) [1/2]

```
Earth::Earth ( )
```

class construction/destruction

Implementation of the [Earth](#) class Default constructor.

#### 5.6.2.2 [Earth\(\)](#) [2/2]

```
Earth::Earth (
    const Earth & copy )
```

Copy constructor.

##### Parameters

<i>copy</i>	the <a href="#">Earth</a> object to copy
-------------	--

#### 5.6.2.3 [~Earth\(\)](#)

```
Earth::~~Earth ( ) [virtual]
```

Detructor

### 5.6.3 Member Function Documentation

#### 5.6.3.1 [ComputeGMT\(\)](#)

```
Real Earth::ComputeGMT (
    Real jd ) [virtual]
```

Compute the Greenwich Mean Time.

Returns the GMT.

## Parameters

<i>jd</i>	the Julian date at which to compute the GMT.
-----------	--

## Returns

GMT

## 5.6.3.2 Convert()

```
Rvector3 Earth::Convert (
    const Rvector3 & origValue,
    const std::string & fromType,
    const std::string & toType ) [virtual]
```

Convert between body-fixed representations.

Conversion method between body fixed representations. Valid values for &origvalue are "Cartesian", "Spherical", and "Ellipsoid"

## Parameters

<i>origValue</i>	data in given representation
<i>fromType</i>	representation from which to convert
<i>toType</i>	representation to which to convert

## Returns

Converted state from the specified "from" representation to the specified "to" representation

## 5.6.3.3 FixedToTopo()

```
Rmatrix33 Earth::FixedToTopo (
    Real gdLat,
    Real gdLon ) [virtual]
```

Compute the body-fixed-to-topocentric rotaiton matrix.

Returns the rotation matrix to convert from body-fixed to topocentric.

## Parameters

<i>gdLat</i>	the geodetic latitude
<i>gdLon</i>	the geodetic longitude

**Returns**

the rotation matrix from body-fixed to topocentric

**5.6.3.4 FixedToTopocentric()**

```
Rvector3 Earth::FixedToTopocentric (
    const Rvector3 & bodyFixedVector,
    const Real lat,
    const Real lon ) [virtual]
```

Convert the input vector from body-fixed to topocentric.

Converts the input vector from body-fixed to topocentric

**Parameters**

<i>bodyFixedVector</i>	input vector in body-fixed
<i>lat</i>	latitude
<i>lon</i>	longitude

**Returns**

vector in topocentric coordinates

**5.6.3.5 GeocentricToGeodeticLat()**

```
Real Earth::GeocentricToGeodeticLat (
    Real gcLat ) [virtual]
```

Convert geocentric latitude to geodetic latitude.

Converts from a geocentric latitude to a geodetic latitude.

**Parameters**

<i>gcLat</i>	the geocentric latitude
--------------	-------------------------

**Returns**

the geodetic latitude

## 5.6.3.6 GetBodyFixedState()

```
Rvector3 Earth::GetBodyFixedState (
    Rvector3 inertialState,
    Real jd ) [virtual]
```

Get the body-fixed state.

Returns the body-fixed state given the inertial stat and the time.

## Parameters

<i>inertialState</i>	the inertial state.
<i>jd</i>	the Julian date at which to compute the body-fixed state.

## Returns

body-fixed state

## 5.6.3.7 GetEarthSunDistRaDec()

```
void Earth::GetEarthSunDistRaDec (
    Real jd,
    Rvector3 & rSun,
    Real & rtAsc,
    Real & decl ) [virtual]
```

Get the Earth-Sun distance.

Returns the Earth-Sun distance.

## Parameters

<i>jd</i>	[in] the Julian data at which to compute the distance
<i>rSun</i>	[out] the Earth-to-Sun vector
<i>rtAsc</i>	[out] right ascension
<i>decl</i>	[out] declination

## 5.6.3.8 GetInertialToFixedRotation()

```
Rmatrix33 Earth::GetInertialToFixedRotation (
    Real jd ) [virtual]
```

Get the inertial-to-fixed rotation matrix.

Returns the inertial-to-fixed rotation matrix.

**Parameters**

<i>jd</i>	the Julian date at which to compute the rotation matrix.
-----------	--

**Returns**

inertial-to-fixed rotation matrix

**5.6.3.9 GetRadius()**

```
Real Earth::GetRadius ( )
```

Get the mean equatorial radius.

Returns the mean equatorial radius.

**Returns**

the mean equatorial radius

**5.6.3.10 GetSunPositionInBodyCoords()**

```
Rvector3 Earth::GetSunPositionInBodyCoords (
    Real jd,
    const std::string & toType ) [virtual]
```

Get the Sun position in body coordinates.

Computes sun position in body coordinates returning position in requested representation.

**Parameters**

<i>jd</i>	Julian date
<i>toType</i>	Output representation. Valid values for are "Cartesian", "Spherical", and "Ellipsoid"

**Returns**

Sun position in body coordinates in requested representation

**5.6.3.11 InertialToBodyFixed()**

```
Rvector3 Earth::InertialToBodyFixed (
    const Rvector3 & inertialVector,
```



```
Real jd,
const std::string & toType ) [virtual]
```

Convert the input vector from inertial to body-fixed.

Conversion method between inertial and body fixed representations. Valid values for &toType are "Cartesian", "Spherical", and "Ellipsoid"

#### Parameters

<i>inertialVector</i>	inertial Cartesian vector
<i>jd</i>	Julian date associated with <inertialVector>
<i>toType</i>	representation to which to convert

#### Returns

Converted state from the inertial Cartesian input to the specified "to" representation

#### 5.6.3.12 operator=()

```
Earth & Earth::operator= (
    const Earth & copy )
```

The operator= for the [Earth](#).

#### Parameters

<i>copy</i>	the <a href="#">Earth</a> object to copy
-------------	--

### 5.6.4 Member Data Documentation

#### 5.6.4.1 flattening

```
Real Earth::flattening [protected]
```

Flattening of the [Earth](#).

#### 5.6.4.2 J2

```
Real Earth::J2 [protected]
```

J2 term for [Earth](#).

#### 5.6.4.3 lastRotationTime

```
Real Earth::lastRotationTime [protected]
```

Save the last computed rotation time, for performance.

#### 5.6.4.4 mu

```
Real Earth::mu [protected]
```

Gravitational parameter of the [Earth](#).

#### 5.6.4.5 radius

```
Real Earth::radius [protected]
```

Equatorial radius of the [Earth](#).

#### 5.6.4.6 rotationResult

```
Rmatrix33 Earth::rotationResult [protected]
```

Local class data for performance.

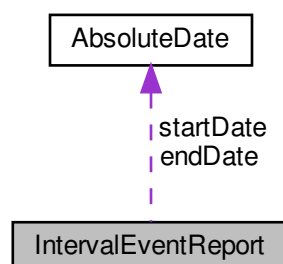
The documentation for this class was generated from the following files:

- [src/Earth.hpp](#)
- [src/Earth.cpp](#)

## 5.7 IntervalEventReport Class Reference

```
#include <IntervalEventReport.hpp>
```

Collaboration diagram for IntervalEventReport:



## Public Member Functions

- [IntervalEventReport](#) ()  
*class construction/destruction*
- [IntervalEventReport](#) (const [IntervalEventReport](#) &copy)
- [IntervalEventReport](#) & operator= (const [IntervalEventReport](#) &copy)
- virtual ~[IntervalEventReport](#) ()
- virtual void [SetStartDate](#) (const [AbsoluteDate](#) &toDate)  
*Set the start date.*
- virtual void [SetEndDate](#) (const [AbsoluteDate](#) &toDate)  
*Set the end date.*
- virtual const [AbsoluteDate](#) & [GetStartDate](#) ()  
*Get the start date.*
- virtual const [AbsoluteDate](#) & [GetEndDate](#) ()  
*Get the end date.*
- virtual void [SetPOIIndex](#) (Integer toldx)  
*Set the POI index.*
- virtual Integer [GetPOIIndex](#) ()  
*Get the POI index.*
- void [AddPOIEvent](#) (const [VisiblePOIReport](#) &theReport)  
*Add a POI event.*
- virtual std::vector< [VisiblePOIReport](#) > [GetPOIEvents](#) ()  
*Get an array of POI event reports.*
- void [SetAllPOIEvents](#) (std::vector< [VisiblePOIReport](#) >)  
*Set all of the POI events at once.*

## Protected Attributes

- Integer [poiIndex](#)  
*Index of point of interest.*
- [AbsoluteDate](#) [startDate](#)  
*Start date of the interval event.*
- [AbsoluteDate](#) [endDate](#)  
*End date of the interval event.*
- std::vector< [VisiblePOIReport](#) > [discretePOIEvents](#)  
*Vector of discrete event reports (VisiblePOIReports)*

### 5.7.1 Detailed Description

Definition of the Interval Event Report class.

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 IntervalEventReport() [1/2]

```
IntervalEventReport::IntervalEventReport ( )
```

class construction/destruction

Implementation of the Interval Event Report class. Constructs [IntervalEventReport](#) instance (default constructor).

## Parameters

<details>	A message providing the details of the exception.
-----------	---

## 5.7.2.2 IntervalEventReport() [2/2]

```
IntervalEventReport::IntervalEventReport (
    const IntervalEventReport & copy )
```

Constructs [IntervalEventReport](#) instance (copy constructor).

## Parameters

<i>be</i>	The instance that is copied.
-----------	------------------------------

## 5.7.2.3 ~IntervalEventReport()

```
IntervalEventReport::~IntervalEventReport ( ) [virtual]
```

Destructs [IntervalEventReport](#) instance

## 5.7.3 Member Function Documentation

## 5.7.3.1 AddPOIEvent()

```
void IntervalEventReport::AddPOIEvent (
    const VisiblePOIReport & theReport )
```

Add a POI event.

Appends a [VisiblePOIReport](#) to the discrtePOIEvents std::vector

## Parameters

<a href="#">VisiblePOIReport</a>	report.
----------------------------------	---------

#### 5.7.3.2 GetEndDate()

```
const AbsoluteDate & IntervalEventReport::GetEndDate ( ) [virtual]
```

Get the end date.

Returns the end date for the report.

##### Returns

The end time for the report.

#### 5.7.3.3 GetPOIEvents()

```
std::vector< VisiblePOIReport > IntervalEventReport::GetPOIEvents ( ) [virtual]
```

Get an array of POI event reports.

Returns the vector of VisiblePOIReports

##### Returns

The vector of VisiblePOIReports

#### 5.7.3.4 GetPOIIndex()

```
Integer IntervalEventReport::GetPOIIndex ( ) [virtual]
```

Get the POI index.

Returns the POI Index for the report.

##### Returns

<toIdx> Index for the report

#### 5.7.3.5 GetStartDate()

```
const AbsoluteDate & IntervalEventReport::GetStartDate ( ) [virtual]
```

Get the start date.

Returns the start date for the report.

##### Returns

The start time for the report.

#### 5.7.3.6 operator=()

```
IntervalEventReport & IntervalEventReport::operator= (
    const IntervalEventReport & copy )
```

IntervalEventReport operator=.

## Parameters

<i>be</i>	The instance that is copied.
-----------	------------------------------

## 5.7.3.7 SetAllPOIEvents()

```
void IntervalEventReport::SetAllPOIEvents (
    std::vector< VisiblePOIReport > theEvents )
```

Set all of the POI events at once.

Sets the entire discrtePOIEvents std::vector

## Parameters

<i>std::vector&lt; VisiblePOIReport&gt;.</i>	
--	--

## 5.7.3.8 SetEndDate()

```
void IntervalEventReport::SetEndDate (
    const AbsoluteDate & toDate ) [virtual]
```

Set the end date.

Sets the end date for the report.

## Parameters

<i>toDate</i>	The end time for the report.
---------------	------------------------------

## 5.7.3.9 SetPOIIndex()

```
void IntervalEventReport::SetPOIIndex (
    Integer toIdx ) [virtual]
```

Set the POI index.

Sets the POI Index for the report.

## Parameters

<i>&lt;toIdx&gt;</i>	Index for the report
----------------------	----------------------

### 5.7.3.10 SetStartDate()

```
void IntervalEventReport::SetStartDate (
    const AbsoluteDate & toDate ) [virtual]
```

Set the start date.

Sets the start date for the report.

#### Parameters

<i>toDate</i>	The start time for the report.
---------------	--------------------------------

## 5.7.4 Member Data Documentation

### 5.7.4.1 discretePOIEvents

```
std::vector<VisiblePOIReport> IntervalEventReport::discretePOIEvents [protected]
```

Vector of discrete event reports (VisiblePOIReports)

### 5.7.4.2 endDate

```
AbsoluteDate IntervalEventReport::endDate [protected]
```

End date of the interval event.

### 5.7.4.3 poiIndex

```
Integer IntervalEventReport::poiIndex [protected]
```

Index of point of interest.

#### 5.7.4.4 startDate

`AbsoluteDate` `IntervalEventReport::startDate` [protected]

Start date of the interval event.

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

- `src/IntervalEventReport.hpp`
- `src/IntervalEventReport.cpp`

## 5.8 KeyValueStatistics Class Reference

```
#include <KeyValueStatistics.hpp>
```

### Public Member Functions

- `KeyValueStatistics` (`Real minVal`, `Real maxVal`, `Real avgVal`)  
*class construction/destruction*
- `KeyValueStatistics` (`const KeyValueStatistics &copy`)
- `KeyValueStatistics & operator=` (`const KeyValueStatistics &copy`)
- `virtual ~KeyValueStatistics` ()
- `virtual Real GetMinValue` ()
- `virtual Real GetMaxValue` ()
- `virtual Real GetAvgValue` ()

### Protected Attributes

- `Real minValue`  
*Minimum value.*
- `Real maxValue`  
*Maximum value.*
- `Real avgValue`  
*Average value.*

#### 5.8.1 Detailed Description

Definition of the `KeyValueStatistics` class.

#### 5.8.2 Constructor & Destructor Documentation



#### 5.8.2.1 KeyValueStatistics() [1/2]

```
KeyValueStatistics::KeyValueStatistics (
    Real minVal,
    Real maxVal,
    Real avgVal )
```

class construction/destruction

Implementation of [KeyValueStatistics](#) class. Default constructor.

#### 5.8.2.2 KeyValueStatistics() [2/2]

```
KeyValueStatistics::KeyValueStatistics (
    const KeyValueStatistics & copy )
```

Copy constructor.

Parameters

<i>copy</i>	the object to copy
-------------	--------------------

#### 5.8.2.3 ~KeyValueStatistics()

```
KeyValueStatistics::~~KeyValueStatistics ( ) [virtual]
```

Destructor.

### 5.8.3 Member Function Documentation

#### 5.8.3.1 GetAvgValue()

```
Real KeyValueStatistics::GetAvgValue ( ) [virtual]
```

Returns the average value.

Returns

the average value

#### 5.8.3.2 GetMaxValue()

```
Real KeyValueStatistics::GetMaxValue ( ) [virtual]
```

Returns the maximum value.

##### Returns

the maximum value

#### 5.8.3.3 GetMinValue()

```
Real KeyValueStatistics::GetMinValue ( ) [virtual]
```

Returns the minimum value.

##### Returns

the minimum value

#### 5.8.3.4 operator=()

```
KeyValueStatistics & KeyValueStatistics::operator= (
    const KeyValueStatistics & copy )
```

The operator= for the [KeyValueStatistics](#) class.

##### Parameters

<i>copy</i>	the object to copy
-------------	--------------------

### 5.8.4 Member Data Documentation

#### 5.8.4.1 avgValue

```
Real KeyValueStatistics::avgValue [protected]
```

Average value.

#### 5.8.4.2 maxValue

Real KeyValueStatistics::maxValue [protected]

Maximum value.

#### 5.8.4.3 minValue

Real KeyValueStatistics::minValue [protected]

Minimum value.

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

- [src/KeyValueStatistics.hpp](#)
- [src/KeyValueStatistics.cpp](#)

## 5.9 LinearAlgebra Class Reference

```
#include <LinearAlgebra.hpp>
```

### Static Public Member Functions

- static void [LineSegmentIntersect](#) (const Rmatrix &XY1, const Rmatrix &XY2, std::vector< IntegerArray > &adjacency, Rmatrix &matrixX, Rmatrix &matrixY, Rmatrix &distance1To2, Rmatrix &distance2To1, std::vector< IntegerArray > &parallelAdjacency, std::vector< IntegerArray > &coincidentAdjacency)  
*Set the Gregorian date.*

### 5.9.1 Detailed Description

Definition of the [LinearAlgebra](#) class. NOTE: This is a static class: No instances of this class may be declared.

### 5.9.2 Member Function Documentation

#### 5.9.2.1 LineSegmentIntersect()

```
void LinearAlgebra::LineSegmentIntersect (
    const Rmatrix & XY1,
    const Rmatrix & XY2,
    std::vector< IntegerArray > & adjacency,
    Rmatrix & matrixX,
    Rmatrix & matrixY,
    Rmatrix & distance1To2,
    Rmatrix & distance2To1,
    std::vector< IntegerArray > & parallelAdjacency,
    std::vector< IntegerArray > & coincidentAdjacency ) [static]
```

Set the Gregorian date.

Implementation of the [LinearAlgebra](#) class This method finds the 2D Cartesian Coordinates of intersection points between the set of line segments given in XY1 and XY2

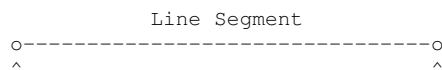
## Parameters

<i>XY1</i>	array of line segments - each line is (x1, y1, x2, y2) where (x1,y1) is the start point and (x2,y2) is the end
<i>XY2</i>	array of line segments - each line is (x1, y1, x2, y2) where (x1,y1) is the start point and (x2,y2) is the end
<i>adjacency</i>	[output] entry (i,j) is 1 if line segments XY1(i,*) and XY2(j,*) intersect; 0 otherwise
<i>matrixX</i>	[output] entry (i,j) is the X coordinate of the intersection point between line segments XY1(i,*) and XY2(j,*)
<i>matrixY</i>	[output] entry (i,j) is the Y coordinate of the intersection point between line segments XY1(i,*) and XY2(j,*)
<i>distance1To2</i>	[output] entry (i,j) is the normalized distance from the start point of the line segment XY1(i,*) to the intersection point with XY2(j,*)
<i>distance2To1</i>	[output] entry (i,j) is the normalized distance from the start point of the line segment XY1(j,*) to the intersection point with XY2(i,*)
<i>parallelAdjacency</i>	[output] entry (i,j) is 1 if line segments XY1(i,*) and XY2(j,*) are parallel; 0 otherwise
<i>coincidentAdjacency</i>	[output] entry (i,j) is 1 if line segments XY1(i,*) and XY2(j,*) are coincident; 0 otherwise

## Note

Notes from original MATLAB: function out = lineSegmentIntersect(XY1,XY2) LINESEGMENTINTERSECT  
Intersections of line segments. OUT = LINESEGMENTINTERSECT(XY1,XY2) finds the 2D Cartesian Coordinates of intersection points between the set of line segments given in XY1 and XY2.

XY1 and XY2 are N1x4 and N2x4 matrices. Rows correspond to line segments. Each row is of the form [x1 y1 x2 y2] where (x1,y1) is the start point and (x2,y2) is the end point of a line segment:



(x1,y1) (x2,y2)

OUT is a structure with fields:

'intAdjacencyMatrix' : N1xN2 indicator matrix where the entry (i,j) is 1 if line segments XY1(i,:) and XY2(j,:) intersect.

'intMatrixX' : N1xN2 matrix where the entry (i,j) is the X coordinate of the intersection point between line segments XY1(i,:) and XY2(j,:).

'intMatrixY' : N1xN2 matrix where the entry (i,j) is the Y coordinate of the intersection point between line segments XY1(i,:) and XY2(j,:).

'intNormalizedDistance1To2' : N1xN2 matrix where the (i,j) entry is the normalized distance from the start point of line segment XY1(i,:) to the intersection point with XY2(j,:).

'intNormalizedDistance2To1' : N1xN2 matrix where the (i,j) entry is the normalized distance from the start point of line segment XY1(j,:) to the intersection point with XY2(i,:).

'parAdjacencyMatrix' : N1xN2 indicator matrix where the (i,j) entry is 1 if line segments XY1(i,:) and XY2(j,:) are parallel.

'coincAdjacencyMatrix' : N1xN2 indicator matrix where the (i,j) entry is 1 if line segments XY1(i,:) and XY2(j,:) are coincident.

Version: 1.00, April 03, 2010 Version: 1.10, April 10, 2010 Author: U. Murat Erdem

## CHANGELOG:

Ver. 1.00: -Initial release.

Ver. 1.10:

- Changed the input parameters. Now the function accepts two sets of line segments. The intersection analysis is done between these sets and not in the same set.
- Changed and added fields of the output. Now the analysis provides more information about the intersections and line segments.
- Performance tweaks.

I opted not to call this 'curve intersect' because it would be misleading unless you accept that curves are pairwise linear constructs. I tried to put emphasis on speed by vectorizing the code as much as possible. There should still be enough room to optimize the code but I left those out for the sake of clarity. The math behind is given in: <http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/> If you really are interested in squeezing as much horse power as possible out of this code I would advise to remove the argument checks and tweak the creation of the OUT a little bit.

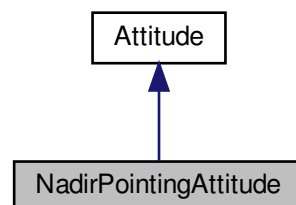
The documentation for this class was generated from the following files:

- src/[LinearAlgebra.hpp](#)
- src/[LinearAlgebra.cpp](#)

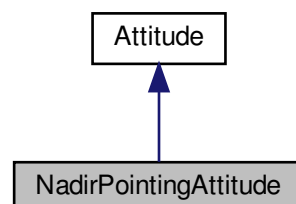
## 5.10 NadirPointingAttitude Class Reference

```
#include <NadirPointingAttitude.hpp>
```

Inheritance diagram for NadirPointingAttitude:



Collaboration diagram for NadirPointingAttitude:



## Public Member Functions

- [NadirPointingAttitude](#) ()  
*class construction/destruction*
- [NadirPointingAttitude](#) (const [NadirPointingAttitude](#) &copy)
- [NadirPointingAttitude](#) & operator= (const [NadirPointingAttitude](#) &copy)
- virtual [~NadirPointingAttitude](#) ()
- virtual Rmatrix33 [InertialToReference](#) (const Rvector6 &centralBodyState)  
*Converts the inertial-to-reference matrix.*

## Protected Attributes

- Rvector3 [centralBodyFixedPos](#)  
*Local class data for performance.*
- Rvector3 [centralBodyFixedVel](#)
- Rvector3 [zHat](#)
- Rvector3 [xHat](#)
- Rvector3 [yHat](#)
- Rmatrix33 [R\\_fixed\\_to\\_nadir](#)
- Rmatrix33 [R\\_fixed\\_to\\_nadir\\_transposed](#)

### 5.10.1 Detailed Description

Definition of the [NadirPointingAttitude](#) class. This class models the spacecraft attitude state.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 [NadirPointingAttitude\(\)](#) [1/2]

```
NadirPointingAttitude::NadirPointingAttitude ( )
```

class construction/destruction

Implementation of the base [NadirPointingAttitude](#) class Default constructor.

#### 5.10.2.2 [NadirPointingAttitude\(\)](#) [2/2]

```
NadirPointingAttitude::NadirPointingAttitude (
    const NadirPointingAttitude & copy )
```

Copy constructor.

#### Parameters

<i>copy</i>	the <a href="#">NadirPointingAttitude</a> object to copy
-------------	--

## 5.10.2.3 ~NadirPointingAttitude()

```
NadirPointingAttitude::~~NadirPointingAttitude ( ) [virtual]
```

Destructor.

## 5.10.3 Member Function Documentation

## 5.10.3.1 InertialToReference()

```
Rmatrix33 NadirPointingAttitude::InertialToReference (
    const Rvector6 & centralBodyState ) [virtual]
```

Converts the inertial-to-reference matrix.

**Todo** is this misnamed?

This method computes the matrix that converts from inertial to the reference frame, given the input central body state

## Parameters

<i>centralBodyState</i>	central body state
-------------------------	--------------------

## Returns

matrix from body to inertial

Reimplemented from [Attitude](#).

## 5.10.3.2 operator=()

```
NadirPointingAttitude & NadirPointingAttitude::operator= (
    const NadirPointingAttitude & copy )
```

The operator= for [NadirPointingAttitude](#).

## Parameters

<i>copy</i>	the <a href="#">NadirPointingAttitude</a> object to copy
-------------	--

## 5.10.4 Member Data Documentation

### 5.10.4.1 centralBodyFixedPos

`Rvector3 NadirPointingAttitude::centralBodyFixedPos` [protected]

Local class data for performance.

### 5.10.4.2 centralBodyFixedVel

`Rvector3 NadirPointingAttitude::centralBodyFixedVel` [protected]

### 5.10.4.3 R\_fixed\_to\_nadir

`Rmatrix33 NadirPointingAttitude::R_fixed_to_nadir` [protected]

### 5.10.4.4 R\_fixed\_to\_nadir\_transposed

`Rmatrix33 NadirPointingAttitude::R_fixed_to_nadir_transposed` [protected]

### 5.10.4.5 xHat

`Rvector3 NadirPointingAttitude::xHat` [protected]

### 5.10.4.6 yHat

`Rvector3 NadirPointingAttitude::yHat` [protected]



## 5.10.4.7 zHat

```
Rvector3 NadirPointingAttitude::zHat [protected]
```

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

- [src/NadirPointingAttitude.hpp](#)
- [src/NadirPointingAttitude.cpp](#)

## 5.11 OrbitState Class Reference

```
#include <OrbitState.hpp>
```

### Public Member Functions

- [OrbitState](#) ()  
*class construction/destruction*
- [OrbitState](#) (const [OrbitState](#) &copy)
- [OrbitState](#) & [operator=](#) (const [OrbitState](#) &copy)
- virtual [~OrbitState](#) ()
- virtual void [SetKeplerianState](#) (Real SMA, Real ECC, Real INC, Real RAAN, Real AOP, Real TA)  
*Set the Keplerian State elements.*
- virtual void [SetKeplerianVectorState](#) (const Rvector6 &kepl)  
*Set the Kerlerian state vector.*
- virtual void [SetCartesianState](#) (const Rvector6 &cart)  
*Set the Cartesian state.*
- virtual void [SetGravityParameter](#) (Real toGrav)  
*Set the gravity parameter.*
- virtual Rvector6 [GetKeplerianState](#) ()  
*Return the Keplerian state.*
- virtual Rvector6 [GetCartesianState](#) ()  
*Return the Cartesian state.*
- virtual [OrbitState](#) \* [Clone](#) () const  
*Clone the [OrbitState](#) object.*

### Protected Member Functions

- Rvector6 [ConvertKeplerianToCartesian](#) (Real a, Real e, Real i, Real Om, Real om, Real nu)  
*State conversion methods.*
- Rvector6 [ConvertCartesianToKeplerian](#) (const Rvector6 &cart)

### Protected Attributes

- Rvector6 [currentState](#)  
*Current state in cartesian format.*
- Real [mu](#)  
*Gravitational parameter for the central body.*

### 5.11.1 Detailed Description

Definition of the [OrbitState](#) class. This class computes and converts Cartesian and Keplerian states.

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 [OrbitState\(\)](#) [1/2]

```
OrbitState::OrbitState ( )
```

class construction/destruction

Implementation of the [OrbitState](#) class Default constructor.

#### 5.11.2.2 [OrbitState\(\)](#) [2/2]

```
OrbitState::OrbitState (
    const OrbitState & copy )
```

Copy constructor.

##### Parameters

<i>copy</i>	the <a href="#">OrbitState</a> object to copy
-------------	---

#### 5.11.2.3 [~OrbitState\(\)](#)

```
OrbitState::~~OrbitState ( ) [virtual]
```

Destructor.

### 5.11.3 Member Function Documentation

#### 5.11.3.1 [Clone\(\)](#)

```
OrbitState * OrbitState::Clone ( ) const [virtual]
```

Clone the [OrbitState](#) object.

This method returns a clone of the [OrbitState](#).

##### Returns

clone of the [OrbitState](#).

## 5.11.3.2 ConvertCartesianToKeplerian()

```
Rvector6 OrbitState::ConvertCartesianToKeplerian (
    const Rvector6 & cart ) [protected]
```

Converts the cartesian state to a keplerian state.

## Parameters

<i>cart</i>	cartesian state
-------------	-----------------

## Returns

the keplerian state as a 6-element vector

## 5.11.3.3 ConvertKeplerianToCartesian()

```
Rvector6 OrbitState::ConvertKeplerianToCartesian (
    Real a,
    Real e,
    Real i,
    Real Om,
    Real om,
    Real nu ) [protected]
```

State conversion methods.

Converts the keplerian state to a cartesian state.

## Parameters

<i>a</i>	semimajor axis
<i>e</i>	eccentricity
<i>i</i>	inclination
<i>Om</i>	right ascension of the ascending node
<i>om</i>	argument of periapsis
<i>nu</i>	true anomaly

## Returns

the cartesian state as a 6-element vector

## 5.11.3.4 GetCartesianState()

```
Rvector6 OrbitState::GetCartesianState ( ) [virtual]
```

Return the Cartesian state.

Returns the cartesian state as a 6-element vector

**Returns**

cartesian state

**5.11.3.5 GetKeplerianState()**

```
Rvector6 OrbitState::GetKeplerianState ( ) [virtual]
```

Return the Keplerian state.

Returns the keplerian state as a 6-element vector

**Returns**

keplerian state

**5.11.3.6 operator=()**

```
OrbitState & OrbitState::operator= (
    const OrbitState & copy )
```

The operator= for [OrbitState](#).

**Parameters**

<i>copy</i>	the <a href="#">OrbitState</a> object to copy
-------------	---

**5.11.3.7 SetCartesianState()**

```
void OrbitState::SetCartesianState (
    const Rvector6 & cart ) [virtual]
```

Set the Cartesian state.

Sets the cartesian state, as a 6-element vector.

**Parameters**

<i>cart</i>	cartesian state (units should be consistent with gravParam)
-------------	---

## 5.11.3.8 SetGravityParameter()

```
void OrbitState::SetGravityParameter (
    Real toGrav ) [virtual]
```

Set the gravity parameter.

Sets the gravity parameter.

## Parameters

<i>toGrav</i>	gravity parameter
---------------	-------------------

## 5.11.3.9 SetKeplerianState()

```
void OrbitState::SetKeplerianState (
    Real SMA,
    Real ECC,
    Real INC,
    Real RAAN,
    Real AOP,
    Real TA ) [virtual]
```

Set the Keplerian State elements.

Sets the keplerian state, element by element.

## Parameters

<i>SMA</i>	semimajor axis
<i>ECC</i>	eccentricity
<i>INC</i>	inclination
<i>RAAN</i>	right ascension of the ascending node
<i>AOP</i>	argument of periapsis
<i>TA</i>	true anomaly

## 5.11.3.10 SetKeplerianVectorState()

```
void OrbitState::SetKeplerianVectorState (
    const Rvector6 & kepl ) [virtual]
```

Set the Kerlerian state vector.

Sets the keplerian state, as a 6-element vector.

## Parameters

<i>kepl</i>	keplerian state
-------------	-----------------

## 5.11.4 Member Data Documentation

### 5.11.4.1 currentState

`Rvector6 OrbitState::currentState` [protected]

Current state in cartesian format.

### 5.11.4.2 mu

`Real OrbitState::mu` [protected]

Gravitational parameter for the central body.

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

- [src/OrbitState.hpp](#)
- [src/OrbitState.cpp](#)

## 5.12 PointGroup Class Reference

```
#include <PointGroup.hpp>
```

### Public Member Functions

- [PointGroup](#) ()  
*class construction/destruction*
- [PointGroup](#) (const [PointGroup](#) &copy)
- [PointGroup](#) & [operator=](#) (const [PointGroup](#) &copy)
- virtual [~PointGroup](#) ()
- virtual void [AddUserDefinedPoints](#) (const RealArray &lats, const RealArray &lons)  
*Add user defined points to the group.*
- virtual void [AddHelicalPointsByNumPoints](#) (Integer numGridPoints)  
*Compute and add the specified number of user-defined points.*
- virtual void [AddHelicalPointsByAngle](#) (Real angleBetweenPoints)
- virtual Rvector3 \* [GetPointPositionVector](#) (Integer idx)  
*Get point position for given index.*
- virtual void [GetLatAndLon](#) (Integer idx, Real &theLat, Real &theLon)  
*Get the latitude and longitude for the given index.*
- virtual Integer [GetNumPoints](#) ()  
*Get the number of points.*
- virtual void [GetLatLonVectors](#) (RealArray &lats, RealArray &lons)  
*Get the latitude and longitude vectors.*
- virtual void [SetLatLonBounds](#) (Real latUp, Real latLow, Real lonUp, Real lonLow)  
*Set the latitude and longitude bounds values.*

## Protected Member Functions

- bool [CheckHasPoints](#) ()  
*Protected methods for managing points.*
- void [AccumulatePoints](#) (Real lat1, Real lon1)
- void [ComputeTestPoints](#) (const std::string &modelName, Integer numGridPts)
- void [ComputeHelicalPoints](#) (Integer numReqPts)

## Protected Attributes

- RealArray [lat](#)  
*Latitude coordinates of grid points.*
- RealArray [lon](#)  
*Longitude coordinates of grid points.*
- std::vector< Rvector3 \* > [coords](#)
- Integer [numPoints](#)  
*num of points*
- Integer [numRequestedPoints](#)  
*Number of points requested in the point algorithm.*
- Real [latUpper](#)  
*Upper bound on allowable latitude  $-\pi/2 \leq \text{latUpper} \leq \pi/2$ .*
- Real [latLower](#)  
*Upper bound on allowable latitude  $-\pi/2 \leq \text{latLower} \leq \pi/2$ .*
- Real [lonUpper](#)  
*Upper bound on allowable longitude.*
- Real [lonLower](#)  
*Upper bound on allowable longitude.*

### 5.12.1 Detailed Description

Definition of the [PointGroup](#) class. This class stores latitudes, longitudes, and coordinates for points that are either set on input or computed in the class based on an input number or angle.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 [PointGroup\(\)](#) [1/2]

```
PointGroup::PointGroup ( )
```

class construction/destruction

Default constructor for the [PointGroup](#) class.

#### 5.12.2.2 [PointGroup\(\)](#) [2/2]

```
PointGroup::PointGroup (
    const PointGroup & copy )
```

Copy constructor for the [PointGroup](#) class.

## Parameters

<i>copy</i>	<a href="#">PointGroup</a> object to copy
-------------	---

5.12.2.3 `~PointGroup()`

```
PointGroup::~~PointGroup ( ) [virtual]
```

Destructor for the [PointGroup](#) class.

## 5.12.3 Member Function Documentation

5.12.3.1 `AccumulatePoints()`

```
void PointGroup::AccumulatePoints (
    Real lat1,
    Real lon1 ) [protected]
```

Adds a point with the specified latitude and longitude. The coordinates are computed from the input latitude and longitude.

## Parameters

<i>lat1</i>	latitude for the point to add
<i>lon1</i>	longitude for the point to add

5.12.3.2 `AddHelicalPointsByAngle()`

```
void PointGroup::AddHelicalPointsByAngle (
    Real angleBetweenPoints ) [virtual]
```

Compute and add points to the list of points, based on the input angle

Computes and adds points to the list of points, based on the input angle.

## Parameters

<i>angleBetweenPoints</i>	angle between points
---------------------------	----------------------



## 5.12.3.3 AddHelicalPointsByNumPoints()

```
void PointGroup::AddHelicalPointsByNumPoints (
    Integer numGridPoints ) [virtual]
```

Compute and add the specified number of user-defined points.

Computes and adds the specified number of user-defined points to the list of points.

## Parameters

<i>numGridPoints</i>	number of grid points to add
----------------------	------------------------------

## 5.12.3.4 AddUserDefinedPoints()

```
void PointGroup::AddUserDefinedPoints (
    const RealArray & lats,
    const RealArray & lons ) [virtual]
```

Add user defined points to the group.

Adds user-defined points to the list of points, given the input latitudes and longitudes.

## Parameters

<i>lats</i>	list of latitudes for the points to add
<i>lons</i>	list of longitudes for the points to add

## 5.12.3.5 CheckHasPoints()

```
bool PointGroup::CheckHasPoints ( ) [protected]
```

Protected methods for managing points.

Checks to see if there are any points set or computed

## Returns

true if there are points; false otherwise

## 5.12.3.6 ComputeHelicalPoints()

```
void PointGroup::ComputeHelicalPoints (
    Integer numReqPts ) [protected]
```

Computes the number of test points specified, using a model.

## Parameters

<i>numReqPts</i>	number of test points to compute and add
------------------	--

## 5.12.3.7 ComputeTestPoints()

```
void PointGroup::ComputeTestPoints (
    const std::string & modelName,
    Integer numGridPts ) [protected]
```

Computes the number of test points specified, for the model specified.

## Parameters

<i>modelName</i>	model for the points
<i>numGridPoints</i>	number of points to compute and add

## 5.12.3.8 GetLatAndLon()

```
void PointGroup::GetLatAndLon (
    Integer idx,
    Real & theLat,
    Real & theLon ) [virtual]
```

Get the latitude and longitude for the given index.

Returns the latitude and longitude of the specified point.

## Parameters

<i>idx</i>	[in] index of point whose latitude/longitude to return
<i>theLat</i>	[out] latitude of the specified point
<i>theLon</i>	[out] longitude of the specified point

## 5.12.3.9 GetLatLonVectors()

```
void PointGroup::GetLatLonVectors (
    RealArray & lats,
    RealArray & lons ) [virtual]
```

Get the latitude and longitude vectors.

Returns vectors of latitudes and longitudes for the points.

## Parameters

<i>lats</i>	[out] array of latitudes for the points
<i>lons</i>	[out] array of longitudes for the points

## 5.12.3.10 GetNumPoints()

```
Integer PointGroup::GetNumPoints ( ) [virtual]
```

Get the number of points.

Returns the number of points.

## Returns

the number of points

## 5.12.3.11 GetPointPositionVector()

```
Rvector3 * PointGroup::GetPointPositionVector (
    Integer idx ) [virtual]
```

Get point position for given index.

Returns the coordinates of the specified point.

## Parameters

<i>idx</i>	index of point whose coordinates to return
------------	--

## Returns

a 3-vector representing the coordinates of the specified point

## 5.12.3.12 operator=()

```
PointGroup & PointGroup::operator= (
    const PointGroup & copy )
```

operator= for the [PointGroup](#) class.

## Parameters

<i>copy</i>	<a href="#">PointGroup</a> object to copy
-------------	---

## 5.12.3.13 SetLatLonBounds()

```
void PointGroup::SetLatLonBounds (
    Real latUp,
    Real latLow,
    Real lonUp,
    Real lonLow ) [virtual]
```

Set the latitude and longitude bounds values.

Sets the latitude and longitude upper and lower bounds.

## Parameters

<i>latUp</i>	upper bound for latitude (radians)
<i>latLow</i>	lower bound for latitude (radians)
<i>lonUp</i>	upper bound for longitude (radians)
<i>lonLow</i>	lower bound for longitude (radians)

## 5.12.4 Member Data Documentation

## 5.12.4.1 coords

```
std::vector<Rvector3*> PointGroup::coords [protected]
```

## 5.12.4.2 lat

```
RealArray PointGroup::lat [protected]
```

Latitude coordinates of grid points.

## 5.12.4.3 latLower

```
Real PointGroup::latLower [protected]
```

Upper bound on allowable latitude  $-\pi/2 \leq \text{latLower} \leq \pi/2$ .

#### 5.12.4.4 latUpper

`Real PointGroup::latUpper [protected]`

Upper bound on allowable latitude  $-\pi/2 \leq \text{latUpper} \leq \pi/2$ .

#### 5.12.4.5 lon

`RealArray PointGroup::lon [protected]`

Longitude coordinates of grid points.

#### 5.12.4.6 lonLower

`Real PointGroup::lonLower [protected]`

Upper bound on allowable longitude.

#### 5.12.4.7 lonUpper

`Real PointGroup::lonUpper [protected]`

Upper bound on allowable longitude.

#### 5.12.4.8 numPoints

`Integer PointGroup::numPoints [protected]`

num of points

#### 5.12.4.9 numRequestedPoints

`Integer PointGroup::numRequestedPoints [protected]`

Number of points requested in the point algorithm.

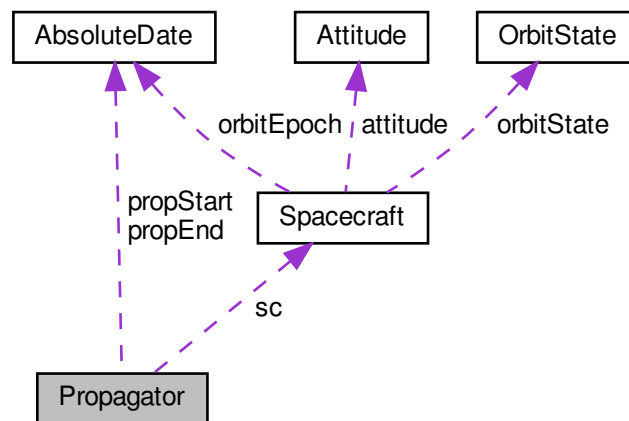
The documentation for this class was generated from the following files:

- [src/PointGroup.hpp](#)
- [src/PointGroup.cpp](#)

## 5.13 Propagator Class Reference

```
#include <Propagator.hpp>
```

Collaboration diagram for Propagator:



### Public Member Functions

- [Propagator](#) ([Spacecraft](#) \*sat)  
*class construction/destruction*
- [Propagator](#) (const [Propagator](#) &copy)
- [Propagator](#) & [operator=](#) (const [Propagator](#) &copy)
- virtual [~Propagator](#) ()
- virtual void [SetPhysicalConstants](#) (Real bodyMu, Real bodyJ2, Real bodyRadius)  
*Set the body physical constants on teh propagator.*
- virtual [Rvector6](#) [Propagate](#) (const [AbsoluteDate](#) &toDate)  
*Propagate the spacecraft.*
- virtual void [GetPropStartEnd](#) ([AbsoluteDate](#) &pStart, [AbsoluteDate](#) &pEnd)  
*Get the propagation start and end times.*
- void [SetApplyDrag](#) (bool applyDrag)  
*Set the flag indicating whether or not to apply drag.*
- bool [GetApplyDrag](#) ()  
*Get the flag indicating whether or not to apply drag.*

### Protected Member Functions

- void [SetOrbitState](#) ([OrbitState](#) \*orbState)  
*Set the orbit state.*
- Real [ComputePeriapsisAltitude](#) ([Rvector6](#) orbElem, Real julianDate)  
*Compute the periapsis altitude.*

- Rvector6 [PropagateOrbitalElements](#) (Real propDuration)  
*Propagate the orbital elements.*
- void [ComputeDragEffects](#) (Real sma, Real ecc, Real altitude, Real &deltaSMAperRev, Real &deltaECCperRev)  
*Compute the drag effects.*
- Real [MeanMotion](#) ()  
*Compute the orbital mean motion.*
- Real [SemiParameter](#) ()  
*Compute the semi parameter.*
- void [ComputeOrbitRates](#) ()  
*Compute the orbit rates.*
- void [ComputeMeanMotionRate](#) ()  
*Compute the mean motion rate.*
- void [ComputeArgumentOfPeriapsisRate](#) ()  
*Compute the argument of periapsis.*
- void [ComputeRightAscensionNodeRate](#) ()  
*Compute the right ascension of the ascending node rate.*

### Protected Attributes

- [Spacecraft](#) \* [sc](#)  
*The spacecraft to be propagated.*
- ExponentialAtmosphere \* [densityModel](#)  
*Density model used in computing effects of atmospheric drag.*
- Real [J2](#)  
*J2 term for [Earth](#).*
- Real [mu](#)  
*Gravitational parameter of the [Earth](#).*
- Real [eqRadius](#)  
*Equatorial radius of the [Earth](#).*
- bool [applyDrag](#)  
*Flag to turn on/off drag modeling.*
- Real [refJd](#)  
*Julian date of the reference orbital elements.*
- [AbsoluteDate](#) [propStart](#)  
*The epoch at which the propagation started.*
- [AbsoluteDate](#) [propEnd](#)
- Real [lastDragUpdateEpoch](#)  
*Epoch of last update to orbit to account for drag effects.*
- Real [orbitPeriod](#)  
*The orbital period.*
- Real [SMA](#)  
*Orbital semi-major axis.*
- Real [ECC](#)  
*Orbital eccentricity.*
- Real [INC](#)  
*Orbital inclination.*
- Real [RAAN](#)  
*Orbital right ascension of the ascending node.*
- Real [AOP](#)

- *Orbital sargument of periapsis.*  
Real [TA](#)
- *Orbital true anomaly.*  
Real [MA](#)
- *Orbital true anomaly.*  
Real [meanMotionRate](#)
- *The drift in mean motion caused by J2.*  
Real [argPeriapsisRate](#)
- *The drift in argument of periapsis caused by J2.*  
Real [rightAscensionNodeRate](#)
- *The drift in right ascention of the ascending node caused by J2.*  
Real [semiLatusRectum](#)
- *The orbital semi-latus rectum.*  
Real [meanMotion](#)
- *The orbital mean motion.*

### Static Protected Attributes

- static const Real [MU\\_FOR\\_EARTH](#) = 398600.4415  
*<static const>=""> Mu for the [Earth](#)*

### 5.13.1 Detailed Description

Definition of the the propagator class.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 Propagator() [1/2]

```
Propagator::Propagator (
    Spacecraft * sat )
```

class construction/destruction

Default constructor for [Propagator](#).

#### Parameters

<i>sat</i>	The spacecraft object
------------	-----------------------

#### 5.13.2.2 Propagator() [2/2]

```
Propagator::Propagator (
```



```
const Propagator & copy )
```

Copy constructor for [Propagator](#).

#### Parameters

<i>copy</i>	The propagator to copy
-------------	------------------------

#### 5.13.2.3 ~Propagator()

```
Propagator::~Propagator ( ) [virtual]
```

destructor for [Propagator](#).

### 5.13.3 Member Function Documentation

#### 5.13.3.1 ComputeArgumentOfPeriapsisRate()

```
void Propagator::ComputeArgumentOfPeriapsisRate ( ) [protected]
```

Compute the argument of periapsis.

Computes the argument of periapsis rate.

#### 5.13.3.2 ComputeDragEffects()

```
void Propagator::ComputeDragEffects (
    Real sma,
    Real ecc,
    Real altitude,
    Real & deltaSMAperRev,
    Real & deltaECCperRev ) [protected]
```

Compute the drag effects.

Computes the drag effects

#### Parameters

<i>sma</i>	[in] semimajor axis
<i>ecc</i>	[in] eccentricity
<i>altitude</i>	[in] altitude
<i>deltaSMAperRev</i>	[out] the delta SMA
<i>deltaECCperRev</i>	[out] the delta ECC

### 5.13.3.3 ComputeMeanMotionRate()

```
void Propagator::ComputeMeanMotionRate ( ) [protected]
```

Compute the mean motion rate.

Computes the mean motion rate.

### 5.13.3.4 ComputeOrbitRates()

```
void Propagator::ComputeOrbitRates ( ) [protected]
```

Compute the orbit rates.

Computes the orbit rates.

### 5.13.3.5 ComputePeriapsisAltitude()

```
Real Propagator::ComputePeriapsisAltitude (
    Rvector6 orbElem,
    Real julianDate ) [protected]
```

Compute the periapsis altitude.

Computes the periapsis altitude

#### Parameters

<i>orbElem</i>	input orbital elements
<i>julianDate</i>	the date at which to compute the periapsis altitude

#### Returns

the periapsis altitude

### 5.13.3.6 ComputeRightAscensionNodeRate()

```
void Propagator::ComputeRightAscensionNodeRate ( ) [protected]
```

Compute the right ascension of the ascending node rate.

Computes the right ascension node rate.

## 5.13.3.7 GetApplyDrag()

```
bool Propagator::GetApplyDrag ( )
```

Get the flag indicating whether or not to apply drag.

Returns the flag indicating whether or not to apply drag

**Returns**

apply drag flag

## 5.13.3.8 GetPropStartEnd()

```
void Propagator::GetPropStartEnd (
    AbsoluteDate & pStart,
    AbsoluteDate & pEnd ) [virtual]
```

Get the propagation start and end times.

Returns the propagator start and end times

**Parameters**

<i>pStart</i>	[out] start time
<i>pEnd</i>	[out] end time

## 5.13.3.9 MeanMotion()

```
Real Propagator::MeanMotion ( ) [protected]
```

Compute the orbital mean motion.

Computes the mean motion.

**Returns**

Mean Motion

## 5.13.3.10 operator=()

```
Propagator & Propagator::operator= (
    const Propagator & copy )
```

operator= for [Propagator](#).

**Parameters**

<i>copy</i>	The propagator to copy
-------------	------------------------

**5.13.3.11 Propagate()**

```
Rvector6 Propagator::Propagate (
    const AbsoluteDate & toDate ) [virtual]
```

Propagate the spacecraft.

Propagates to the input time.

**Parameters**

<i>toDate</i>	date to which to propagate
---------------	----------------------------

**5.13.3.12 PropagateOrbitalElements()**

```
Rvector6 Propagator::PropagateOrbitalElements (
    Real propDuration ) [protected]
```

Propagate the orbital elements.

Propagates the orbital elements for the specified duration.

**Parameters**

<i>propDuration</i>	duration over which to propagate
---------------------	----------------------------------

**Returns**

the propagated orbital elements

**5.13.3.13 SemiParameter()**

```
Real Propagator::SemiParameter ( ) [protected]
```

Compute the semi parameter.

Computes the semi parameter.

**Returns**

SemiParameter

#### 5.13.3.14 SetApplyDrag()

```
void Propagator::SetApplyDrag (
    bool flag )
```

Set the flag indicating whether or not to apply drag.

Sets the flag indicating whether or not to apply drag

##### Parameters

<i>flag</i>	apply drag flag
-------------	-----------------

#### 5.13.3.15 SetOrbitState()

```
void Propagator::SetOrbitState (
    OrbitState * orbState ) [protected]
```

Set the orbit state.

Sets the orbit state on the [Propagator](#).

##### Parameters

<i>orbState</i>	orbit state
-----------------	-------------

#### 5.13.3.16 SetPhysicalConstants()

```
void Propagator::SetPhysicalConstants (
    Real bodyMu,
    Real bodyJ2,
    Real bodyRadius ) [virtual]
```

Set the body physical constants on teh propagator.

Sets physical constant values for the [Propagator](#).

##### Parameters

<i>bodyMu</i>	gravitational parameter to use
<i>bodyJ2</i>	J2 term to use
<i>bodyRadius</i>	radius of the body

### 5.13.4 Member Data Documentation

#### 5.13.4.1 AOP

`Real Propagator::AOP [protected]`

Orbital sargument of periapsis.

#### 5.13.4.2 applyDrag

`bool Propagator::applyDrag [protected]`

Flag to turn on/off drag modeling.

#### 5.13.4.3 argPeriapsisRate

`Real Propagator::argPeriapsisRate [protected]`

The drift in argument of periapsis caused by J2.

#### 5.13.4.4 densityModel

`ExponentialAtmosphere* Propagator::densityModel [protected]`

Density model used in computing effects of atmospheric drag.

#### 5.13.4.5 ECC

`Real Propagator::ECC [protected]`

Orbital eccentricity.

#### 5.13.4.6 eqRadius

`Real Propagator::eqRadius [protected]`

Equatorial radius of the [Earth](#).

#### 5.13.4.7 INC

`Real Propagator::INC [protected]`

Orbital inclination.

#### 5.13.4.8 J2

`Real Propagator::J2 [protected]`

J2 term for [Earth](#).

#### 5.13.4.9 lastDragUpdateEpoch

`Real Propagator::lastDragUpdateEpoch [protected]`

Epoch of last update to orbit to account for drag effects.

#### 5.13.4.10 MA

`Real Propagator::MA [protected]`

Orbital true anomaly.

#### 5.13.4.11 meanMotion

`Real Propagator::meanMotion [protected]`

The orbital mean motion.

#### 5.13.4.12 meanMotionRate

`Real Propagator::meanMotionRate [protected]`

The drift in mean motion caused by J2.

#### 5.13.4.13 mu

`Real Propagator::mu [protected]`

Gravitational parameter of the [Earth](#).

#### 5.13.4.14 MU\_FOR\_EARTH

`const Real Propagator::MU_FOR_EARTH = 398600.4415 [static], [protected]`

`<static const>=""` Mu for the [Earth](#)

Implementation of the propagator class

#### 5.13.4.15 orbitPeriod

`Real Propagator::orbitPeriod [protected]`

The orbital period.

#### 5.13.4.16 propEnd

`AbsoluteDate Propagator::propEnd [protected]`

The epoch at which the propagation ended (so far, i.e. the last propagation time)

#### 5.13.4.17 propStart

`AbsoluteDate Propagator::propStart [protected]`

The epoch at which the propagation started.



#### 5.13.4.18 RAAN

`Real Propagator::RAAN [protected]`

Orbital right ascension of the ascending node.

#### 5.13.4.19 refJd

`Real Propagator::refJd [protected]`

Julian date of the reference orbital elements.

#### 5.13.4.20 rightAscensionNodeRate

`Real Propagator::rightAscensionNodeRate [protected]`

The drift in right ascension of the ascending node caused by J2.

#### 5.13.4.21 sc

`Spacecraft* Propagator::sc [protected]`

The spacecraft to be propagated.

**Todo** should this be an array of sc?

#### 5.13.4.22 semiLatusRectum

`Real Propagator::semiLatusRectum [protected]`

The orbital semi-latus rectum.

#### 5.13.4.23 SMA

`Real Propagator::SMA [protected]`

Orbital semi-major axis.

#### 5.13.4.24 TA

```
Real Propagator::TA [protected]
```

Orbital true anomaly.

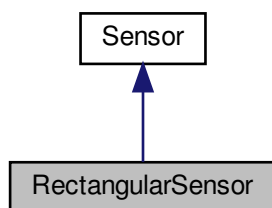
The documentation for this class was generated from the following files:

- [src/Propagator.hpp](#)
- [src/Propagator.cpp](#)

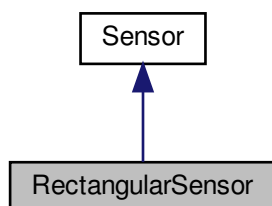
## 5.14 RectangularSensor Class Reference

```
#include <RectangularSensor.hpp>
```

Inheritance diagram for RectangularSensor:



Collaboration diagram for RectangularSensor:



## Public Member Functions

- [RectangularSensor](#) (Real angleWidthIn, Real angleHeightIn)  
*class construction/destruction*
- [RectangularSensor](#) (const [RectangularSensor](#) &copy)
- [RectangularSensor](#) & [operator=](#) (const [RectangularSensor](#) &copy)
- virtual [~RectangularSensor](#) ()
- virtual bool [CheckTargetVisibility](#) (Real viewConeAngle, Real viewClockAngle)
- virtual void [SetAngleWidth](#) (Real angleWidthIn)  
*Set/Get angle width.*
- virtual Real [GetAngleWidth](#) ()
- virtual void [SetAngleHeight](#) (Real angleHeightIn)  
*Set/Get angle height.*
- virtual Real [GetAngleHeight](#) ()

## Protected Attributes

- Real [angleWidth](#) = 0.0  
*Angle width.*
- Real [angleHeight](#) = 0.0  
*Angle height.*

## Additional Inherited Members

### 5.14.1 Detailed Description

Definition of the Conical [Sensor](#) class. This class models a conical sensor.

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 RectangularSensor() [1/2]

```
RectangularSensor::RectangularSensor (
    Real angleWidthIn,
    Real angleHeightIn )
```

class construction/destruction

Constructor

Parameters

<i>angleWidthIn</i>	angle width
<i>angle↔HeightIn</i>	angle height

#### 5.14.2.2 RectangularSensor() [2/2]

```
RectangularSensor::RectangularSensor (
    const RectangularSensor & copy )
```

Copy constructor

Parameters

<i>copy</i>	object to copy
-------------	----------------

#### 5.14.2.3 ~RectangularSensor()

```
RectangularSensor::~~RectangularSensor ( ) [virtual]
```

Destructor

### 5.14.3 Member Function Documentation

#### 5.14.3.1 CheckTargetVisibility()

```
bool RectangularSensor::CheckTargetVisibility (
    Real viewConeAngle,
    Real viewClockAngle ) [virtual]
```

Check the target visibility given the input cone and clock angles: determines whether or not the point is in the sensor FOV.

Determines whether or not the point is in the sensor FOV

Parameters

<i>viewConeAngle</i>	the view cone angle
<i>viewClockAngle</i>	the view clock angle <unused for="" this="" class="">="">

Returns

true if the point is in the sensor FOV; false otherwise

Implements [Sensor](#).

#### 5.14.3.2 GetAngleHeight()

```
Real RectangularSensor::GetAngleHeight ( ) [virtual]
```

Returns the angle height for the [RectangularSensor](#)

##### Returns

the angle height

#### 5.14.3.3 GetAngleWidth()

```
Real RectangularSensor::GetAngleWidth ( ) [virtual]
```

Returns the angle width for the [RectangularSensor](#)

##### Returns

the angle width

#### 5.14.3.4 operator=()

```
RectangularSensor & RectangularSensor::operator= (
    const RectangularSensor & copy )
```

The operator= for the [RectangularSensor](#)

##### Parameters

<i>copy</i>	object to copy
-------------	----------------

#### 5.14.3.5 SetAngleHeight()

```
void RectangularSensor::SetAngleHeight (
    Real angleHeightIn ) [virtual]
```

Set/Get angle height.

Sets the angle height for the [RectangularSensor](#)

## Parameters

<i>angle↔ HeightIn</i>	angle height
----------------------------	--------------

## 5.14.3.6 SetAngleWidth()

```
void RectangularSensor::SetAngleWidth (
    Real angleWidthIn ) [virtual]
```

Set/Get angle width.

Sets the angle width for the [RectangularSensor](#)

## Parameters

<i>angle↔ WidthIn</i>	angle width
---------------------------	-------------

## 5.14.4 Member Data Documentation

## 5.14.4.1 angleHeight

```
Real RectangularSensor::angleHeight = 0.0 [protected]
```

Angle height.

## 5.14.4.2 angleWidth

```
Real RectangularSensor::angleWidth = 0.0 [protected]
```

Angle width.

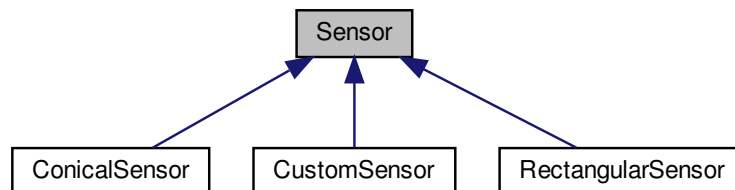
The documentation for this class was generated from the following files:

- [src/RectangularSensor.hpp](#)
- [src/RectangularSensor.cpp](#)

## 5.15 Sensor Class Reference

```
#include <Sensor.hpp>
```

Inheritance diagram for Sensor:



### Public Member Functions

- [Sensor](#) (Real angle1=0.0, Real angle2=0.0, Real angle3=0.0, Integer seq1=1, Integer seq2=2, Integer seq3=3)  
*class construction/destruction*
- [Sensor](#) (const [Sensor](#) &copy)
- [Sensor](#) & [operator=](#) (const [Sensor](#) &copy)
- virtual [~Sensor](#) ()
- virtual void [SetSensorBodyOffsetAngles](#) (Real angle1=0.0, Real angle2=0.0, Real angle3=0.0, Integer seq1=1, Integer seq2=2, Integer seq3=3)  
*Set the sensor-to-body offset angles.*
- virtual Rmatrix33 [GetBodyToSensorMatrix](#) (Real forTime)  
*Get the body-to-sensor matrix.*
- virtual bool [CheckTargetVisibility](#) (Real viewConeAngle, Real viewClockAngle=0.0)=0

### Protected Member Functions

- virtual bool [CheckTargetMaxExcursionAngle](#) (Real viewConeAngle)  
*Check the target maximum excursion angle.*
- virtual void [ComputeBodyToSensorMatrix](#) ()  
*Compute the body-to-sensor matrix.*
- void [ConeClocktoRADEC](#) (Real coneAngle, Real clockAngle, Real &RA, Real &dec)  
*Coordinate conversion utilities.*
- Rvector3 [RADECtoUnitVec](#) (Real RA, Real dec)
- void [UnitVecToStereographic](#) (const Rvector3 &u, Real &xCoord, Real &yCoord)
- void [ConeClockToStereographic](#) (Real coneAngle, Real clockAngle, Real &xCoord, Real &yCoord)
- void [ConeClockArraysToStereographic](#) (const Rvector &coneAngleVec, const Rvector &clockAngleVec, Rvector &xArray, Rvector &yArray)

## Protected Attributes

- Real [maxExcursionAngle](#)  
*The maximum excursion angle.*
- Real [offsetAngle1](#)  
*Offset angles.*
- Real [offsetAngle2](#)
- Real [offsetAngle3](#)
- Integer [eulerSeq1](#)  
*Euler sequence.*
- Integer [eulerSeq2](#)
- Integer [eulerSeq3](#)
- Rmatrix33 [R\\_SB](#)  
*The rotation matrix from the body frame to the sensor frame.*

### 5.15.1 Detailed Description

Definition of the base [Sensor](#) class. This class models a sensor.

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 [Sensor\(\)](#) [1/2]

```
Sensor::Sensor (
    Real angle1 = 0.0,
    Real angle2 = 0.0,
    Real angle3 = 0.0,
    Integer seq1 = 1,
    Integer seq2 = 2,
    Integer seq3 = 3 )
```

class construction/destruction

Implementation of the [Sensor](#) class Constructor

#### Parameters

<i>angle1</i>	The euler angle 1 (degrees)
<i>angle2</i>	The euler angle 2 (degrees)
<i>angle3</i>	The euler angle 3 (degrees)
<i>seq1</i>	Euler sequence 1
<i>seq2</i>	Euler sequence 2
<i>seq3</i>	Euler sequence 3



## 5.15.2.2 Sensor() [2/2]

```
Sensor::Sensor (
    const Sensor & copy )
```

Copy constructor

Parameters

<i>copy</i>	object to copy
-------------	----------------

## 5.15.2.3 ~Sensor()

```
Sensor::~Sensor ( ) [virtual]
```

Destructor

## 5.15.3 Member Function Documentation

## 5.15.3.1 CheckTargetMaxExcursionAngle()

```
bool Sensor::CheckTargetMaxExcursionAngle (
    Real viewConeAngle ) [protected], [virtual]
```

Check the target maximum excursion angle.

Checks if the target lies inside the max excursion angle

Parameters

<i>viewConeAngle</i>	the view cone angle
----------------------	---------------------

Returns

true if the point lies inside the max excursion angle; false otherwise

## 5.15.3.2 CheckTargetVisibility()

```
virtual bool Sensor::CheckTargetVisibility (
    Real viewConeAngle,
    Real viewClockAngle = 0.0 ) [pure virtual]
```

Check the target visibility given the input cone and clock angles: determines whether or not the point is in the sensor FOV.

## Parameters

<i>viewConeAngle</i>	cone angle
<i>viewClockAngle</i>	clock angle

## Returns

true if point is in the sensor FOV; false otherwise

## Note

This method is pure virtual and MUST be implemented in child classes

Implemented in [ConicalSensor](#), [CustomSensor](#), and [RectangularSensor](#).

## 5.15.3.3 ComputeBodyToSensorMatrix()

```
void Sensor::ComputeBodyToSensorMatrix ( ) [protected], [virtual]
```

Compute the body-to-sensor matrix.

Computes the rotation matrix from the body frame to the sensor frame.

## 5.15.3.4 ConeClockArraysToStereographic()

```
void Sensor::ConeClockArraysToStereographic (
    const Rvector & coneAngleVec,
    const Rvector & clockAngleVec,
    Rvector & xArray,
    Rvector & yArray ) [protected]
```

## 5.15.3.5 ConeClocktoRADEC()

```
void Sensor::ConeClocktoRADEC (
    Real coneAngle,
    Real clockAngle,
    Real & RA,
    Real & dec ) [protected]
```

Coordinate conversion utilities.

#### 5.15.3.6 ConeClockToStereographic()

```
void Sensor::ConeClockToStereographic (
    Real coneAngle,
    Real clockAngle,
    Real & xCoord,
    Real & yCoord ) [protected]
```

#### 5.15.3.7 GetBodyToSensorMatrix()

```
Rmatrix33 Sensor::GetBodyToSensorMatrix (
    Real forTime ) [virtual]
```

Get the body-to-sensor matrix.

Returns the rotation matrix from the body frame to the sensor frame.

##### Parameters

<i>forTime</i>	time at which to get the body-to-sensor matrix <unused>
----------------	---

#### 5.15.3.8 operator=()

```
Sensor & Sensor::operator= (
    const Sensor & copy )
```

The operator= for the [Sensor](#)

##### Parameters

<i>copy</i>	object to copy
-------------	----------------

#### 5.15.3.9 RADECToUnitVec()

```
Rvector3 Sensor::RADECToUnitVec (
    Real RA,
    Real dec ) [protected]
```

## 5.15.3.10 SetSensorBodyOffsetAngles()

```
void Sensor::SetSensorBodyOffsetAngles (
    Real angle1 = 0.0,
    Real angle2 = 0.0,
    Real angle3 = 0.0,
    Integer seq1 = 1,
    Integer seq2 = 2,
    Integer seq3 = 3 ) [virtual]
```

Set the sensor-to-body offset angles.

Sets the euler angles and sequence for the sensor

## Parameters

<i>angle1</i>	The euler angle 1 (degrees)
<i>angle2</i>	The euler angle 2 (degrees)
<i>angle3</i>	The euler angle 3 (degrees)
<i>seq1</i>	Euler sequence 1
<i>seq2</i>	Euler sequence 2
<i>seq3</i>	Euler sequence 3

## 5.15.3.11 UnitVecToStereographic()

```
void Sensor::UnitVecToStereographic (
    const Rvector3 & u,
    Real & xCoord,
    Real & yCoord ) [protected]
```

## 5.15.4 Member Data Documentation

## 5.15.4.1 eulerSeq1

```
Integer Sensor::eulerSeq1 [protected]
```

Euler sequence.

## 5.15.4.2 eulerSeq2

```
Integer Sensor::eulerSeq2 [protected]
```

#### 5.15.4.3 eulerSeq3

`Integer Sensor::eulerSeq3 [protected]`

#### 5.15.4.4 maxExcursionAngle

`Real Sensor::maxExcursionAngle [protected]`

The maximum excursion angle.

#### 5.15.4.5 offsetAngle1

`Real Sensor::offsetAngle1 [protected]`

Offset angles.

#### 5.15.4.6 offsetAngle2

`Real Sensor::offsetAngle2 [protected]`

#### 5.15.4.7 offsetAngle3

`Real Sensor::offsetAngle3 [protected]`

#### 5.15.4.8 R\_SB

`Rmatrix33 Sensor::R_SB [protected]`

The rotation matrix from the body frame to the sensor frame.

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

- [src/Sensor.hpp](#)
- [src/Sensor.cpp](#)

## 5.16 SensorElement Struct Reference

```
#include <CustomSensor.hpp>
```

### Public Attributes

- Real [value](#)
- Integer [index](#)

### 5.16.1 Detailed Description

data type and comparison method used in `Sort(Rvector,IntegerArray,bool)` to mimic Matlab sort of both values and indices of original array

### 5.16.2 Member Data Documentation

#### 5.16.2.1 index

```
Integer SensorElement::index
```

#### 5.16.2.2 value

```
Real SensorElement::value
```

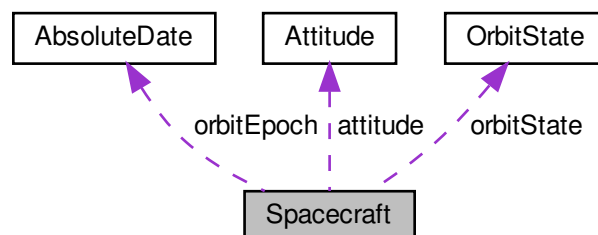
The documentation for this struct was generated from the following file:

- [src/CustomSensor.hpp](#)

## 5.17 Spacecraft Class Reference

```
#include <Spacecraft.hpp>
```

Collaboration diagram for Spacecraft:



## Public Member Functions

- [Spacecraft](#) ([AbsoluteDate](#) \*epoch, [OrbitState](#) \*state, [Attitude](#) \*att, [LagrangeInterpolator](#) \*interp, Real angle1=0.0, Real angle2=0.0, Real angle3=0.0, Integer seq1=1, Integer seq2=2, Integer seq3=3)  
*class construction/destruction*
- [Spacecraft](#) (const [Spacecraft](#) &copy)
- [Spacecraft](#) & [operator=](#) (const [Spacecraft](#) &copy)
- virtual [~Spacecraft](#) ()
- virtual [OrbitState](#) \* [GetOrbitState](#) ()  
*Get the orbit state.*
- virtual [AbsoluteDate](#) \* [GetOrbitEpoch](#) ()  
*Get the orbit epoch.*
- virtual Real [GetJulianDate](#) ()  
*Get the Julian date.*
- virtual [Rvector6](#) [GetCartesianState](#) ()  
*Get the current cartesian state.*
- virtual void [AddSensor](#) ([Sensor](#) \*sensor)  
*Add a sensor to the spacecraft.*
- virtual bool [HasSensors](#) ()  
*Does this spacecraft have sensors?*
- virtual void [SetDragArea](#) (Real area)  
*Set the drag area.*
- virtual void [SetDragCoefficient](#) (Real Cd)  
*Set the drag coefficient.*
- virtual void [SetTotalMass](#) (Real mass)  
*Set the total mass.*
- virtual void [SetAttitude](#) ([Attitude](#) \*att)  
*Set the attitude for the spacecraft.*
- virtual Real [GetDragArea](#) ()  
*Get the drag area.*
- virtual Real [GetDragCoefficient](#) ()
- virtual Real [GetTotalMass](#) ()  
*Get the total mass.*
- virtual [Rvector6](#) [GetCartesianStateAtEpoch](#) (const [AbsoluteDate](#) &atDate)  
*This method returns the interpolated MJ2000 Cartesian state.*
- virtual bool [CheckTargetVisibility](#) (Real targetConeAngle, Real targetClockAngle, Integer sensorNumber)
- virtual bool [CheckTargetVisibility](#) (const [Rvector6](#) &bodyFixedState, const [Rvector3](#) &satToTargetVec, Real atTime, Integer sensorNumber)
- virtual [Rmatrix33](#) [GetBodyFixedToInertial](#) (const [Rvector6](#) &bfState)  
*Get the body-fixed-to-inertial rotation matrix.*
- virtual bool [SetOrbitState](#) (const [AbsoluteDate](#) &t, const [Rvector6](#) &kepl)
- virtual void [SetBodyNadirOffsetAngles](#) (Real angle1=0.0, Real angle2=0.0, Real angle3=0.0, Integer seq1=1, Integer seq2=2, Integer seq3=3)  
*Set the body nadir offset angles for the spacecraft.*
- virtual bool [CanInterpolate](#) (Real atTime)  
*Can the orbit be interpolated - i.e. are there enough points, etc.?*
- virtual bool [TimeToInterpolate](#) (Real atTime, Real &midRange)
- virtual [Rvector6](#) [Interpolate](#) (Real toTime)  
*Interpolate the data to the input toTime.*



## Protected Member Functions

- virtual void [InertialToConeClock](#) (const Rvector3 &viewVec, Real &cone, Real &clock)  
*Convert inertial view vector to cone and clock angles.*
- virtual void [ComputeNadirToBodyMatrix](#) ()  
*Compute the nadir-to-body-matrix.*

## Protected Attributes

- Real [dragCoefficient](#)  
*Drag coefficient.*
- Real [dragArea](#)  
*Drag area in  $m^2$ .*
- Real [totalMass](#)  
*Total Mass in kg.*
- [OrbitState](#) \* [orbitState](#)  
*Orbit State.*
- [AbsoluteDate](#) \* [orbitEpoch](#)  
*Orbit Epoch.*
- Integer [numSensors](#)  
*Number of attached sensors.*
- std::vector< [Sensor](#) \* > [sensorList](#)  
*Vector of attached sensor objects.*
- [Attitude](#) \* [attitude](#)  
*Pointer to the [Attitude](#) object.*
- LagrangeInterpolator \* [interpolator](#)  
*The interpolator to use (for Hermite only, currently)*
- Real [offsetAngle1](#)  
*Offset angles.*
- Real [offsetAngle2](#)
- Real [offsetAngle3](#)
- Integer [eulerSeq1](#)  
*Euler sequence.*
- Integer [eulerSeq2](#)
- Integer [eulerSeq3](#)
- Rmatrix33 [R\\_BN](#)  
*The rotation matrix from the nadir frame to the body frame.*

### 5.17.1 Detailed Description

Definition of the [Spacecraft](#) class. This class contains data and methods for a simple [Spacecraft](#).

### 5.17.2 Constructor & Destructor Documentation

### 5.17.2.1 `Spacecraft()` [1/2]

```
Spacecraft::Spacecraft (
    AbsoluteDate * epoch,
    OrbitState * state,
    Attitude * att,
    LagrangeInterpolator * interp,
    Real angle1 = 0.0,
    Real angle2 = 0.0,
    Real angle3 = 0.0,
    Integer seq1 = 1,
    Integer seq2 = 2,
    Integer seq3 = 3 )
```

class construction/destruction

Implementation of the `Spacecraft` class. Default constructor for `Spacecraft`.

#### Parameters

<i>epoch</i>	The orbit epoch object
<i>state</i>	The orbit state object
<i>att</i>	The attitude object
<i>interp</i>	The LagrangeInterpolator
<i>angle1</i>	The euler angle 1 (degrees)
<i>angle2</i>	The euler angle 2 (degrees)
<i>angle3</i>	The euler angle 3 (degrees)
<i>seq1</i>	Euler sequence 1
<i>seq2</i>	Euler sequence 2
<i>seq3</i>	Euler sequence 3

### 5.17.2.2 `Spacecraft()` [2/2]

```
Spacecraft::Spacecraft (
    const Spacecraft & copy )
```

Copy constructor for `Spacecraft`.

#### Parameters

<i>copy</i>	The spacecraft of which to create a copy
-------------	--

### 5.17.2.3 `~Spacecraft()`

```
Spacecraft::~Spacecraft ( ) [virtual]
```

destructor for `Spacecraft`.

### 5.17.3 Member Function Documentation

#### 5.17.3.1 AddSensor()

```
void Spacecraft::AddSensor (
    Sensor * sensor ) [virtual]
```

Add a sensor to the spacecraft.

Adds the input sensor to the [Spacecraft](#)'s sensor list.

##### Parameters

<i>sensor</i>	<a href="#">Sensor</a> to add to the list
---------------	---

**Todo** • check for sensor already on list!!

#### 5.17.3.2 CanInterpolate()

```
bool Spacecraft::CanInterpolate (
    Real atTime ) [virtual]
```

Can the orbit be interpolated - i.e. are there enough points, etc.?

Can the orbit be interpolated (is it feasible given the number of points, etc.?)

##### Parameters

<i>atTime</i>	input time
<i>checkRange</i>	check the range to see if we need to interpolate

##### Returns

true if interpolation is feasible; false otherwise

#### 5.17.3.3 CheckTargetVisibility() [1/2]

```
bool Spacecraft::CheckTargetVisibility (
    Real targetConeAngle,
    Real targetClockAngle,
    Integer sensorNumber ) [virtual]
```

Check the target visibility given the input cone and clock angles for the input sensor number

Returns a flag indicating whether or not the point is within the

## Parameters

<i>targetConeAngle</i>	the cone angle
<i>targetClockAngle</i>	the clock angle
<i>sensorNumber</i>	sensor for which to check target visibility

## Returns

true if point is visible, false otherwise

## 5.17.3.4 CheckTargetVisibility() [2/2]

```
bool Spacecraft::CheckTargetVisibility (
    const Rvector6 & bodyFixedState,
    const Rvector3 & satToTargetVec,
    Real atTime,
    Integer sensorNumber ) [virtual]
```

Check the target visibility given the input body fixed state and spacecraft-to-target vector, at the input time, for the input sensor number

Returns a flag indicating whether or not the point is within the visible to the sensor at the given time, given the satToTargetVec.

## Parameters

<i>bodyFixedState</i>	input body fixed state
<i>satToTargetVec</i>	spacecraft-to-target vector
<i>atTime</i>	time for which to check the target visibility
<i>sensorNumber</i>	sensor for which to check target visibility

## Returns

true if point is visible, false otherwise

## 5.17.3.5 ComputeNadirToBodyMatrix()

```
void Spacecraft::ComputeNadirToBodyMatrix ( ) [protected], [virtual]
```

Compute the nadir-to-body-matrix.

Computes the rotation matrix from the body frame to the sensor frame.

#### 5.17.3.6 GetBodyFixedToInertial()

```
Rmatrix33 Spacecraft::GetBodyFixedToInertial (
    const Rvector6 & bfState ) [virtual]
```

Get the body-fixed-to-inertial rotation matrix.

Returns the bodyfixed-to-inertial matrix, given the input state

##### Parameters

<i>bfState</i>	body-fixed state
----------------	------------------

##### Returns

bodyfixed-to-inertial matrix

#### 5.17.3.7 GetCartesianState()

```
Rvector6 Spacecraft::GetCartesianState ( ) [virtual]
```

Get the current cartesian state.

Returns the [Spacecraft](#)'s cartesian state.

##### Returns

[Spacecraft](#)'s cartesian state

#### 5.17.3.8 GetCartesianStateAtEpoch()

```
Rvector6 Spacecraft::GetCartesianStateAtEpoch (
    const AbsoluteDate & atDate ) [virtual]
```

This method returns the interpolated MJ2000 Cartesian state.

Gets the [Spacecraft](#)'s cartesian state ([Earth](#) MJ2000Eq) at the input time

##### Parameters

<i>atDate</i>	the date for which to get the cartesian state
---------------	---

##### Returns

state

### 5.17.3.9 GetDragArea()

```
Real Spacecraft::GetDragArea ( ) [virtual]
```

Get the drag area.

Gets the [Spacecraft](#)'s drag area.

#### Returns

the drag area in  $m^2$

### 5.17.3.10 GetDragCoefficient()

```
Real Spacecraft::GetDragCoefficient ( ) [virtual]
```

Gets the [Spacecraft](#)'s drag coefficient.

#### Returns

the drag coefficient

### 5.17.3.11 GetJulianDate()

```
Real Spacecraft::GetJulianDate ( ) [virtual]
```

Get the Julian date.

Returns the [Spacecraft](#)'s Julian Date.

#### Returns

[Spacecraft](#)'s JulianDate

### 5.17.3.12 GetOrbitEpoch()

```
AbsoluteDate * Spacecraft::GetOrbitEpoch ( ) [virtual]
```

Get the orbit epoch.

Returns a pointer to the [Spacecraft](#)'s [AbsoluteDate](#) object.

#### Returns

pointer to the spacecraft's [AbsoluteDate](#)

#### 5.17.3.13 GetOrbitState()

```
OrbitState * Spacecraft::GetOrbitState ( ) [virtual]
```

Get the orbit state.

Returns a pointer to the [Spacecraft's OrbitState](#) object.

##### Returns

pointer to the spacecraft's [OrbitState](#)

#### 5.17.3.14 GetTotalMass()

```
Real Spacecraft::GetTotalMass ( ) [virtual]
```

Get the total mass.

Gets the [Spacecraft's](#) total mass.

##### Returns

the total mass

#### 5.17.3.15 HasSensors()

```
bool Spacecraft::HasSensors ( ) [virtual]
```

Does this spacecraft have sensors?

Returns a flag indicating whether or not the spacecraft has sensors.

##### Returns

flag indicating whether or not the spacecraft has sensors.

#### 5.17.3.16 InertialToConeClock()

```
void Spacecraft::InertialToConeClock (
    const Rvector3 & viewVec,
    Real & cone,
    Real & clock ) [protected], [virtual]
```

Convert inertial view vector to cone and clock angles.

**Todo** • do we need to buffer states here as well??

Computes the rotation matrix from the body frame to the sensor frame.



## Parameters

<i>viewVec</i>	[in] input view vector
<i>cone</i>	[out] cone angle
<i>clock</i>	[out] clock angle

## 5.17.3.17 Interpolate()

```
Rvector6 Spacecraft::Interpolate (
    Real toTime ) [virtual]
```

Interpolate the data to the input toTime.

Interpolate the orbit data at the input time

## Parameters

<i>atTime</i>	input time
---------------	------------

## Returns

interpolated state data

## 5.17.3.18 operator=()

```
Spacecraft & Spacecraft::operator= (
    const Spacecraft & copy )
```

operator= for [Spacecraft](#).

## Parameters

<i>copy</i>	The spacecraft whose values to copy
-------------	-------------------------------------

## 5.17.3.19 SetAttitude()

```
void Spacecraft::SetAttitude (
    Attitude * att ) [virtual]
```

Set the attitude for the spacecraft.

Sets the [Spacecraft](#)'s attitude object

## Parameters

<i>att</i>	the attitude object
------------	---------------------

## 5.17.3.20 SetBodyNadirOffsetAngles()

```
void Spacecraft::SetBodyNadirOffsetAngles (
    Real angle1 = 0.0,
    Real angle2 = 0.0,
    Real angle3 = 0.0,
    Integer seq1 = 1,
    Integer seq2 = 2,
    Integer seq3 = 3 ) [virtual]
```

Set the body nadir offset angles for the spacecraft.

Sets the body nadir offset angles

## Parameters

<i>angle1</i>	euler angle 1 (degrees)
<i>angle2</i>	euler angle 2 (degrees)
<i>angle3</i>	euler angle 3 (degrees)
<i>seq1</i>	euler msequence 1
<i>seq2</i>	euler msequence 2
<i>seq3</i>	euler msequence 3

## 5.17.3.21 SetDragArea()

```
void Spacecraft::SetDragArea (
    Real area ) [virtual]
```

Set the drag area.

Sets the [Spacecraft](#)'s drag area.

## Parameters

<i>the</i>	drag area in m <sup>2</sup>
------------	-----------------------------

## 5.17.3.22 SetDragCoefficient()

```
void Spacecraft::SetDragCoefficient (
```

```
Real Cd ) [virtual]
```

Set the drag coefficient.

Sets the [Spacecraft](#)'s drag coefficient.

#### Parameters

<i>the</i>	drag coefficient
------------	------------------

#### 5.17.3.23 SetOrbitState()

```
bool Spacecraft::SetOrbitState (
    const AbsoluteDate & t,
    const Rvector6 & kepl ) [virtual]
```

Add an orbit state (Keplerian elements) for the spacecraft at the input time t

Sets the orbit state on the [Spacecraft](#)

#### Parameters

<i>t</i>	input time
<i>kepl</i>	input keplerian elements

#### Returns

true if set; false otherwise

#### 5.17.3.24 SetTotalMass()

```
void Spacecraft::SetTotalMass (
    Real mass ) [virtual]
```

Set the total mass.

Sets the [Spacecraft](#)'s total mass.

#### Parameters

<i>the</i>	total mass
------------	------------

### 5.17.3.25 TimeToInterpolate()

```
bool Spacecraft::TimeToInterpolate (
    Real atTime,
    Real & midRange ) [virtual]
```

Is it time to interpolate? i.e. are there enough points? if so, what is the midpoint of the independent variable lower/upper range

Get the midpoint time to interpolate to, if

#### Parameters

<i>atTime</i>	[in] input time
<i>midRange</i>	[out] middle of the interpolation range size

#### Returns

true if interpolation should be performed at the input time; false otherwise

## 5.17.4 Member Data Documentation

### 5.17.4.1 attitude

```
Attitude* Spacecraft::attitude [protected]
```

Pointer to the [Attitude](#) object.

### 5.17.4.2 dragArea

```
Real Spacecraft::dragArea [protected]
```

Drag area in m<sup>2</sup>.

### 5.17.4.3 dragCoefficient

```
Real Spacecraft::dragCoefficient [protected]
```

Drag coefficient.

#### 5.17.4.4 eulerSeq1

Integer Spacecraft::eulerSeq1 [protected]

Euler sequence.

#### 5.17.4.5 eulerSeq2

Integer Spacecraft::eulerSeq2 [protected]

#### 5.17.4.6 eulerSeq3

Integer Spacecraft::eulerSeq3 [protected]

#### 5.17.4.7 interpolator

LagrangeInterpolator\* Spacecraft::interpolator [protected]

The interpolator to use (for Hermite only, currently)

#### 5.17.4.8 numSensors

Integer Spacecraft::numSensors [protected]

Number of attached sensors.

#### 5.17.4.9 offsetAngle1

Real Spacecraft::offsetAngle1 [protected]

Offset angles.

#### 5.17.4.10 offsetAngle2

Real Spacecraft::offsetAngle2 [protected]

#### 5.17.4.11 offsetAngle3

`Real Spacecraft::offsetAngle3 [protected]`

#### 5.17.4.12 orbitEpoch

`AbsoluteDate* Spacecraft::orbitEpoch [protected]`

Orbit Epoch.

#### 5.17.4.13 orbitState

`OrbitState* Spacecraft::orbitState [protected]`

Orbit State.

#### 5.17.4.14 R\_BN

`Rmatrix33 Spacecraft::R_BN [protected]`

The rotation matrix from the nadir frame to the body frame.

#### 5.17.4.15 sensorList

`std::vector<Sensor*> Spacecraft::sensorList [protected]`

Vector of attached sensor objects.

#### 5.17.4.16 totalMass

`Real Spacecraft::totalMass [protected]`

Total Mass in kg.

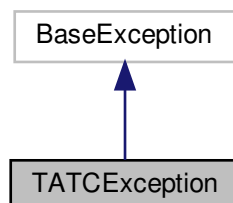
The documentation for this class was generated from the following files:

- [src/Spacecraft.hpp](#)
- [src/Spacecraft.cpp](#)

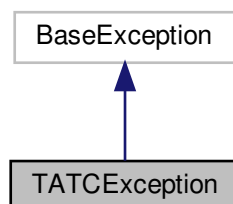
## 5.18 TATCException Class Reference

```
#include <TATCException.hpp>
```

Inheritance diagram for TATCException:



Collaboration diagram for TATCException:



### Public Member Functions

- [TATCException](#) (const std::string &details="")  
*class construction/destruction*
- [TATCException](#) (const [TATCException](#) &be)

### 5.18.1 Detailed Description

Exception class used by the TAT-C code to generate visibilty reports. Exception class used to report issues with event location.

### 5.18.2 Constructor & Destructor Documentation

### 5.18.2.1 TATCException() [1/2]

```
TATCException::TATCException (
    const std::string & details = "" )
```

class construction/destruction

Exception class used by the TAT-C code to generate visibility reports. Constructs [TATCException](#) instance (default constructor).

#### Parameters

<i>details</i>	A message providing the details of the exception.
----------------	---

### 5.18.2.2 TATCException() [2/2]

```
TATCException::TATCException (
    const TATCException & be )
```

Constructs [TATCException](#) instance (copy constructor).

#### Parameters

<i>be</i>	The instance that is copied.
-----------	------------------------------

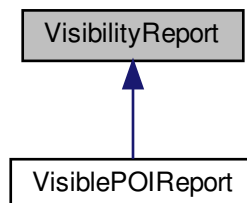
The documentation for this class was generated from the following files:

- [src/TATCException.hpp](#)
- [src/TATCException.cpp](#)

## 5.19 VisibilityReport Class Reference

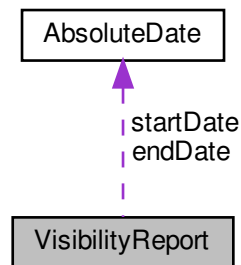
```
#include <VisibilityReport.hpp>
```

Inheritance diagram for VisibilityReport:





Collaboration diagram for VisibilityReport:



### Public Member Functions

- [VisibilityReport](#) ()  
*class construction/destruction*
- [VisibilityReport](#) (const [VisibilityReport](#) &copy)
- [VisibilityReport](#) & [operator=](#) (const [VisibilityReport](#) &copy)
- virtual [~VisibilityReport](#) ()
- virtual void [SetStartDate](#) (const [AbsoluteDate](#) &toDate)  
*Set the start date.*
- virtual void [SetEndDate](#) (const [AbsoluteDate](#) &toDate)  
*Set the end date.*
- virtual const [AbsoluteDate](#) & [GetStartDate](#) ()  
*Get the start date.*
- virtual const [AbsoluteDate](#) & [GetEndDate](#) ()  
*Get the end date.*

### Protected Attributes

- [AbsoluteDate](#) [startDate](#)  
*Start date of the interval event.*
- [AbsoluteDate](#) [endDate](#)  
*End date of the interval event.*

#### 5.19.1 Detailed Description

Definition of the visibility report base class. This class reports and stores visibility data.

#### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 VisibilityReport() [1/2]

```
VisibilityReport::VisibilityReport ( )
```

class construction/destruction

Implementation of the visibility report base class. Constructs [VisibilityReport](#) instance (default constructor).

#### 5.19.2.2 VisibilityReport() [2/2]

```
VisibilityReport::VisibilityReport (
    const VisibilityReport & copy )
```

Constructs [VisibilityReport](#) instance (copy constructor).

##### Parameters

<i>be</i>	The instance that is copied.
-----------	------------------------------

#### 5.19.2.3 ~VisibilityReport()

```
VisibilityReport::~~VisibilityReport ( ) [virtual]
```

Destructs [VisibilityReport](#) instance

### 5.19.3 Member Function Documentation

#### 5.19.3.1 GetEndDate()

```
const AbsoluteDate & VisibilityReport::GetEndDate ( ) [virtual]
```

Get the end date.

Returns the end date for the report.

##### Returns

The end time for the report.

#### 5.19.3.2 GetStartDate()

```
const AbsoluteDate & VisibilityReport::GetStartDate ( ) [virtual]
```

Get the start date.

Returns the start date for the report.

##### Returns

The start time for the report.

#### 5.19.3.3 operator=()

```
VisibilityReport & VisibilityReport::operator= (
    const VisibilityReport & copy )
```

VisibilityReport operator=.

##### Parameters

<i>be</i>	The instance that is copied.
-----------	------------------------------

#### 5.19.3.4 SetEndDate()

```
void VisibilityReport::SetEndDate (
    const AbsoluteDate & toDate ) [virtual]
```

Set the end date.

Sets the end date for the report.

##### Parameters

<i>toDate</i>	The end time for the report.
---------------	------------------------------

#### 5.19.3.5 SetStartDate()

```
void VisibilityReport::SetStartDate (
    const AbsoluteDate & toDate ) [virtual]
```

Set the start date.

Sets the start date for the report.

## Parameters

<i>toDate</i>	The start time for the report.
---------------	--------------------------------

## 5.19.4 Member Data Documentation

### 5.19.4.1 endDate

`AbsoluteDate` `VisibilityReport::endDate` [protected]

End date of the interval event.

### 5.19.4.2 startDate

`AbsoluteDate` `VisibilityReport::startDate` [protected]

Start date of the interval event.

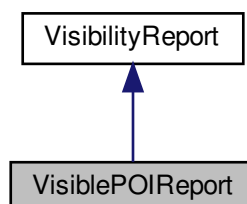
The documentation for this class was generated from the following files:

- [src/VisibilityReport.hpp](#)
- [src/VisibilityReport.cpp](#)

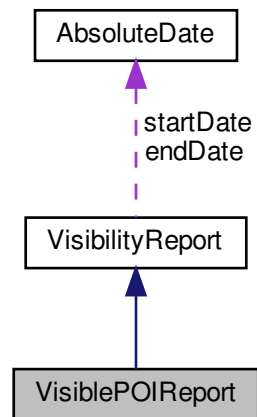
## 5.20 VisiblePOIReport Class Reference

```
#include <VisiblePOIReport.hpp>
```

Inheritance diagram for VisiblePOIReport:



Collaboration diagram for VisiblePOIReport:



## Public Member Functions

- [VisiblePOIReport](#) ()  
*class construction/destruction*
- [VisiblePOIReport](#) (const [VisiblePOIReport](#) &copy)
- [VisiblePOIReport](#) & operator= (const [VisiblePOIReport](#) &copy)
- virtual [~VisiblePOIReport](#) ()
- virtual void [SetPOIIndex](#) (Integer toldx)  
*Set/Get the POI index.*
- virtual Integer [GetPOIIndex](#) ()
- virtual void [SetObsZenith](#) (Real obsZenithIn)  
*Set/Get the observation zenith angle.*
- virtual Real [GetObsZenith](#) ()
- virtual void [SetObsAzimuth](#) (Real obsZenithIn)  
*Set/Get the observation azimuth.*
- virtual Real [GetObsAzimuth](#) ()
- virtual void [SetObsRange](#) (Real obsRangeIn)  
*Set/Get the observation range.*
- virtual Real [GetObsRange](#) ()
- virtual void [SetSunZenith](#) (Real sunZenithIn)  
*Set/Get the Sun zenith angle.*
- virtual Real [GetSunZenith](#) ()
- virtual void [SetSunAzimuth](#) (Real obsZenithIn)  
*Set/Get the Sun azimuth.*
- virtual Real [GetSunAzimuth](#) ()

## Protected Attributes

- Integer [poiIndex](#)  
*Index of point of interest.*
- Real [obsZenith](#)  
*The observation zenith angle.*
- Real [obsAzimuth](#)  
*The observation azimuth angle.*
- Real [obsRange](#)  
*The observation range angle.*
- Real [sunZenith](#)  
*The Sun zenith angle.*
- Real [sunAzimuth](#)  
*The Sun azimuth angle.*

### 5.20.1 Detailed Description

Definition of the visibility POI report class. This class is the container for data on POI interval events.

### 5.20.2 Constructor & Destructor Documentation

#### 5.20.2.1 VisiblePOIReport() [1/2]

```
VisiblePOIReport::VisiblePOIReport ( )
```

class construction/destruction

Implementation of the visibility POI report class. Default constructor for [VisiblePOIReport](#).

#### 5.20.2.2 VisiblePOIReport() [2/2]

```
VisiblePOIReport::VisiblePOIReport (
    const VisiblePOIReport & copy )
```

Copy constructor for [VisiblePOIReport](#).

#### 5.20.2.3 ~VisiblePOIReport()

```
VisiblePOIReport::~~VisiblePOIReport ( ) [virtual]
```

Destructor for [VisiblePOIReport](#).

### 5.20.3 Member Function Documentation

### 5.20.3.1 GetObsAzimuth()

```
Real VisiblePOIReport::GetObsAzimuth ( ) [virtual]
```

Returns the Azimuth angle of observation w/r/t target

#### Returns

Azimuth angle of observation w/r/t target

### 5.20.3.2 GetObsRange()

```
Real VisiblePOIReport::GetObsRange ( ) [virtual]
```

Returns the Range angle of observation w/r/t target

#### Returns

Range angle of observation w/r/t target

### 5.20.3.3 GetObsZenith()

```
Real VisiblePOIReport::GetObsZenith ( ) [virtual]
```

Returns the zenith angle of observation w/r/t target

#### Returns

zenith angle of observation w/r/t target

### 5.20.3.4 GetPOIIndex()

```
Integer VisiblePOIReport::GetPOIIndex ( ) [virtual]
```

Returns the POI Index for the report.

#### Returns

<toIdx> Index for the report

### 5.20.3.5 GetSunAzimuth()

```
Real VisiblePOIReport::GetSunAzimuth ( ) [virtual]
```

Returns the zenith angle of sun w/r/t target

#### Returns

zenith angle of sun w/r/t target

### 5.20.3.6 GetSunZenith()

```
Real VisiblePOIReport::GetSunZenith ( ) [virtual]
```

Returns the zenith angle of sun w/r/t target

#### Returns

zenith angle of sun w/r/t target

### 5.20.3.7 operator=()

```
VisiblePOIReport & VisiblePOIReport::operator= (
    const VisiblePOIReport & copy )
```

operator= for [VisiblePOIReport](#).

### 5.20.3.8 SetObsAzimuth()

```
void VisiblePOIReport::SetObsAzimuth (
    Real obsAzimuthIn ) [virtual]
```

Set/Get the observation azimuth.

Sets the Azimuth angle of observation w/r/t target

#### Parameters

<i>obs</i> ↔ <i>AzimuthIn</i>	The observation Azimuth angle
----------------------------------	-------------------------------



### 5.20.3.9 SetObsRange()

```
void VisiblePOIReport::SetObsRange (
    Real obsRangeIn ) [virtual]
```

Set/Get the observation range.

Sets the Range angle of observation w/r/t target

#### Parameters

<i>obs↔ RangeIn</i>	The observation Range angle
-------------------------	-----------------------------

### 5.20.3.10 SetObsZenith()

```
void VisiblePOIReport::SetObsZenith (
    Real obsZenithIn ) [virtual]
```

Set/Get the observation zenith angle.

Sets the zenith angle of observation w/r/t target

#### Parameters

<i>obs↔ ZenithIn</i>	The observation zenith angle
--------------------------	------------------------------

### 5.20.3.11 SetPOIIndex()

```
void VisiblePOIReport::SetPOIIndex (
    Integer toIdx ) [virtual]
```

Set/Get the POI index.

Sets the POI Index for the report.

#### Parameters

< <i>toIdx</i> >	Index for the report
------------------	----------------------

#### 5.20.3.12 SetSunAzimuth()

```
void VisiblePOIReport::SetSunAzimuth (
    Real sunAzimuthIn ) [virtual]
```

Set/Get the Sun azimuth.

Sets the zenith angle of sun w/r/t target

##### Parameters

<i>sun↔ AzimuthIn</i>	The sun zenith angle
---------------------------	----------------------

#### 5.20.3.13 SetSunZenith()

```
void VisiblePOIReport::SetSunZenith (
    Real sunZenithIn ) [virtual]
```

Set/Get the Sun zenith angle.

Sets the zenith angle of sun w/r/t target

##### Parameters

<i>sun↔ ZenithIn</i>	The sun zenith angle
--------------------------	----------------------

### 5.20.4 Member Data Documentation

#### 5.20.4.1 obsAzimuth

```
Real VisiblePOIReport::obsAzimuth [protected]
```

The observation azimuth angle.

#### 5.20.4.2 obsRange

```
Real VisiblePOIReport::obsRange [protected]
```

The observation range angle.

#### 5.20.4.3 obsZenith

Real VisiblePOIReport::obsZenith [protected]

The observation zenith angle.

#### 5.20.4.4 poiIndex

Integer VisiblePOIReport::poiIndex [protected]

Index of point of interest.

#### 5.20.4.5 sunAzimuth

Real VisiblePOIReport::sunAzimuth [protected]

The Sun azimuth angle.

#### 5.20.4.6 sunZenith

Real VisiblePOIReport::sunZenith [protected]

The Sun zenith angle.

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

- [src/VisiblePOIReport.hpp](#)
- [src/VisiblePOIReport.cpp](#)

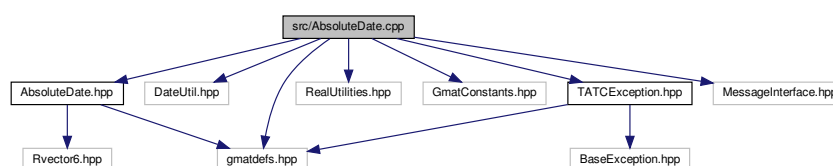


## Chapter 6

# File Documentation

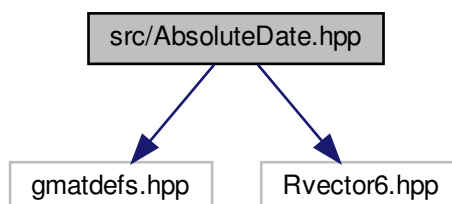
### 6.1 src/AbsoluteDate.cpp File Reference

```
#include "gmatdefs.hpp"  
#include "DateUtil.hpp"  
#include "AbsoluteDate.hpp"  
#include "RealUtilities.hpp"  
#include "GmatConstants.hpp"  
#include "TATCException.hpp"  
#include "MessageInterface.hpp"  
Include dependency graph for AbsoluteDate.cpp:
```

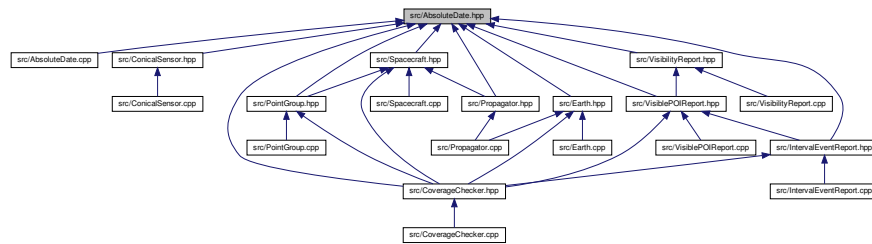


### 6.2 src/AbsoluteDate.hpp File Reference

```
#include "gmatdefs.hpp"  
#include "Rvector6.hpp"  
Include dependency graph for AbsoluteDate.hpp:
```



This graph shows which files directly or indirectly include this file:



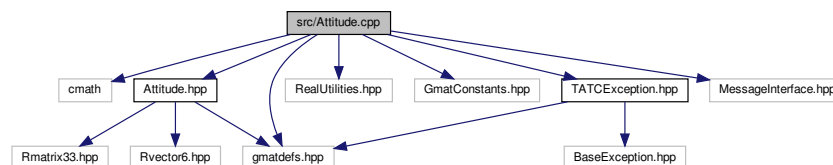
## Classes

- class [AbsoluteDate](#)

## 6.3 src/Attitude.cpp File Reference

```
#include <cmath>
#include "gmatdefs.hpp"
#include "Attitude.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "TATCException.hpp"
#include "MessageInterface.hpp"
```

Include dependency graph for Attitude.cpp:

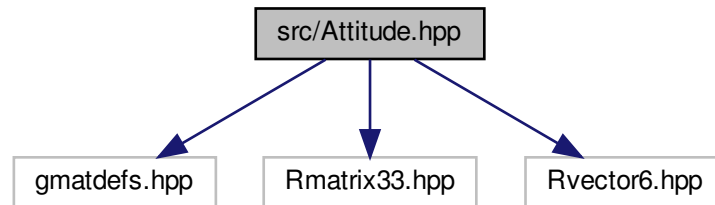


## 6.4 src/Attitude.hpp File Reference

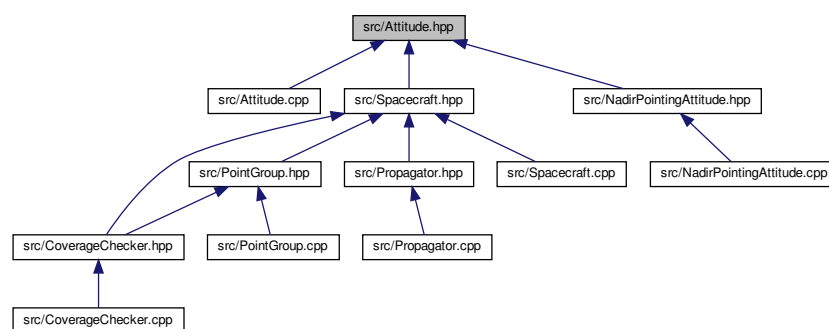
```
#include "gmatdefs.hpp"
#include "Rmatrix33.hpp"
```

```
#include "Rvector6.hpp"
```

Include dependency graph for Attitude.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

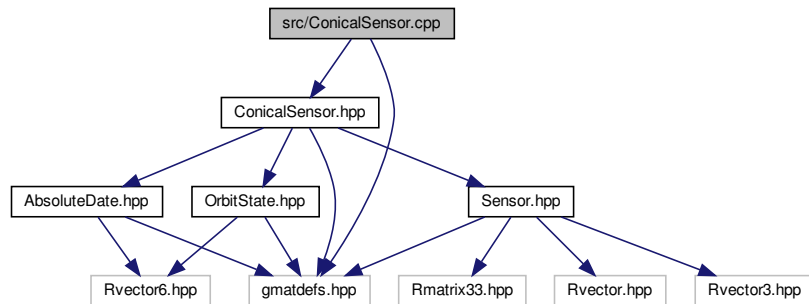
- class [Attitude](#)

## 6.5 src/ConicalSensor.cpp File Reference

```
#include "gmatdefs.hpp"
```

```
#include "ConicalSensor.hpp"
```

Include dependency graph for ConicalSensor.cpp:



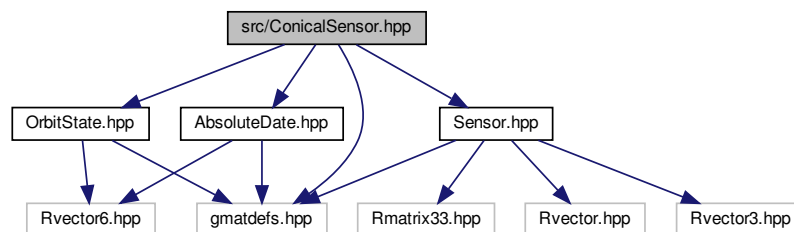
## 6.6 src/ConicalSensor.hpp File Reference

```

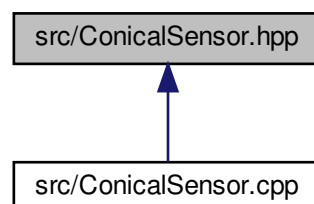
#include "gmatsdefs.hpp"
#include "OrbitState.hpp"
#include "AbsoluteDate.hpp"
#include "Sensor.hpp"

```

Include dependency graph for ConicalSensor.hpp:

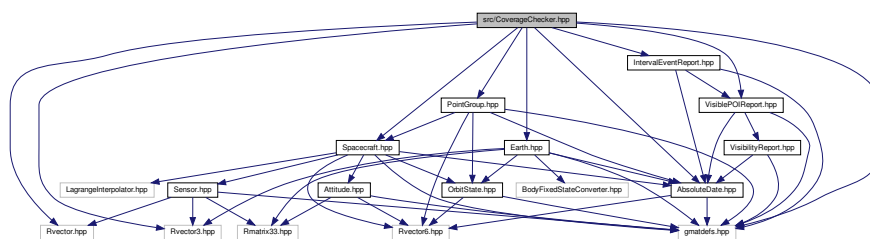
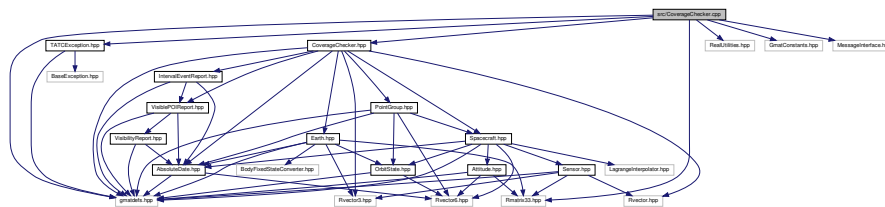


This graph shows which files directly or indirectly include this file:

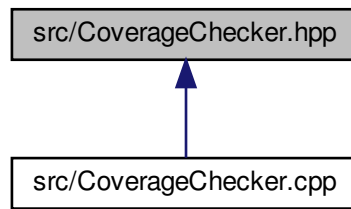




- class **ConicalSensor**



This graph shows which files directly or indirectly include this file:



## Classes

- class [CoverageChecker](#)

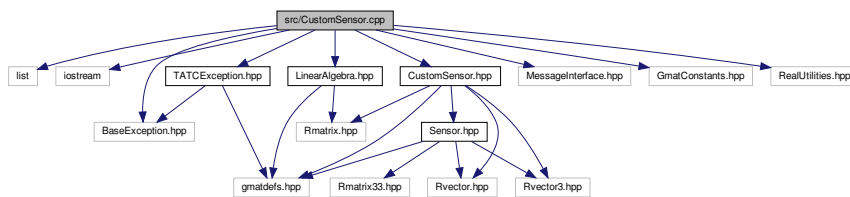
## 6.9 src/CustomSensor.cpp File Reference

```

#include <list>
#include <iostream>
#include "CustomSensor.hpp"
#include "MessageInterface.hpp"
#include "BaseException.hpp"
#include "TATCException.hpp"
#include "GmatConstants.hpp"
#include "RealUtilities.hpp"
#include "LinearAlgebra.hpp"

```

Include dependency graph for CustomSensor.cpp:



## Functions

- bool [CompareSensorElements](#) (const [SensorElement](#) &e1, const [SensorElement](#) &e2)

### 6.9.1 Function Documentation

## 6.9.1.1 CompareSensorElements()

```
bool CompareSensorElements (
    const SensorElement & e1,
    const SensorElement & e2 )
```

compares 2 elements using ordering of values

## Parameters

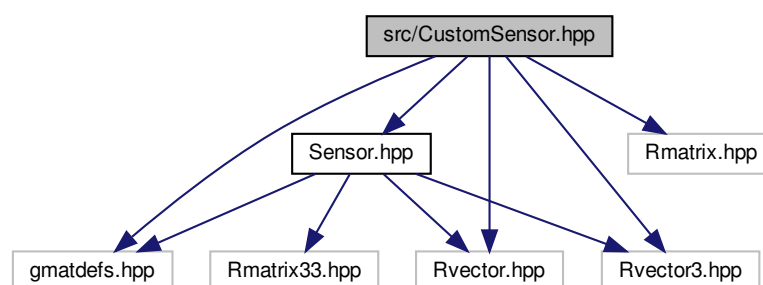
<i>e1</i>	first element in comparison
<i>e2</i>	second element in comparison

## Returns

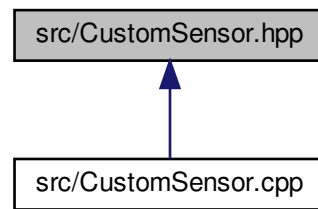
true if value of e1 is less than value of e2, false otherwise

## 6.10 src/CustomSensor.hpp File Reference

```
#include "Sensor.hpp"
#include "gmatdefs.hpp"
#include "Rvector.hpp"
#include "Rvector3.hpp"
#include "Rmatrix.hpp"
Include dependency graph for CustomSensor.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [CustomSensor](#)
- struct [SensorElement](#)

## Functions

- bool [CompareSensorElements](#) (const [SensorElement](#) &e1, const [SensorElement](#) &e2)

### 6.10.1 Function Documentation

#### 6.10.1.1 CompareSensorElements()

```
bool CompareSensorElements (  
    const SensorElement & e1,  
    const SensorElement & e2 )
```

compares 2 elements using ordering of values

#### Parameters

<i>e1</i>	first element in comparison
<i>e2</i>	second element in comparison

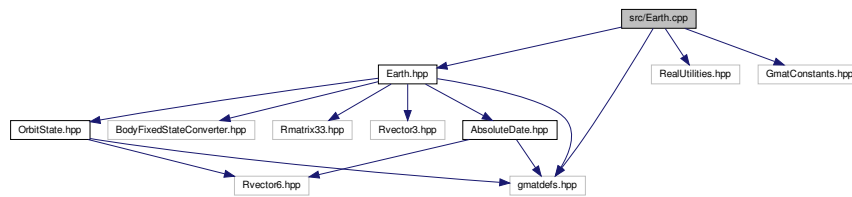
#### Returns

true if value of e1 is less than value of e2, false otherwise

## 6.11 src/Earth.cpp File Reference

```
#include "gmatdefs.hpp"
#include "Earth.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
```

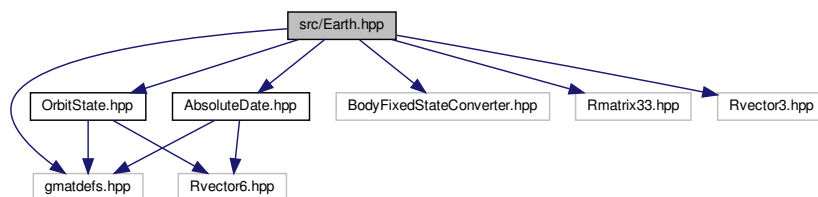
Include dependency graph for Earth.cpp:



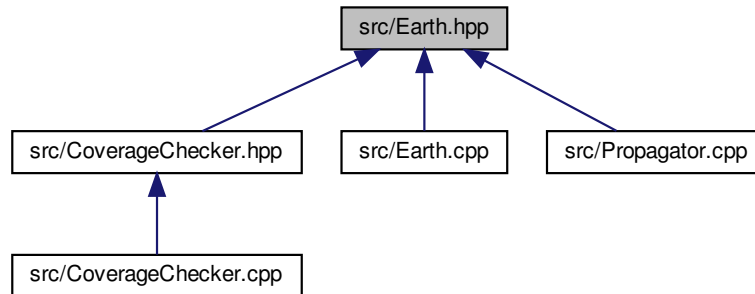
## 6.12 src/Earth.hpp File Reference

```
#include "gmatdefs.hpp"
#include "OrbitState.hpp"
#include "AbsoluteDate.hpp"
#include "BodyFixedStateConverter.hpp"
#include "Rmatrix33.hpp"
#include "Rvector3.hpp"
```

Include dependency graph for Earth.hpp:



This graph shows which files directly or indirectly include this file:



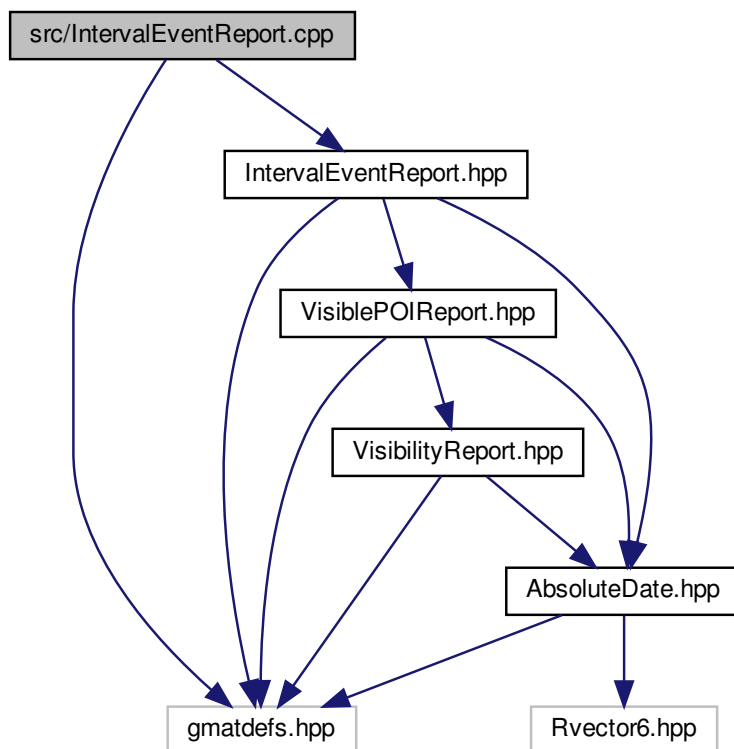
## Classes

- class [Earth](#)

## 6.13 src/IntervalEventReport.cpp File Reference

```
#include "gmatdefs.hpp"  
#include "IntervalEventReport.hpp"
```

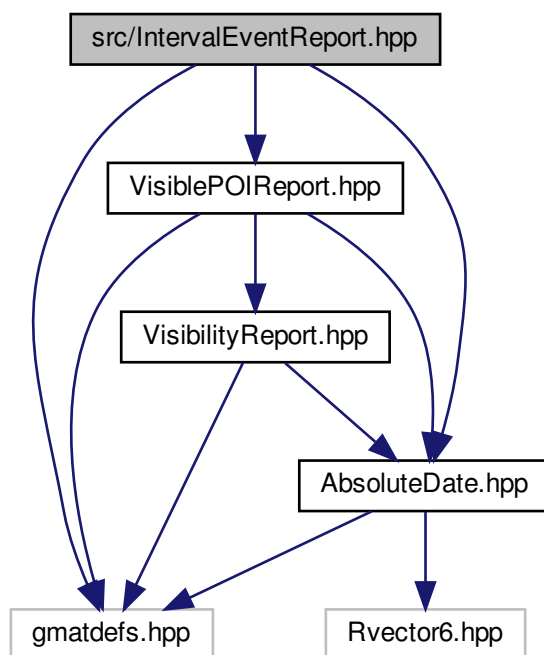
Include dependency graph for IntervalEventReport.cpp:



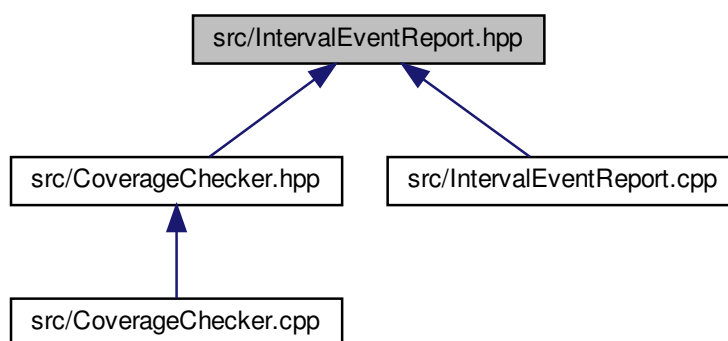
## 6.14 src/IntervalEventReport.hpp File Reference

```
#include "gmtdefs.hpp"
#include "AbsoluteDate.hpp"
#include "VisiblePOIReport.hpp"
```

Include dependency graph for `IntervalEventReport.hpp`:



This graph shows which files directly or indirectly include this file:



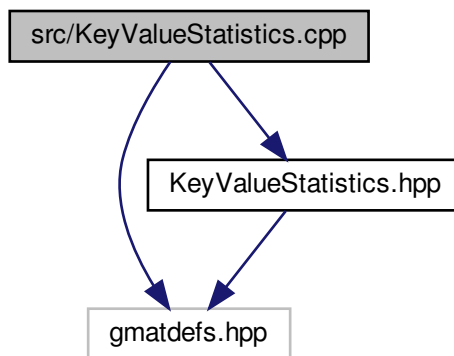
## Classes

- class [IntervalEventReport](#)



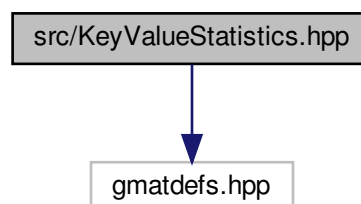
## 6.15 src/KeyValueStatistics.cpp File Reference

```
#include "gmatdefs.hpp"  
#include "KeyValueStatistics.hpp"  
Include dependency graph for KeyValueStatistics.cpp:
```

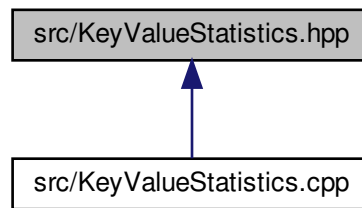


## 6.16 src/KeyValueStatistics.hpp File Reference

```
#include "gmatdefs.hpp"  
Include dependency graph for KeyValueStatistics.hpp:
```



This graph shows which files directly or indirectly include this file:



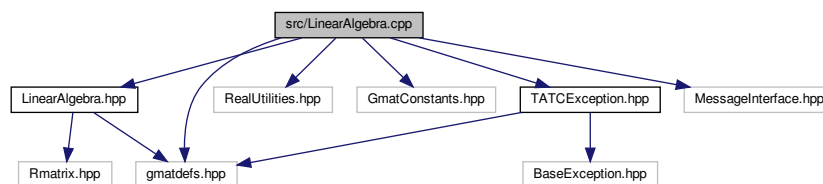
## Classes

- class [KeyValueStatistics](#)

## 6.17 src/LinearAlgebra.cpp File Reference

```
#include "gmatdefs.hpp"
#include "LinearAlgebra.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "TATCException.hpp"
#include "MessageInterface.hpp"
```

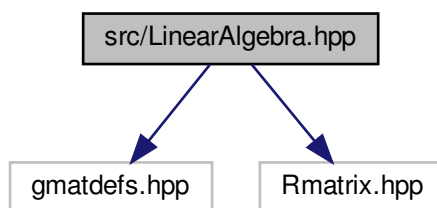
Include dependency graph for LinearAlgebra.cpp:



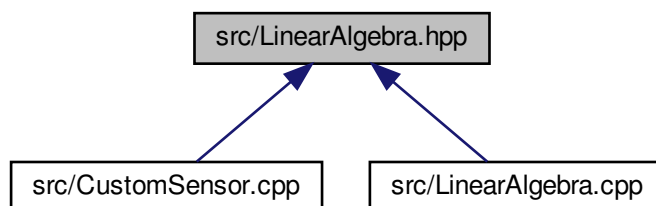
## 6.18 src/LinearAlgebra.hpp File Reference

```
#include "gmatdefs.hpp"
#include "Rmatrix.hpp"
```

Include dependency graph for LinearAlgebra.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

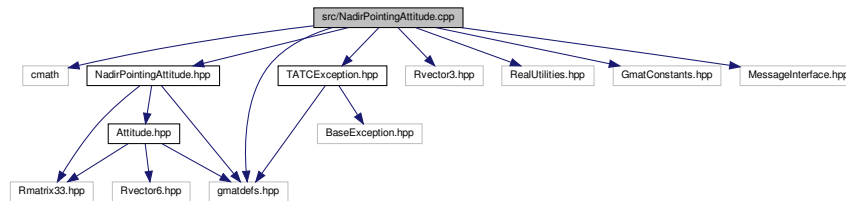
- class [LinearAlgebra](#)

## 6.19 src/NadirPointingAttitude.cpp File Reference

```
#include <cmath>
#include "gmdefs.hpp"
#include "Rvector3.hpp"
#include "NadirPointingAttitude.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "TATCException.hpp"
```

```
#include "MessageInterface.hpp"
```

Include dependency graph for NadirPointingAttitude.cpp:



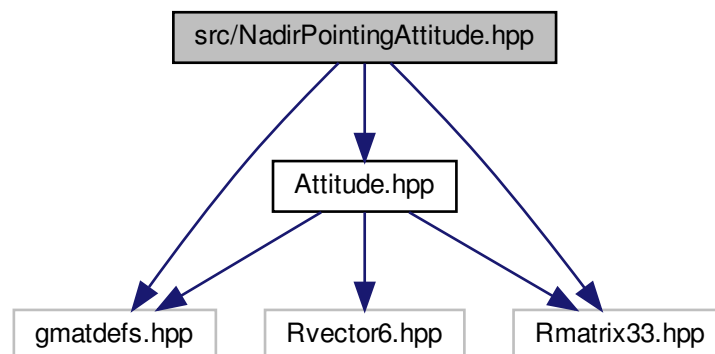
## 6.20 src/NadirPointingAttitude.hpp File Reference

```
#include "gmatdefs.hpp"
```

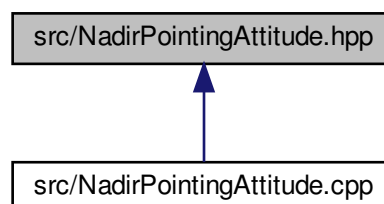
```
#include "Attitude.hpp"
```

```
#include "Rmatrix33.hpp"
```

Include dependency graph for NadirPointingAttitude.hpp:



This graph shows which files directly or indirectly include this file:



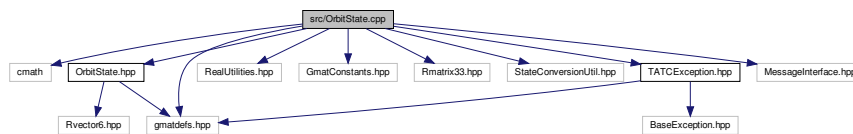
## Classes

- class [NadirPointingAttitude](#)

## 6.21 src/OrbitState.cpp File Reference

```
#include <cmath>
#include "gmatdefs.hpp"
#include "OrbitState.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "Rmatrix33.hpp"
#include "StateConversionUtil.hpp"
#include "TATCException.hpp"
#include "MessageInterface.hpp"
```

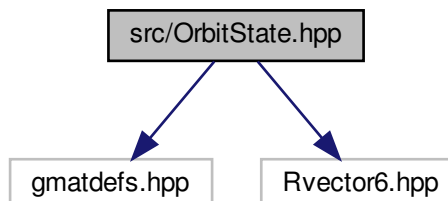
Include dependency graph for OrbitState.cpp:



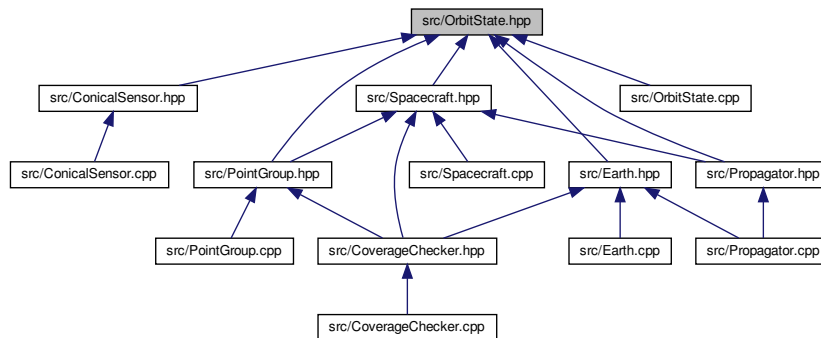
## 6.22 src/OrbitState.hpp File Reference

```
#include "gmatdefs.hpp"
#include "Rvector6.hpp"
```

Include dependency graph for OrbitState.hpp:



This graph shows which files directly or indirectly include this file:

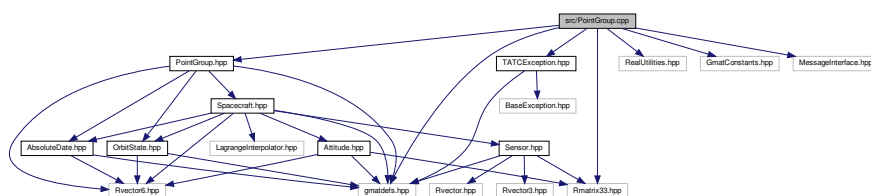


## Classes

- class [OrbitState](#)

## 6.23 src/PointGroup.cpp File Reference

```
#include "gmatdefs.hpp"
#include "PointGroup.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "Rmatrix33.hpp"
#include "TATCException.hpp"
#include "MessageInterface.hpp"
Include dependency graph for PointGroup.cpp:
```

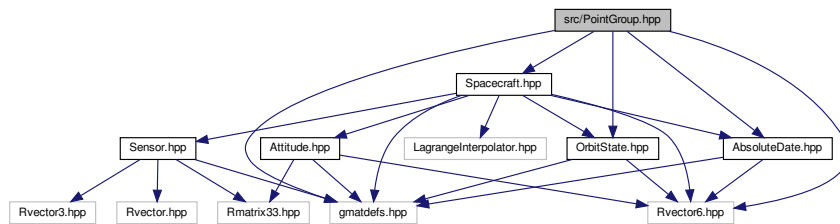


## 6.24 src/PointGroup.hpp File Reference

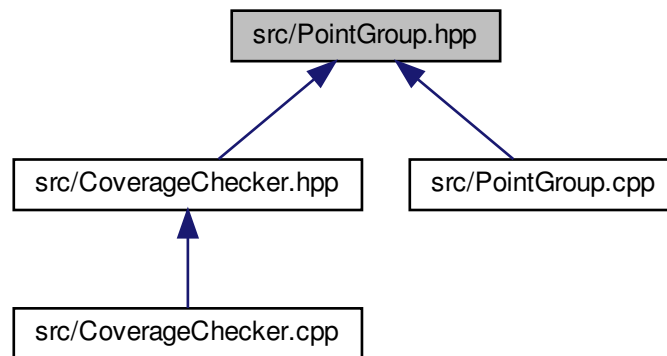
```
#include "gmatdefs.hpp"
#include "AbsoluteDate.hpp"
#include "Spacecraft.hpp"
#include "OrbitState.hpp"
```

```
#include "Rvector6.hpp"
```

Include dependency graph for PointGroup.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

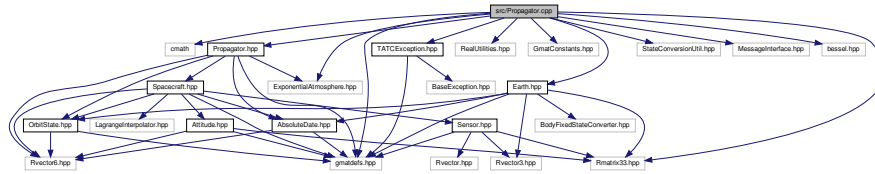
- class [PointGroup](#)

## 6.25 src/Propagator.cpp File Reference

```
#include <cmath>
#include "gmatdefs.hpp"
#include "Propagator.hpp"
#include "RealUtilities.hpp"
#include "GmatConstants.hpp"
#include "Rmatrix33.hpp"
#include "TATCException.hpp"
#include "StateConversionUtil.hpp"
#include "MessageInterface.hpp"
#include "bessel.hpp"
#include "ExponentialAtmosphere.hpp"
```

```
#include "Earth.hpp"
```

Include dependency graph for Propagator.cpp:



## 6.26 src/Propagator.hpp File Reference

```
#include "gmatdefs.hpp"
```

```
#include "AbsoluteDate.hpp"
```

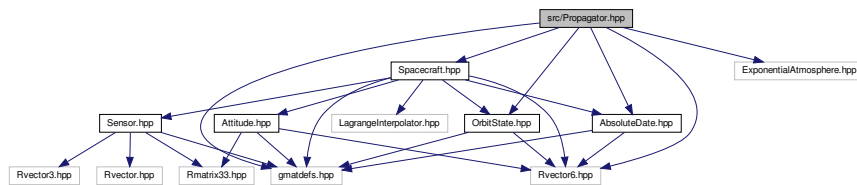
```
#include "Spacecraft.hpp"
```

```
#include "OrbitState.hpp"
```

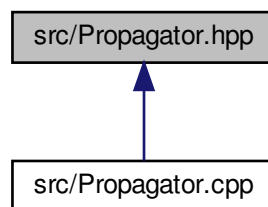
```
#include "Rvector6.hpp"
```

```
#include "ExponentialAtmosphere.hpp"
```

Include dependency graph for Propagator.hpp:



This graph shows which files directly or indirectly include this file:



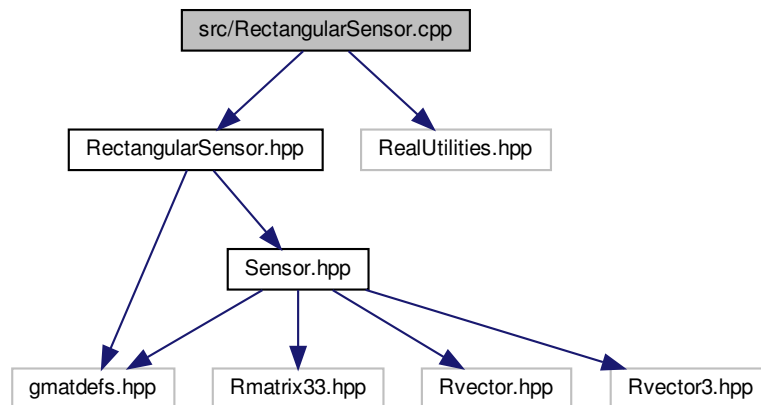
## Classes

- class [Propagator](#)



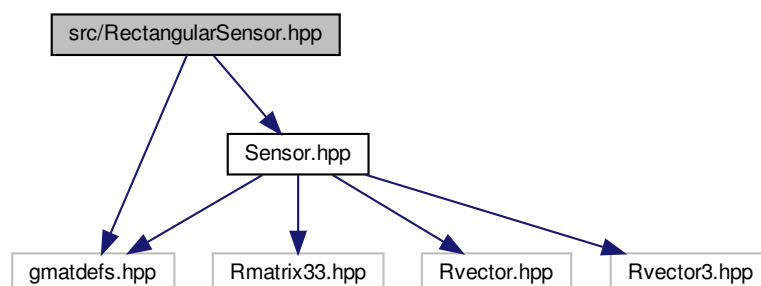
## 6.27 src/RectangularSensor.cpp File Reference

```
#include "RectangularSensor.hpp"  
#include "RealUtilities.hpp"  
Include dependency graph for RectangularSensor.cpp:
```

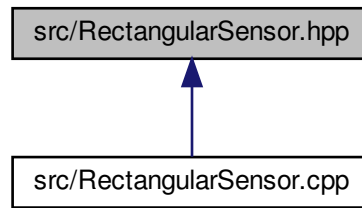


## 6.28 src/RectangularSensor.hpp File Reference

```
#include "gmtdefs.hpp"  
#include "Sensor.hpp"  
Include dependency graph for RectangularSensor.hpp:
```



This graph shows which files directly or indirectly include this file:

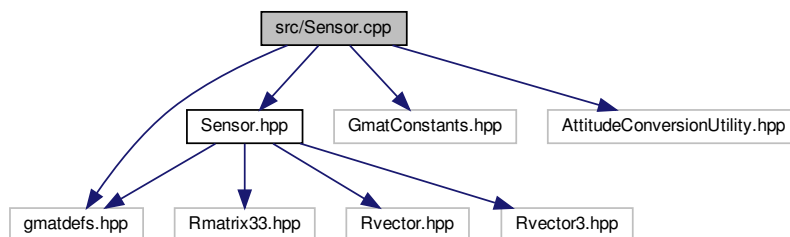


## Classes

- class [RectangularSensor](#)

## 6.29 src/Sensor.cpp File Reference

```
#include "gmatdefs.hpp"
#include "Sensor.hpp"
#include "GmatConstants.hpp"
#include "AttitudeConversionUtility.hpp"
Include dependency graph for Sensor.cpp:
```

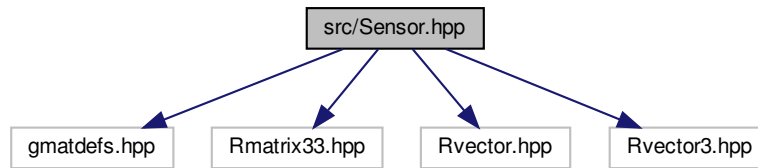


## 6.30 src/Sensor.hpp File Reference

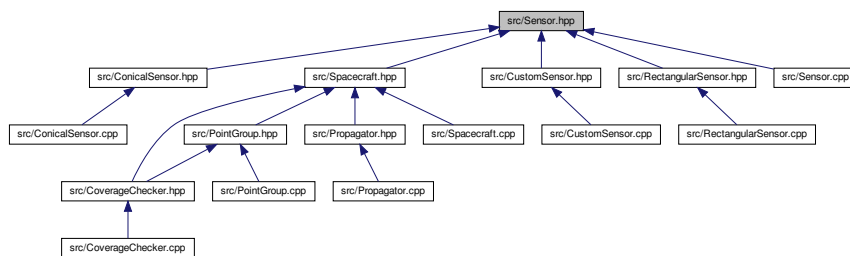
```
#include "gmatdefs.hpp"
#include "Rmatrix33.hpp"
#include "Rvector.hpp"
```

```
#include "Rvector3.hpp"
```

Include dependency graph for Sensor.hpp:



This graph shows which files directly or indirectly include this file:



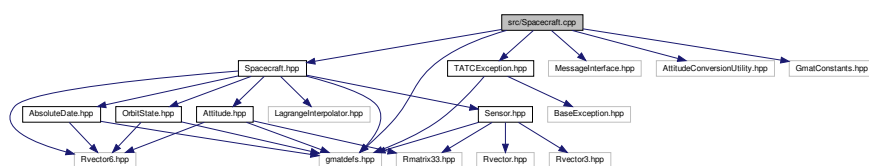
## Classes

- class [Sensor](#)

## 6.31 src/Spacecraft.cpp File Reference

```
#include "gmatsdefs.hpp"
#include "Spacecraft.hpp"
#include "TATCException.hpp"
#include "MessageInterface.hpp"
#include "AttitudeConversionUtility.hpp"
#include "GmatConstants.hpp"
```

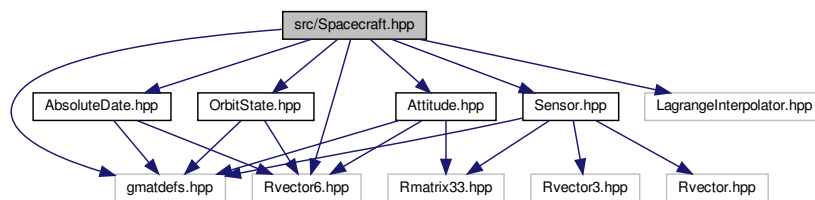
Include dependency graph for Spacecraft.cpp:



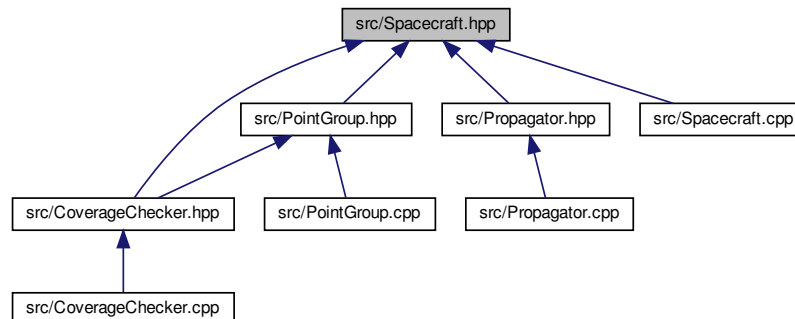
## 6.32 src/Spacecraft.hpp File Reference

```
#include "gmatdefs.hpp"
#include "OrbitState.hpp"
#include "AbsoluteDate.hpp"
#include "Sensor.hpp"
#include "Attitude.hpp"
#include "LagrangeInterpolator.hpp"
#include "Rvector6.hpp"
```

Include dependency graph for Spacecraft.hpp:



This graph shows which files directly or indirectly include this file:



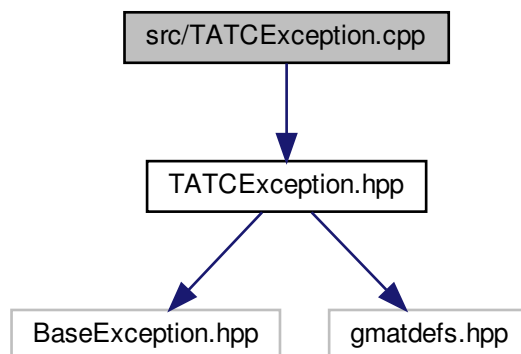
## Classes

- class [Spacecraft](#)

## 6.33 src/TATCException.cpp File Reference

```
#include "TATCException.hpp"
```

Include dependency graph for TATCException.cpp:

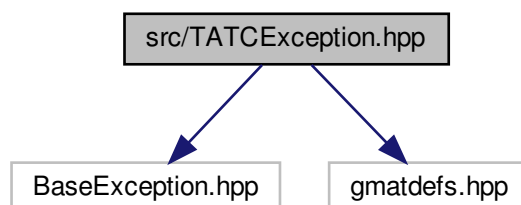


## 6.34 src/TATCException.hpp File Reference

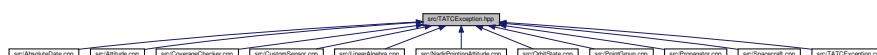
```
#include "BaseException.hpp"
#include "gmatdefs.hpp"

```

Include dependency graph for TATCException.hpp:



This graph shows which files directly or indirectly include this file:

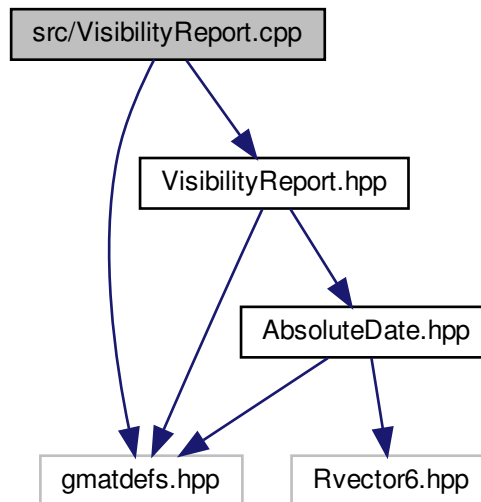


## Classes

- class [TATCException](#)

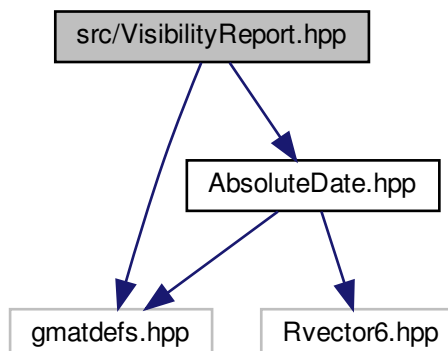
### 6.35 src/VisibilityReport.cpp File Reference

```
#include "gmatdefs.hpp"  
#include "VisibilityReport.hpp"  
Include dependency graph for VisibilityReport.cpp:
```

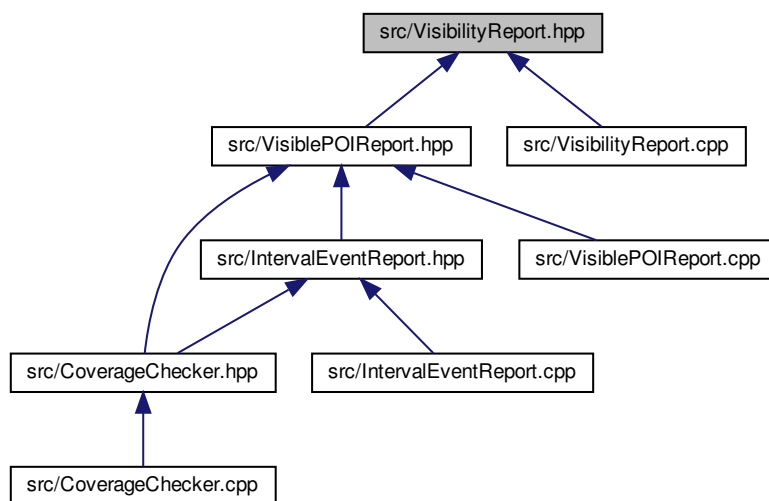


### 6.36 src/VisibilityReport.hpp File Reference

```
#include "gmatdefs.hpp"  
#include "AbsoluteDate.hpp"  
Include dependency graph for VisibilityReport.hpp:
```



This graph shows which files directly or indirectly include this file:



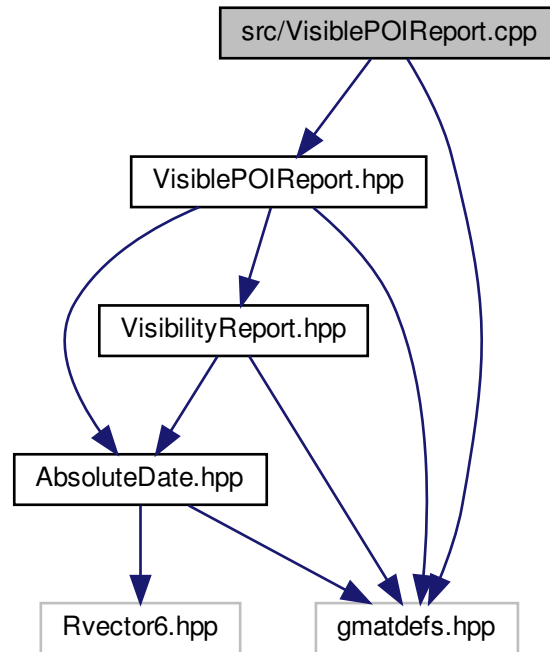
## Classes

- class [VisibilityReport](#)

## 6.37 src/VisiblePOIReport.cpp File Reference

```
#include "gmatdefs.hpp"  
#include "VisiblePOIReport.hpp"
```

Include dependency graph for VisiblePOIReport.cpp:

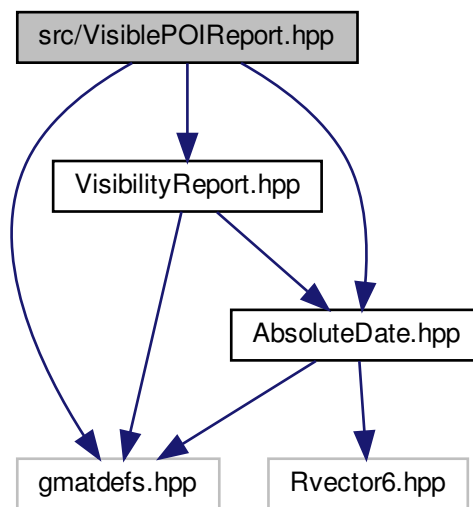


### 6.38 `src/VisiblePOIReport.hpp` File Reference

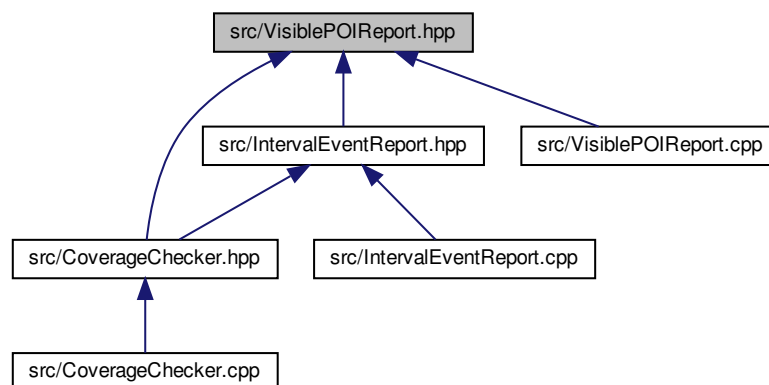
```
#include "gmatdefs.hpp"
#include "AbsoluteDate.hpp"
#include "VisibilityReport.hpp"
```



Include dependency graph for VisiblePOIReport.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [VisiblePOIReport](#)



# Index

- ~AbsoluteDate
  - AbsoluteDate, [10](#)
- ~Attitude
  - Attitude, [15](#)
- ~ConicalSensor
  - ConicalSensor, [20](#)
- ~CoverageChecker
  - CoverageChecker, [24](#)
- ~CustomSensor
  - CustomSensor, [33](#)
- ~Earth
  - Earth, [40](#)
- ~IntervalEventReport
  - IntervalEventReport, [48](#)
- ~KeyValueStatistics
  - KeyValueStatistics, [53](#)
- ~NadirPointingAttitude
  - NadirPointingAttitude, [59](#)
- ~OrbitState
  - OrbitState, [62](#)
- ~PointGroup
  - PointGroup, [68](#)
- ~Propagator
  - Propagator, [77](#)
- ~RectangularSensor
  - RectangularSensor, [88](#)
- ~Sensor
  - Sensor, [93](#)
- ~Spacecraft
  - Spacecraft, [102](#)
- ~VisibilityReport
  - VisibilityReport, [118](#)
- ~VisiblePOIReport
  - VisiblePOIReport, [122](#)
- AOP
  - Propagator, [82](#)
- AbsoluteDate, [9](#)
  - ~AbsoluteDate, [10](#)
  - AbsoluteDate, [10](#)
  - Advance, [11](#)
  - Clone, [11](#)
  - currentDate, [13](#)
  - DAYS\_PER\_MONTH, [14](#)
  - GetGregorianDate, [11](#)
  - GetJulianDate, [11](#)
  - GregorianToJulianDate, [12](#)
  - JD\_1900, [14](#)
  - operator=, [12](#)
  - SetGregorianDate, [13](#)
  - SetJulianDate, [13](#)
- AccumulateCoverageData
  - CoverageChecker, [24](#)
- AccumulatePoints
  - PointGroup, [68](#)
- AddHelicalPointsByAngle
  - PointGroup, [68](#)
- AddHelicalPointsByNumPoints
  - PointGroup, [68](#)
- AddPOIEvent
  - IntervalEventReport, [48](#)
- AddSensor
  - Spacecraft, [103](#)
- AddUserDefinedPoints
  - PointGroup, [69](#)
- Advance
  - AbsoluteDate, [11](#)
- angleHeight
  - RectangularSensor, [90](#)
- angleWidth
  - RectangularSensor, [90](#)
- applyDrag
  - Propagator, [82](#)
- argPeriapsisRate
  - Propagator, [82](#)
- Attitude, [14](#)
  - ~Attitude, [15](#)
  - Attitude, [15](#)
  - Clone, [16](#)
  - InertialToReference, [16](#)
  - operator=, [16](#)
- attitude
  - Spacecraft, [112](#)
- avgValue
  - KeyValueStatistics, [54](#)
- BODY\_RADIUS
  - CoverageChecker, [28](#)
- bfState
  - CoverageChecker, [28](#)
- bodyUnit
  - CoverageChecker, [28](#)
- CanInterpolate
  - Spacecraft, [103](#)
- centralBody
  - CoverageChecker, [28](#)
- centralBodyFixedPos
  - NadirPointingAttitude, [60](#)
- centralBodyFixedVel

- NadirPointingAttitude, 60
- CheckGridFeasibility
  - CoverageChecker, 25
- CheckHasPoints
  - PointGroup, 69
- CheckPointCoverage
  - CoverageChecker, 26
- CheckRegionVisibility
  - CustomSensor, 33
- CheckTargetMaxExcursionAngle
  - Sensor, 93
- CheckTargetMaxExcursionCoordinates
  - CustomSensor, 33
- CheckTargetVisibility
  - ConicalSensor, 20
  - CustomSensor, 34
  - RectangularSensor, 88
  - Sensor, 93
  - Spacecraft, 103, 105
- clockAngleVec
  - CustomSensor, 37
- Clone
  - AbsoluteDate, 11
  - Attitude, 16
  - OrbitState, 62
- CompareSensorElements
  - CustomSensor.cpp, 134
  - CustomSensor.hpp, 136
- ComputeArgumentOfPeriapsisRate
  - Propagator, 77
- ComputeBodyToSensorMatrix
  - Sensor, 95
- ComputeDragEffects
  - Propagator, 77
- ComputeExternalPoints
  - CustomSensor, 34
- ComputeGMT
  - Earth, 40
- ComputeHelicalPoints
  - PointGroup, 69
- ComputeMeanMotionRate
  - Propagator, 78
- ComputeNadirToBodyMatrix
  - Spacecraft, 105
- ComputeOrbitRates
  - Propagator, 78
- computePOIGeometryData
  - CoverageChecker, 28
- ComputePeriapsisAltitude
  - Propagator, 78
- ComputeRightAscensionNodeRate
  - Propagator, 78
- ComputeTestPoints
  - PointGroup, 70
- coneAngleVec
  - CustomSensor, 37
- ConeClockArraysToStereographic
  - Sensor, 95
- ConeClockToStereographic
  - Sensor, 95
- ConeClocktoRADEC
  - Sensor, 95
- ConicalSensor, 17
  - ~ConicalSensor, 20
  - CheckTargetVisibility, 20
  - ConicalSensor, 18
  - fieldOfView, 21
  - GetFieldOfView, 20
  - operator=, 20
  - SetFieldOfView, 21
- Convert
  - Earth, 41
- ConvertCartesianToKeplerian
  - OrbitState, 62
- ConvertKeplerianToCartesian
  - OrbitState, 63
- coords
  - PointGroup, 72
- CoverageChecker, 22
  - ~CoverageChecker, 24
  - AccumulateCoverageData, 24
  - BODY\_RADIUS, 28
  - bfState, 28
  - bodyUnit, 28
  - centralBody, 28
  - CheckGridFeasibility, 25
  - CheckPointCoverage, 26
  - computePOIGeometryData, 28
  - CoverageChecker, 23, 24
  - coverageEnd, 28
  - coverageStart, 29
  - CreateNewPOIReport, 26
  - dateData, 29
  - discreteEventData, 29
  - feasibilityTest, 29
  - GetEarthFixedSatState, 26
  - numEventsPerPoint, 29
  - operator=, 27
  - pointArray, 29
  - pointGroup, 30
  - ProcessCoverageData, 27
  - ptPos, 30
  - rangeVec, 30
  - sc, 30
  - SetComputePOIGeometryData, 27
  - timeldx, 30
  - timeSeriesData, 30
- coverageEnd
  - CoverageChecker, 28
- coverageStart
  - CoverageChecker, 29
- CreateNewPOIReport
  - CoverageChecker, 26
- currentDate
  - AbsoluteDate, 13
- currentState

- OrbitState, 66
- CustomSensor, 31
  - ~CustomSensor, 33
  - CheckRegionVisibility, 33
  - CheckTargetMaxExcursionCoordinates, 33
  - CheckTargetVisibility, 34
  - clockAngleVec, 37
  - ComputeExternalPoints, 34
  - coneAngleVec, 37
  - CustomSensor, 32, 33
  - externalPointArray, 37
  - Max, 34
  - maxXExcursion, 37
  - maxYExcursion, 37
  - Min, 34
  - minXExcursion, 37
  - minYExcursion, 38
  - numFOVPoints, 38
  - numTestPoints, 38
  - operator=, 36
  - PointsToSegments, 36
  - RegionsFullyContained, 36
  - segmentArray, 38
  - Sort, 36
  - xProjectionCoordArray, 38
  - yProjectionCoordArray, 38
- CustomSensor.cpp
  - CompareSensorElements, 134
- CustomSensor.hpp
  - CompareSensorElements, 136
- DAYS\_PER\_MONTH
  - AbsoluteDate, 14
- dateData
  - CoverageChecker, 29
- densityModel
  - Propagator, 82
- discreteEventData
  - CoverageChecker, 29
- discretePOIEvents
  - IntervalEventReport, 51
- dragArea
  - Spacecraft, 112
- dragCoefficient
  - Spacecraft, 112
- ECC
  - Propagator, 82
- Earth, 39
  - ~Earth, 40
  - ComputeGMT, 40
  - Convert, 41
  - Earth, 40
  - FixedToTopo, 41
  - FixedToTopocentric, 42
  - flattening, 45
  - GeocentricToGeodeticLat, 42
  - GetBodyFixedState, 42
  - GetEarthSunDistRaDec, 43
  - GetInertialToFixedRotation, 43
  - GetRadius, 44
  - GetSunPositionInBodyCoords, 44
  - InertialToBodyFixed, 44
  - J2, 45
  - lastRotationTime, 45
  - mu, 46
  - operator=, 45
  - radius, 46
  - rotationResult, 46
- endDate
  - IntervalEventReport, 51
  - VisibilityReport, 120
- eqRadius
  - Propagator, 82
- eulerSeq1
  - Sensor, 97
  - Spacecraft, 112
- eulerSeq2
  - Sensor, 97
  - Spacecraft, 113
- eulerSeq3
  - Sensor, 97
  - Spacecraft, 113
- externalPointArray
  - CustomSensor, 37
- feasibilityTest
  - CoverageChecker, 29
- fieldOfView
  - ConicalSensor, 21
- FixedToTopo
  - Earth, 41
- FixedToTopocentric
  - Earth, 42
- flattening
  - Earth, 45
- GeocentricToGeodeticLat
  - Earth, 42
- GetAngleHeight
  - RectangularSensor, 88
- GetAngleWidth
  - RectangularSensor, 89
- GetApplyDrag
  - Propagator, 78
- GetAvgValue
  - KeyValueStatistics, 53
- GetBodyFixedState
  - Earth, 42
- GetBodyFixedToInertial
  - Spacecraft, 105
- GetBodyToSensorMatrix
  - Sensor, 96
- GetCartesianState
  - OrbitState, 63
  - Spacecraft, 106
- GetCartesianStateAtEpoch
  - Spacecraft, 106

- GetDragArea
  - Spacecraft, 107
- GetDragCoefficient
  - Spacecraft, 107
- GetEarthFixedSatState
  - CoverageChecker, 26
- GetEarthSunDistRaDec
  - Earth, 43
- GetEndDate
  - IntervalEventReport, 48
  - VisibilityReport, 118
- GetFieldOfView
  - ConicalSensor, 20
- GetGregorianDate
  - AbsoluteDate, 11
- GetInertialToFixedRotation
  - Earth, 43
- GetJulianDate
  - AbsoluteDate, 11
  - Spacecraft, 107
- GetKeplerianState
  - OrbitState, 64
- GetLatAndLon
  - PointGroup, 70
- GetLatLonVectors
  - PointGroup, 70
- GetMaxValue
  - KeyValueStatistics, 53
- GetMinValue
  - KeyValueStatistics, 54
- GetNumPoints
  - PointGroup, 71
- GetObsAzimuth
  - VisiblePOIReport, 122
- GetObsRange
  - VisiblePOIReport, 123
- GetObsZenith
  - VisiblePOIReport, 123
- GetOrbitEpoch
  - Spacecraft, 107
- GetOrbitState
  - Spacecraft, 107
- GetPOIEvents
  - IntervalEventReport, 49
- GetPOIIndex
  - IntervalEventReport, 49
  - VisiblePOIReport, 123
- GetPointPositionVector
  - PointGroup, 71
- GetPropStartEnd
  - Propagator, 79
- GetRadius
  - Earth, 44
- GetStartDate
  - IntervalEventReport, 49
  - VisibilityReport, 118
- GetSunAzimuth
  - VisiblePOIReport, 123
- GetSunPositionInBodyCoords
  - Earth, 44
- GetSunZenith
  - VisiblePOIReport, 124
- GetTotalMass
  - Spacecraft, 108
- GregorianToJulianDate
  - AbsoluteDate, 12
- HasSensors
  - Spacecraft, 108
- INC
  - Propagator, 83
- index
  - SensorElement, 99
- InertialToBodyFixed
  - Earth, 44
- InertialToConeClock
  - Spacecraft, 108
- InertialToReference
  - Attitude, 16
  - NadirPointingAttitude, 59
- Interpolate
  - Spacecraft, 109
- interpolator
  - Spacecraft, 113
- IntervalEventReport, 46
  - ~IntervalEventReport, 48
  - AddPOIEvent, 48
  - discretePOIEvents, 51
  - endDate, 51
  - GetEndDate, 48
  - GetPOIEvents, 49
  - GetPOIIndex, 49
  - GetStartDate, 49
  - IntervalEventReport, 47, 48
  - operator=, 49
  - poiIndex, 51
  - SetAllPOIEvents, 50
  - SetEndDate, 50
  - SetPOIIndex, 50
  - SetStartDate, 51
  - startDate, 51
- J2
  - Earth, 45
  - Propagator, 83
- JD\_1900
  - AbsoluteDate, 14
- KeyValueStatistics, 52
  - ~KeyValueStatistics, 53
  - avgValue, 54
  - GetAvgValue, 53
  - GetMaxValue, 53
  - GetMinValue, 54
  - KeyValueStatistics, 52, 53
  - maxValue, 54

- minValue, [55](#)
- operator=, [54](#)
- lastDragUpdateEpoch
  - Propagator, [83](#)
- lastRotationTime
  - Earth, [45](#)
- lat
  - PointGroup, [72](#)
- latLower
  - PointGroup, [72](#)
- latUpper
  - PointGroup, [72](#)
- LineSegmentIntersect
  - LinearAlgebra, [55](#)
- LinearAlgebra, [55](#)
  - LineSegmentIntersect, [55](#)
- lon
  - PointGroup, [73](#)
- lonLower
  - PointGroup, [73](#)
- lonUpper
  - PointGroup, [73](#)
- MU\_FOR\_EARTH
  - Propagator, [84](#)
- MA
  - Propagator, [83](#)
- Max
  - CustomSensor, [34](#)
- maxExcursionAngle
  - Sensor, [98](#)
- maxValue
  - KeyValueStatistics, [54](#)
- maxXExcursion
  - CustomSensor, [37](#)
- maxYExcursion
  - CustomSensor, [37](#)
- MeanMotion
  - Propagator, [79](#)
- meanMotion
  - Propagator, [83](#)
- meanMotionRate
  - Propagator, [83](#)
- Min
  - CustomSensor, [34](#)
- minValue
  - KeyValueStatistics, [55](#)
- minXExcursion
  - CustomSensor, [37](#)
- minYExcursion
  - CustomSensor, [38](#)
- mu
  - Earth, [46](#)
  - OrbitState, [66](#)
  - Propagator, [84](#)
- NadirPointingAttitude, [57](#)
  - ~NadirPointingAttitude, [59](#)
- centralBodyFixedPos, [60](#)
- centralBodyFixedVel, [60](#)
- InertialToReference, [59](#)
- NadirPointingAttitude, [58](#)
- operator=, [59](#)
- R\_fixed\_to\_nadir, [60](#)
- R\_fixed\_to\_nadir\_transposed, [60](#)
- xHat, [60](#)
- yHat, [60](#)
- zHat, [60](#)
- numEventsPerPoint
  - CoverageChecker, [29](#)
- numFOVPoints
  - CustomSensor, [38](#)
- numPoints
  - PointGroup, [73](#)
- numRequestedPoints
  - PointGroup, [73](#)
- numSensors
  - Spacecraft, [113](#)
- numTestPoints
  - CustomSensor, [38](#)
- obsAzimuth
  - VisiblePOIReport, [126](#)
- obsRange
  - VisiblePOIReport, [126](#)
- obsZenith
  - VisiblePOIReport, [126](#)
- offsetAngle1
  - Sensor, [98](#)
  - Spacecraft, [113](#)
- offsetAngle2
  - Sensor, [98](#)
  - Spacecraft, [113](#)
- offsetAngle3
  - Sensor, [98](#)
  - Spacecraft, [113](#)
- operator=
  - AbsoluteDate, [12](#)
  - Attitude, [16](#)
  - ConicalSensor, [20](#)
  - CoverageChecker, [27](#)
  - CustomSensor, [36](#)
  - Earth, [45](#)
  - IntervalEventReport, [49](#)
  - KeyValueStatistics, [54](#)
  - NadirPointingAttitude, [59](#)
  - OrbitState, [64](#)
  - PointGroup, [71](#)
  - Propagator, [79](#)
  - RectangularSensor, [89](#)
  - Sensor, [96](#)
  - Spacecraft, [109](#)
  - VisibilityReport, [119](#)
  - VisiblePOIReport, [124](#)
- orbitEpoch
  - Spacecraft, [114](#)
- orbitPeriod

- Propagator, 84
- OrbitState, 61
  - ~OrbitState, 62
  - Clone, 62
  - ConvertCartesianToKeplerian, 62
  - ConvertKeplerianToCartesian, 63
  - currentState, 66
  - GetCartesianState, 63
  - GetKeplerianState, 64
  - mu, 66
  - operator=, 64
  - OrbitState, 62
  - SetCartesianState, 64
  - SetGravityParameter, 64
  - SetKeplerianState, 65
  - SetKeplerianVectorState, 65
- orbitState
  - Spacecraft, 114
- poiIndex
  - IntervalEventReport, 51
  - VisiblePOIReport, 127
- pointArray
  - CoverageChecker, 29
- PointGroup, 66
  - ~PointGroup, 68
  - AccumulatePoints, 68
  - AddHelicalPointsByAngle, 68
  - AddHelicalPointsByNumPoints, 68
  - AddUserDefinedPoints, 69
  - CheckHasPoints, 69
  - ComputeHelicalPoints, 69
  - ComputeTestPoints, 70
  - coords, 72
  - GetLatAndLon, 70
  - GetLatLonVectors, 70
  - GetNumPoints, 71
  - GetPointPositionVector, 71
  - lat, 72
  - latLower, 72
  - latUpper, 72
  - lon, 73
  - lonLower, 73
  - lonUpper, 73
  - numPoints, 73
  - numRequestedPoints, 73
  - operator=, 71
  - PointGroup, 67
  - SetLatLonBounds, 72
- pointGroup
  - CoverageChecker, 30
- PointsToSegments
  - CustomSensor, 36
- ProcessCoverageData
  - CoverageChecker, 27
- propEnd
  - Propagator, 84
- propStart
  - Propagator, 84
- Propagate
  - Propagator, 80
- PropagateOrbitalElements
  - Propagator, 80
- Propagator, 74
  - ~Propagator, 77
  - AOP, 82
  - applyDrag, 82
  - argPeriapsisRate, 82
  - ComputeArgumentOfPeriapsisRate, 77
  - ComputeDragEffects, 77
  - ComputeMeanMotionRate, 78
  - ComputeOrbitRates, 78
  - ComputePeriapsisAltitude, 78
  - ComputeRightAscensionNodeRate, 78
  - densityModel, 82
  - ECC, 82
  - eqRadius, 82
  - GetApplyDrag, 78
  - GetPropStartEnd, 79
  - INC, 83
  - J2, 83
  - lastDragUpdateEpoch, 83
  - MU\_FOR\_EARTH, 84
  - MA, 83
  - MeanMotion, 79
  - meanMotion, 83
  - meanMotionRate, 83
  - mu, 84
  - operator=, 79
  - orbitPeriod, 84
  - propEnd, 84
  - propStart, 84
  - Propagate, 80
  - PropagateOrbitalElements, 80
  - Propagator, 76
  - RAAN, 84
  - refJd, 85
  - rightAscensionNodeRate, 85
  - SMA, 85
  - sc, 85
  - semiLatusRectum, 85
  - SemiParameter, 80
  - SetApplyDrag, 80
  - SetOrbitState, 81
  - SetPhysicalConstants, 81
  - TA, 85
- ptPos
  - CoverageChecker, 30
- R\_BN
  - Spacecraft, 114
- R\_SB
  - Sensor, 98
- R\_fixed\_to\_nadir
  - NadirPointingAttitude, 60
- R\_fixed\_to\_nadir\_transposed
  - NadirPointingAttitude, 60
- RAAN



- Propagator, [84](#)
- RADECtoUnitVec
  - Sensor, [96](#)
- radius
  - Earth, [46](#)
- rangeVec
  - CoverageChecker, [30](#)
- RectangularSensor, [86](#)
  - ~RectangularSensor, [88](#)
  - angleHeight, [90](#)
  - angleWidth, [90](#)
  - CheckTargetVisibility, [88](#)
  - GetAngleHeight, [88](#)
  - GetAngleWidth, [89](#)
  - operator=, [89](#)
  - RectangularSensor, [87](#), [88](#)
  - SetAngleHeight, [89](#)
  - SetAngleWidth, [90](#)
- refJd
  - Propagator, [85](#)
- RegionIsFullyContained
  - CustomSensor, [36](#)
- rightAscensionNodeRate
  - Propagator, [85](#)
- rotationResult
  - Earth, [46](#)
- SMA
  - Propagator, [85](#)
- sc
  - CoverageChecker, [30](#)
  - Propagator, [85](#)
- segmentArray
  - CustomSensor, [38](#)
- semiLatusRectum
  - Propagator, [85](#)
- SemiParameter
  - Propagator, [80](#)
- Sensor, [91](#)
  - ~Sensor, [93](#)
  - CheckTargetMaxExcursionAngle, [93](#)
  - CheckTargetVisibility, [93](#)
  - ComputeBodyToSensorMatrix, [95](#)
  - ConeClockArraysToStereographic, [95](#)
  - ConeClockToStereographic, [95](#)
  - ConeClocktoRADEC, [95](#)
  - eulerSeq1, [97](#)
  - eulerSeq2, [97](#)
  - eulerSeq3, [97](#)
  - GetBodyToSensorMatrix, [96](#)
  - maxExcursionAngle, [98](#)
  - offsetAngle1, [98](#)
  - offsetAngle2, [98](#)
  - offsetAngle3, [98](#)
  - operator=, [96](#)
  - R\_SB, [98](#)
  - RADECtoUnitVec, [96](#)
  - Sensor, [92](#)
  - SetSensorBodyOffsetAngles, [96](#)
  - UnitVecToStereographic, [97](#)
- SensorElement, [99](#)
  - index, [99](#)
  - value, [99](#)
- sensorList
  - Spacecraft, [114](#)
- SetAllPOIEvents
  - IntervalEventReport, [50](#)
- SetAngleHeight
  - RectangularSensor, [89](#)
- SetAngleWidth
  - RectangularSensor, [90](#)
- SetApplyDrag
  - Propagator, [80](#)
- SetAttitude
  - Spacecraft, [109](#)
- SetBodyNadirOffsetAngles
  - Spacecraft, [110](#)
- SetCartesianState
  - OrbitState, [64](#)
- SetComputePOIGeometryData
  - CoverageChecker, [27](#)
- SetDragArea
  - Spacecraft, [110](#)
- SetDragCoefficient
  - Spacecraft, [110](#)
- SetEndDate
  - IntervalEventReport, [50](#)
  - VisibilityReport, [119](#)
- SetFieldOfView
  - ConicalSensor, [21](#)
- SetGravityParameter
  - OrbitState, [64](#)
- SetGregorianDate
  - AbsoluteDate, [13](#)
- SetJulianDate
  - AbsoluteDate, [13](#)
- SetKeplerianState
  - OrbitState, [65](#)
- SetKeplerianVectorState
  - OrbitState, [65](#)
- SetLatLonBounds
  - PointGroup, [72](#)
- SetObsAzimuth
  - VisiblePOIReport, [124](#)
- SetObsRange
  - VisiblePOIReport, [124](#)
- SetObsZenith
  - VisiblePOIReport, [125](#)
- SetOrbitState
  - Propagator, [81](#)
  - Spacecraft, [111](#)
- SetPOIIndex
  - IntervalEventReport, [50](#)
  - VisiblePOIReport, [125](#)
- SetPhysicalConstants
  - Propagator, [81](#)
- SetSensorBodyOffsetAngles

- Sensor, 96
- SetStartDate
  - IntervalEventReport, 51
  - VisibilityReport, 119
- SetSunAzimuth
  - VisiblePOIReport, 125
- SetSunZenith
  - VisiblePOIReport, 126
- SetTotalMass
  - Spacecraft, 111
- Sort
  - CustomSensor, 36
- Spacecraft, 99
  - ~Spacecraft, 102
  - AddSensor, 103
  - attitude, 112
  - CanInterpolate, 103
  - CheckTargetVisibility, 103, 105
  - ComputeNadirToBodyMatrix, 105
  - dragArea, 112
  - dragCoefficient, 112
  - eulerSeq1, 112
  - eulerSeq2, 113
  - eulerSeq3, 113
  - GetBodyFixedToInertial, 105
  - GetCartesianState, 106
  - GetCartesianStateAtEpoch, 106
  - GetDragArea, 107
  - GetDragCoefficient, 107
  - GetJulianDate, 107
  - GetOrbitEpoch, 107
  - GetOrbitState, 107
  - GetTotalMass, 108
  - HasSensors, 108
  - InertialToConeClock, 108
  - Interpolate, 109
  - interpolator, 113
  - numSensors, 113
  - offsetAngle1, 113
  - offsetAngle2, 113
  - offsetAngle3, 113
  - operator=, 109
  - orbitEpoch, 114
  - orbitState, 114
  - R\_BN, 114
  - sensorList, 114
  - SetAttitude, 109
  - SetBodyNadirOffsetAngles, 110
  - SetDragArea, 110
  - SetDragCoefficient, 110
  - SetOrbitState, 111
  - SetTotalMass, 111
  - Spacecraft, 101, 102
  - TimeToInterpolate, 111
  - totalMass, 114
- src/AbsoluteDate.cpp, 129
- src/AbsoluteDate.hpp, 129
- src/Attitude.cpp, 130
  - src/Attitude.hpp, 130
  - src/ConicalSensor.cpp, 131
  - src/ConicalSensor.hpp, 132
  - src/CoverageChecker.cpp, 133
  - src/CoverageChecker.hpp, 133
  - src/CustomSensor.cpp, 134
  - src/CustomSensor.hpp, 135
  - src/Earth.cpp, 137
  - src/Earth.hpp, 137
  - src/IntervalEventReport.cpp, 138
  - src/IntervalEventReport.hpp, 139
  - src/KeyValueStatistics.cpp, 141
  - src/KeyValueStatistics.hpp, 141
  - src/LinearAlgebra.cpp, 142
  - src/LinearAlgebra.hpp, 142
  - src/NadirPointingAttitude.cpp, 143
  - src/NadirPointingAttitude.hpp, 144
  - src/OrbitState.cpp, 145
  - src/OrbitState.hpp, 145
  - src/PointGroup.cpp, 146
  - src/PointGroup.hpp, 146
  - src/Propagator.cpp, 147
  - src/Propagator.hpp, 148
  - src/RectangularSensor.cpp, 149
  - src/RectangularSensor.hpp, 149
  - src/Sensor.cpp, 150
  - src/Sensor.hpp, 150
  - src/Spacecraft.cpp, 151
  - src/Spacecraft.hpp, 152
  - src/TATCException.cpp, 152
  - src/TATCException.hpp, 153
  - src/VisibilityReport.cpp, 154
  - src/VisibilityReport.hpp, 154
  - src/VisiblePOIReport.cpp, 155
  - src/VisiblePOIReport.hpp, 156
- startDate
  - IntervalEventReport, 51
  - VisibilityReport, 120
- sunAzimuth
  - VisiblePOIReport, 127
- sunZenith
  - VisiblePOIReport, 127
- TATCException, 115
  - TATCException, 115, 116
- TA
  - Propagator, 85
- timelidx
  - CoverageChecker, 30
- timeSeriesData
  - CoverageChecker, 30
- TimeToInterpolate
  - Spacecraft, 111
- totalMass
  - Spacecraft, 114
- UnitVecToStereographic
  - Sensor, 97

value

SensorElement, [99](#)

VisibilityReport, [116](#)

~VisibilityReport, [118](#)

endDate, [120](#)

GetEndDate, [118](#)

GetStartDate, [118](#)

operator=, [119](#)

SetEndDate, [119](#)

SetStartDate, [119](#)

startDate, [120](#)

VisibilityReport, [117](#), [118](#)

VisiblePOIReport, [120](#)

~VisiblePOIReport, [122](#)

GetObsAzimuth, [122](#)

GetObsRange, [123](#)

GetObsZenith, [123](#)

GetPOIIndex, [123](#)

GetSunAzimuth, [123](#)

GetSunZenith, [124](#)

obsAzimuth, [126](#)

obsRange, [126](#)

obsZenith, [126](#)

operator=, [124](#)

poiIndex, [127](#)

SetObsAzimuth, [124](#)

SetObsRange, [124](#)

SetObsZenith, [125](#)

SetPOIIndex, [125](#)

SetSunAzimuth, [125](#)

SetSunZenith, [126](#)

sunAzimuth, [127](#)

sunZenith, [127](#)

VisiblePOIReport, [122](#)

xHat

NadirPointingAttitude, [60](#)

xProjectionCoordArray

CustomSensor, [38](#)

yHat

NadirPointingAttitude, [60](#)

yProjectionCoordArray

CustomSensor, [38](#)

zHat

NadirPointingAttitude, [60](#)