# MultiBody Systems Virginia Tech

## MBSVT 1.0

## Reference manual [1]

Fri Aug 30 2013 13:39:35

[1] This material is based upon work supported by the National Science Foundation under Grant Number 1130667 Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# Contents


**1 Modules Index** **1**

1.1 Modules List . . . . . 1

**2 Data Type Index** **3**

2.1 Class List . . . . . 3

**3 File Index** **5**

3.1 File List . . . . . 5

**4 Module Documentation** **7**

4.1 CONSTANTS Module Reference . . . . . 7

4.1.1 Detailed Description . . . . . 8

4.1.2 Function/Subroutine Documentation . . . . . 8

4.1.2.1 initialize_CALLBACKS . . . . . 8

4.1.2.2 initialize_CONSTANTS . . . . . 9

4.1.2.3 setDIM . . . . . 9

4.1.2.4 setNIN . . . . . 9

4.1.2.5 setNMT . . . . . 9

4.1.2.6 setNRT . . . . . 9

4.1.3 Variable Documentation . . . . . 9

4.1.3.1 alfa . . . . . 9

4.1.3.2 DIM . . . . . 9

4.1.3.3 dt . . . . . 9

4.1.3.4 Dynamics . . . . . 9

4.1.3.5 E_RK . . . . . 9

4.1.3.6 E_RK2 . . . . . 9

4.1.3.7 g . . . . . . . . . . . . . . . . . . . . . . . . . . 9
4.1.3.8 I_RK . . . . . . . . . . . . . . . . . . . . . . . . 9
4.1.3.9 I_RK_ADJ . . . . . . . . . . . . . . . . . . . . . . 9
4.1.3.10 I_RK_TLM . . . . . . . . . . . . . . . . . . . . . . 9
4.1.3.11 Kinematics . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.12 maxiteppos . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.13 NIN . . . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.14 NMT . . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.15 NRT . . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.16 omega . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.17 padjinit_user . . . . . . . . . . . . . . . . . . . . 10
4.1.3.18 pdamping_user . . . . . . . . . . . . . . . . . . . 10
4.1.3.19 pdgdp_user . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.20 pdgdy_user . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.21 pforces_user . . . . . . . . . . . . . . . . . . . . 10
4.1.3.22 pgfun_user . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.23 pivgdlval . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.24 pmpv_user . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.25 pqro_user . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.26 PROTECTED . . . . . . . . . . . . . . . . . . . . 10
4.1.3.27 psi . . . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.28 pstiffness_user . . . . . . . . . . . . . . . . . . . 10
4.1.3.29 Sensitivity_ADJ . . . . . . . . . . . . . . . . . . . 10
4.1.3.30 Sensitivity_TLM . . . . . . . . . . . . . . . . . . . 10
4.1.3.31 SWFORM . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.32 SWINT . . . . . . . . . . . . . . . . . . . . . . . 10
4.1.3.33 tolNRppos . . . . . . . . . . . . . . . . . . . . . . 10
4.2 CONSTRAINTS Module Reference . . . . . . . . . . . . . . . . . . 11
4.2.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 13
4.2.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 13
4.2.2.1 ADDConstr_dot1 . . . . . . . . . . . . . . . . . . . 13
4.2.2.2 ADDConstr_Drive_dist . . . . . . . . . . . . . . . . 14
4.2.2.3 ADDConstr_Drive_eu0 . . . . . . . . . . . . . . . . 14
4.2.2.4 ADDConstr_Drive_eu1 . . . . . . . . . . . . . . . . 15

4.2.2.5 ADDConstr_Drive_eu2 . . . . . . . . . . . . . . . . . 15
4.2.2.6 ADDConstr_Drive_eu3 . . . . . . . . . . . . . . . . . 16
4.2.2.7 ADDConstr_Drive_rgEul . . . . . . . . . . . . . . . . 17
4.2.2.8 ADDConstr_Drive_rgx . . . . . . . . . . . . . . . . . 17
4.2.2.9 ADDConstr_Drive_rgy . . . . . . . . . . . . . . . . . 18
4.2.2.10 ADDConstr_Drive_rgz . . . . . . . . . . . . . . . . . 18
4.2.2.11 ADDConstr_RevJoint . . . . . . . . . . . . . . . . . 19
4.2.2.12 ADDConstr_SpheJoint . . . . . . . . . . . . . . . . . 20
4.2.2.13 ADDConstr_TransJoint . . . . . . . . . . . . . . . . . 20
4.2.2.14 ADDConstr_UnitEulParam . . . . . . . . . . . . . . . 21
4.2.2.15 CONSTRAINTS_Setup . . . . . . . . . . . . . . . . . 22
4.2.2.16 evalconstraints . . . . . . . . . . . . . . . . . . . . 22
4.2.2.17 evalderderjacobian . . . . . . . . . . . . . . . . . . 23
4.2.2.18 evalderjacobian . . . . . . . . . . . . . . . . . . . . 24
4.2.2.19 evalderjacobianqp . . . . . . . . . . . . . . . . . . . 24
4.2.2.20 evalfit . . . . . . . . . . . . . . . . . . . . . . . . 25
4.2.2.21 evalfitp . . . . . . . . . . . . . . . . . . . . . . . 26
4.2.2.22 evaljacob_jacob . . . . . . . . . . . . . . . . . . . . 27
4.2.2.23 evaljacobderjacobianqp . . . . . . . . . . . . . . . . 28
4.2.2.24 evaljacobian . . . . . . . . . . . . . . . . . . . . . 28
4.2.2.25 evaljacobT_jacob . . . . . . . . . . . . . . . . . . . 29
4.2.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 29
4.2.3.1 Constr_dot1 . . . . . . . . . . . . . . . . . . . . . 29
4.2.3.2 Constr_dot1GB . . . . . . . . . . . . . . . . . . . . 29
4.2.3.3 Constr_Drive_Dist . . . . . . . . . . . . . . . . . . . 29
4.2.3.4 Constr_Drive_DistGB . . . . . . . . . . . . . . . . . 29
4.2.3.5 Constr_Drive_rgEul . . . . . . . . . . . . . . . . . . 30
4.2.3.6 Constr_RevJoint . . . . . . . . . . . . . . . . . . . 30
4.2.3.7 Constr_RevJointGB . . . . . . . . . . . . . . . . . . 30
4.2.3.8 Constr_SpheJoint . . . . . . . . . . . . . . . . . . . 30
4.2.3.9 Constr_SpheJointGB . . . . . . . . . . . . . . . . . 30
4.2.3.10 Constr_TransJoint . . . . . . . . . . . . . . . . . . . 30
4.2.3.11 Constr_TransJointGB . . . . . . . . . . . . . . . . . 30
4.2.3.12 Constr_UnitEulParam . . . . . . . . . . . . . . . . . 30

4.2.3.13 nConstr_dot1 . . . . . . . . . . . . . . . . . . . . . 30
4.2.3.14 nConstr_dot1GB . . . . . . . . . . . . . . . . . . . 30
4.2.3.15 nConstr_Drive_dist . . . . . . . . . . . . . . . . . . 30
4.2.3.16 nConstr_Drive_distGB . . . . . . . . . . . . . . . . 30
4.2.3.17 nConstr_Drive_rgEul . . . . . . . . . . . . . . . . . 30
4.2.3.18 nConstr_RevJoint . . . . . . . . . . . . . . . . . . 30
4.2.3.19 nConstr_RevJointGB . . . . . . . . . . . . . . . . . 30
4.2.3.20 nConstr_SpheJoint . . . . . . . . . . . . . . . . . . 30
4.2.3.21 nConstr_SpheJointGB . . . . . . . . . . . . . . . . 30
4.2.3.22 nConstr_TransJoint . . . . . . . . . . . . . . . . . 30
4.2.3.23 nConstr_TransJointGB . . . . . . . . . . . . . . . . 30
4.2.3.24 nConstr_UnitEulParam . . . . . . . . . . . . . . . . 30
4.3 d2Jacobdt2 Module Reference . . . . . . . . . . . . . . . . . . . . 31
4.3.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 32
4.3.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 32
4.3.2.1 d2Jacobdt2_Setup . . . . . . . . . . . . . . . . . . 32
4.3.2.2 ddj_dot1 . . . . . . . . . . . . . . . . . . . . . . 32
4.3.2.3 ddj_dot1GB . . . . . . . . . . . . . . . . . . . . . 33
4.3.2.4 ddj_Drive_dist . . . . . . . . . . . . . . . . . . . . 34
4.3.2.5 ddj_Drive_distGB . . . . . . . . . . . . . . . . . . 34
4.3.2.6 ddj_revolute . . . . . . . . . . . . . . . . . . . . . 35
4.3.2.7 ddj_revoluteGB . . . . . . . . . . . . . . . . . . . . 36
4.3.2.8 ddj_spherical . . . . . . . . . . . . . . . . . . . . 37
4.3.2.9 ddj_sphericalGB . . . . . . . . . . . . . . . . . . . 37
4.3.2.10 ddj_trans . . . . . . . . . . . . . . . . . . . . . . 38
4.3.2.11 ddj_transGB . . . . . . . . . . . . . . . . . . . . . 39
4.3.2.12 ddj_UnitEulParam . . . . . . . . . . . . . . . . . . 40
4.3.2.13 deallocFiqpp . . . . . . . . . . . . . . . . . . . . 40
4.3.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 40
4.3.3.1 Fiqpp . . . . . . . . . . . . . . . . . . . . . . . . 40
4.3.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . . 40
4.4 DERIVED_TYPES Module Reference . . . . . . . . . . . . . . . . . . 40
4.4.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 42
4.4.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 42

4.4.2.1 indeu . . . 42
4.4.2.2 indeu0 . . . 42
4.4.2.3 indeu1 . . . 42
4.4.2.4 indeu2 . . . 42
4.4.2.5 indeu3 . . . 42
4.4.2.6 indre . . . 42
4.4.2.7 indrg . . . 42
4.4.2.8 indrgx . . . 42
4.4.2.9 indrgy . . . 42
4.4.2.10 indrgz . . . 42
4.5 dJacobdt Module Reference . . . 43
4.5.1 Detailed Description . . . 43
4.5.2 Function/Subroutine Documentation . . . 44
4.5.2.1 deallocFiqp . . . 44
4.5.2.2 dj_dot1 . . . 44
4.5.2.3 dj_dot1GB . . . 44
4.5.2.4 dj_Drive_dist . . . 45
4.5.2.5 dj_Drive_distGB . . . 46
4.5.2.6 dj_revolute . . . 46
4.5.2.7 dj_revoluteGB . . . 47
4.5.2.8 dj_spherical . . . 48
4.5.2.9 dj_sphericalGB . . . 48
4.5.2.10 dj_trans . . . 49
4.5.2.11 dj_transGB . . . 50
4.5.2.12 dj_UnitEulParam . . . 51
4.5.2.13 dJacobdt_Setup . . . 51
4.5.3 Variable Documentation . . . 52
4.5.3.1 Fiqp . . . 52
4.5.3.2 PROTECTED . . . 52
4.6 djacobdt_qp Module Reference . . . 52
4.6.1 Detailed Description . . . 53
4.6.2 Function/Subroutine Documentation . . . 53
4.6.2.1 d_dot1 . . . 53
4.6.2.2 d_dot1GB . . . 54

4.6.2.3 d_Drive_dist . . . . . . . . . . . . . . . . . . . . . 55
4.6.2.4 d_Drive_distGB . . . . . . . . . . . . . . . . . . . 55
4.6.2.5 d_revolute . . . . . . . . . . . . . . . . . . . . . 56
4.6.2.6 d_revoluteGB . . . . . . . . . . . . . . . . . . . . 57
4.6.2.7 d_spherical . . . . . . . . . . . . . . . . . . . . . 58
4.6.2.8 d_sphericalGB . . . . . . . . . . . . . . . . . . . 58
4.6.2.9 d_trans . . . . . . . . . . . . . . . . . . . . . . 59
4.6.2.10 d_transGB . . . . . . . . . . . . . . . . . . . . . 60
4.6.2.11 d_UnitEulParam . . . . . . . . . . . . . . . . . . . 61
4.6.2.12 deallocfiqpqp . . . . . . . . . . . . . . . . . . . 61
4.6.2.13 deallocfit . . . . . . . . . . . . . . . . . . . . . 61
4.6.2.14 djacobdt_qp_Setup . . . . . . . . . . . . . . . . . 62
4.6.2.15 dt_Drive_dist . . . . . . . . . . . . . . . . . . . 62
4.6.2.16 dt_Drive_rgEul . . . . . . . . . . . . . . . . . . . 62
4.6.2.17 dtp_Drive_dist . . . . . . . . . . . . . . . . . . . 62
4.6.2.18 dtp_Drive_rgEul . . . . . . . . . . . . . . . . . . 63
4.6.3 Variable Documentation . . . . . . . . . . . . . . . . . . 63
4.6.3.1 fiqpqp . . . . . . . . . . . . . . . . . . . . . . . 63
4.6.3.2 fit . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.6.3.3 fitp . . . . . . . . . . . . . . . . . . . . . . . . 63
4.6.3.4 PROTECTED . . . . . . . . . . . . . . . . . . . . . 63
4.7 forces Module Reference . . . . . . . . . . . . . . . . . . . . . . 63
4.7.1 Function/Subroutine Documentation . . . . . . . . . . . . . 64
4.7.1.1 force . . . . . . . . . . . . . . . . . . . . . . . . 64
4.7.1.2 TSDA . . . . . . . . . . . . . . . . . . . . . . . . 64
4.7.1.3 TSDA . . . . . . . . . . . . . . . . . . . . . . . . 66
4.7.1.4 TSDA_q . . . . . . . . . . . . . . . . . . . . . . . 66
4.7.1.5 TSDA_q . . . . . . . . . . . . . . . . . . . . . . . 67
4.7.1.6 TSDA_qp . . . . . . . . . . . . . . . . . . . . . . 70
4.7.1.7 TSDA_qp . . . . . . . . . . . . . . . . . . . . . . 71
4.8 formulation_Dynamics Module Reference . . . . . . . . . . . . . . . 72
4.8.1 Detailed Description . . . . . . . . . . . . . . . . . . . 72
4.8.2 Function/Subroutine Documentation . . . . . . . . . . . . . 73
4.8.2.1 Acceleration_penalty . . . . . . . . . . . . . . . . 73

4.8.2.2 Penalty_fun . . . . . . . . . . . . . . . . . . . . . 75
4.8.2.3 Penalty_Tang . . . . . . . . . . . . . . . . . . . . . 76
4.9 formulation_Kinematics Module Reference . . . . . . . . . . . . . . . 77
4.9.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 77
4.9.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 77
4.9.2.1 acceleration_kinematics . . . . . . . . . . . . . . . 77
4.9.2.2 position_kinematics . . . . . . . . . . . . . . . . . 78
4.9.2.3 velocity_kinematics . . . . . . . . . . . . . . . . . 79
4.10 formulation_Sensitivity Module Reference . . . . . . . . . . . . . . . 79
4.10.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 80
4.10.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 80
4.10.2.1 Penalty_Jacp . . . . . . . . . . . . . . . . . . . . . 80
4.11 formulations Module Reference . . . . . . . . . . . . . . . . . . . . 80
4.11.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 81
4.11.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 81
4.11.2.1 acceleration_dynamics . . . . . . . . . . . . . . . . 81
4.11.2.2 integration_dynamics . . . . . . . . . . . . . . . . 82
4.11.2.3 integration_sensitivity . . . . . . . . . . . . . . . . 84
4.11.2.4 Model_Setup . . . . . . . . . . . . . . . . . . . . . 85
4.12 generalized_forces Module Reference . . . . . . . . . . . . . . . . . 86
4.12.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 87
4.12.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 87
4.12.2.1 ADDforce_TSDA . . . . . . . . . . . . . . . . . . . 87
4.12.2.2 evalgenforces . . . . . . . . . . . . . . . . . . . . 87
4.12.2.3 evalgenstiffdamp . . . . . . . . . . . . . . . . . . . 88
4.12.2.4 evalprimitiveforces . . . . . . . . . . . . . . . . . . 90
4.12.2.5 evalprimstiffdamp . . . . . . . . . . . . . . . . . . 90
4.12.2.6 generalized_forces_Setup . . . . . . . . . . . . . . 92
4.12.2.7 genForce1body . . . . . . . . . . . . . . . . . . . . 92
4.12.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 92
4.12.3.1 Cgen . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.12.3.2 Cgrav . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.12.3.3 force_TSDA . . . . . . . . . . . . . . . . . . . . . . 92
4.12.3.4 Kgen . . . . . . . . . . . . . . . . . . . . . . . . . 92

4.12.3.5 Kgrav . . . . . . . . . . . . . . . . . . . . . . . . 92
4.12.3.6 nforce_TSDA . . . . . . . . . . . . . . . . . . . . 92
4.12.3.7 PROTECTED . . . . . . . . . . . . . . . . . . . 93
4.12.3.8 Qgen . . . . . . . . . . . . . . . . . . . . . . . . 93
4.12.3.9 Qgrav . . . . . . . . . . . . . . . . . . . . . . . 93
4.13 Jacob Module Reference . . . . . . . . . . . . . . . . . . . . . . 93
4.13.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 94
4.13.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 94
4.13.2.1 deallocFiq . . . . . . . . . . . . . . . . . . . . . 94
4.13.2.2 j_dot1 . . . . . . . . . . . . . . . . . . . . . . . 94
4.13.2.3 j_dot1GB . . . . . . . . . . . . . . . . . . . . . . 94
4.13.2.4 j_Drive_dist . . . . . . . . . . . . . . . . . . . . 95
4.13.2.5 j_Drive_distGB . . . . . . . . . . . . . . . . . . . 96
4.13.2.6 j_Drive_rgEul . . . . . . . . . . . . . . . . . . . . 97
4.13.2.7 j_revolute . . . . . . . . . . . . . . . . . . . . . 97
4.13.2.8 j_revoluteGB . . . . . . . . . . . . . . . . . . . . 98
4.13.2.9 j_spherical . . . . . . . . . . . . . . . . . . . . . 98
4.13.2.10 j_sphericalGB . . . . . . . . . . . . . . . . . . . 99
4.13.2.11 j_trans . . . . . . . . . . . . . . . . . . . . . . 99
4.13.2.12 j_transGB . . . . . . . . . . . . . . . . . . . . . 100
4.13.2.13 j_UnitEulParam . . . . . . . . . . . . . . . . . . 101
4.13.2.14 Jacob_Setup . . . . . . . . . . . . . . . . . . . . 101
4.13.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 101
4.13.3.1 Fiq . . . . . . . . . . . . . . . . . . . . . . . . . 101
4.13.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . 102
4.14 jacob_djacobdt_qp Module Reference . . . . . . . . . . . . . . . . 102
4.14.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 102
4.14.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 102
4.14.2.1 deallocFiqpqpq . . . . . . . . . . . . . . . . . . . 102
4.14.2.2 dq_dot1 . . . . . . . . . . . . . . . . . . . . . . . 102
4.14.2.3 dq_Drive_dist . . . . . . . . . . . . . . . . . . . . 105
4.14.2.4 dq_Drive_distGB . . . . . . . . . . . . . . . . . . 107
4.14.2.5 dq_revolute . . . . . . . . . . . . . . . . . . . . . 109
4.14.2.6 dq_trans . . . . . . . . . . . . . . . . . . . . . . 111

4.14.2.7 dq_transGB . . . . . . . . . . . . . . . . . . . . . 113
4.14.2.8 jacob_djacobdt_qp_Setup . . . . . . . . . . . . . . 115
4.14.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 115
4.14.3.1 Fiqpqpq . . . . . . . . . . . . . . . . . . . . . . 115
4.14.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . 115
4.15 jacob_jacob Module Reference . . . . . . . . . . . . . . . . . . . . . 115
4.15.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 116
4.15.2 Function/Subroutine Documentation . . . . . . . . . . . . . . . 116
4.15.2.1 deallocFiqqlb . . . . . . . . . . . . . . . . . . . . 116
4.15.2.2 jacob_jacob_Setup . . . . . . . . . . . . . . . . . 116
4.15.2.3 jjlb_dot1 . . . . . . . . . . . . . . . . . . . . . . 116
4.15.2.4 jjlb_dot1GB . . . . . . . . . . . . . . . . . . . . . 118
4.15.2.5 jjlb_Drive_dist . . . . . . . . . . . . . . . . . . . 118
4.15.2.6 jjlb_Drive_distGB . . . . . . . . . . . . . . . . . . 120
4.15.2.7 jjlb_revolute . . . . . . . . . . . . . . . . . . . . . 121
4.15.2.8 jjlb_revoluteGB . . . . . . . . . . . . . . . . . . . 123
4.15.2.9 jjlb_spherical . . . . . . . . . . . . . . . . . . . . 123
4.15.2.10 jjlb_sphericalGB . . . . . . . . . . . . . . . . . . 124
4.15.2.11 jjlb_trans . . . . . . . . . . . . . . . . . . . . . . 125
4.15.2.12 jjlb_transGB . . . . . . . . . . . . . . . . . . . . 127
4.15.2.13 jjlb_UnitEulParam . . . . . . . . . . . . . . . . . . 128
4.15.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 129
4.15.3.1 Fiqqlb . . . . . . . . . . . . . . . . . . . . . . . 129
4.15.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . 129
4.16 jacobT_jacob Module Reference . . . . . . . . . . . . . . . . . . . . 129
4.16.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 130
4.16.2 Function/Subroutine Documentation . . . . . . . . . . . . . . . 130
4.16.2.1 deallocFiqqtlb . . . . . . . . . . . . . . . . . . . 130
4.16.2.2 jacobT_jacob_Setup . . . . . . . . . . . . . . . . . 130
4.16.2.3 jjtlb_dot1 . . . . . . . . . . . . . . . . . . . . . . 130
4.16.2.4 jjtlb_dot1GB . . . . . . . . . . . . . . . . . . . . 131
4.16.2.5 jjtlb_Drive_dist . . . . . . . . . . . . . . . . . . . 132
4.16.2.6 jjtlb_Drive_distGB . . . . . . . . . . . . . . . . . . 133
4.16.2.7 jjtlb_revolute . . . . . . . . . . . . . . . . . . . . 135

4.16.2.8 jjtlb_revoluteGB . . . . . . . . . . . . . . . . . . . . 136
4.16.2.9 jjtlb_spherical . . . . . . . . . . . . . . . . . . . . 137
4.16.2.10 jjtlb_sphericalGB . . . . . . . . . . . . . . . . . . . 138
4.16.2.11 jjtlb_trans . . . . . . . . . . . . . . . . . . . . . . 139
4.16.2.12 jjtlb_transGB . . . . . . . . . . . . . . . . . . . . 140
4.16.2.13 jjtlb_UnitEulParam . . . . . . . . . . . . . . . . . . 142
4.16.2.14 resetFiqqtlb . . . . . . . . . . . . . . . . . . . . . 142
4.16.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 142
4.16.3.1 Fiqqtlb . . . . . . . . . . . . . . . . . . . . . . . . 142
4.16.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . . 142
4.17 mass_massq Module Reference . . . . . . . . . . . . . . . . . . . . 142
4.17.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 143
4.17.2 Function/Subroutine Documentation . . . . . . . . . . . . . . . 143
4.17.2.1 deallocdMdqV . . . . . . . . . . . . . . . . . . . . 143
4.17.2.2 dMdqV_Setup . . . . . . . . . . . . . . . . . . . . 143
4.17.2.3 Eval_Mass_Matrix . . . . . . . . . . . . . . . . . . 143
4.17.2.4 Eval_Mq . . . . . . . . . . . . . . . . . . . . . . . 143
4.17.2.5 Mq . . . . . . . . . . . . . . . . . . . . . . . . . . 144
4.17.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 145
4.17.3.1 Mqlb . . . . . . . . . . . . . . . . . . . . . . . . . 145
4.17.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . . 145
4.18 math_oper Module Reference . . . . . . . . . . . . . . . . . . . . . 145
4.18.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 148
4.18.2 Function/Subroutine Documentation . . . . . . . . . . . . . . . 148
4.18.2.1 d2MREulerParam_t2 . . . . . . . . . . . . . . . . . 148
4.18.2.2 dMREulerParam_t . . . . . . . . . . . . . . . . . . 148
4.18.2.3 dMREulerParam_t_e1 . . . . . . . . . . . . . . . . . 148
4.18.2.4 dMREulerParam_t_e1d . . . . . . . . . . . . . . . . 149
4.18.2.5 dMREulerParam_t_e2 . . . . . . . . . . . . . . . . . 149
4.18.2.6 dMREulerParam_t_e2d . . . . . . . . . . . . . . . . 149
4.18.2.7 dMREulerParam_t_e3 . . . . . . . . . . . . . . . . . 149
4.18.2.8 dMREulerParam_t_e3d . . . . . . . . . . . . . . . . 150
4.18.2.9 dMREulerParam_t_e4 . . . . . . . . . . . . . . . . . 150
4.18.2.10 dMREulerParam_t_e4d . . . . . . . . . . . . . . . . 150

4.18.2.11 dMREulerParam_v . . . . . . . . . . . . . . . . . . . 150
4.18.2.12 dMREulerParam_v_e1 . . . . . . . . . . . . . . . . . 151
4.18.2.13 dMREulerParam_v_e2 . . . . . . . . . . . . . . . . . 151
4.18.2.14 dMREulerParam_v_e3 . . . . . . . . . . . . . . . . . 151
4.18.2.15 dMREulerParam_v_e4 . . . . . . . . . . . . . . . . . 151
4.18.2.16 GAV . . . . . . . . . . . . . . . . . . . . . . . . 151
4.18.2.17 GAV_e1 . . . . . . . . . . . . . . . . . . . . . . . 152
4.18.2.18 GAV_e2 . . . . . . . . . . . . . . . . . . . . . . . 152
4.18.2.19 GAV_e3 . . . . . . . . . . . . . . . . . . . . . . . 152
4.18.2.20 GAV_e4 . . . . . . . . . . . . . . . . . . . . . . . 152
4.18.2.21 Gmatrix . . . . . . . . . . . . . . . . . . . . . . . 153
4.18.2.22 Gmatrix_e1 . . . . . . . . . . . . . . . . . . . . . 153
4.18.2.23 Gmatrix_e2 . . . . . . . . . . . . . . . . . . . . . 153
4.18.2.24 Gmatrix_e3 . . . . . . . . . . . . . . . . . . . . . 153
4.18.2.25 Gmatrix_e4 . . . . . . . . . . . . . . . . . . . . . 153
4.18.2.26 MREulerParam . . . . . . . . . . . . . . . . . . . . 153
4.18.2.27 MREulerParam_e1 . . . . . . . . . . . . . . . . . . 154
4.18.2.28 MREulerParam_e2 . . . . . . . . . . . . . . . . . . 154
4.18.2.29 MREulerParam_e3 . . . . . . . . . . . . . . . . . . 154
4.18.2.30 MREulerParam_e4 . . . . . . . . . . . . . . . . . . 154
4.18.2.31 norm . . . . . . . . . . . . . . . . . . . . . . . . 154
4.18.2.32 normalize . . . . . . . . . . . . . . . . . . . . . . 155
4.18.2.33 PerpVectors . . . . . . . . . . . . . . . . . . . . . 155
4.18.2.34 pvect . . . . . . . . . . . . . . . . . . . . . . . . 155
4.18.2.35 tensor_product . . . . . . . . . . . . . . . . . . . . 156
4.18.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 156
4.18.3.1 EYE3 . . . . . . . . . . . . . . . . . . . . . . . . . 156
4.18.3.2 EYE4 . . . . . . . . . . . . . . . . . . . . . . . . . 156
4.18.3.3 ZEROS3 . . . . . . . . . . . . . . . . . . . . . . . . 156
4.19 matlab_caller Module Reference . . . . . . . . . . . . . . . . . . . . 156
4.19.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 157
4.19.2 Function/Subroutine Documentation . . . . . . . . . . . . . . . 157
4.19.2.1 MATLAB_CHECKSES . . . . . . . . . . . . . . . . . 157
4.19.2.2 MATLAB_CLOSESES . . . . . . . . . . . . . . . . . 157

4.19.2.3 MATLAB_EVALSTRING . . . . . . . . . . . . . . . . 157
4.19.2.4 MATLAB_GETREAL . . . . . . . . . . . . . . . . . . 158
4.19.2.5 MATLAB_GETREALVECTOR . . . . . . . . . . . . . 158
4.19.2.6 MATLAB_OPENSES . . . . . . . . . . . . . . . . . . 158
4.19.2.7 MATLAB_PLOT . . . . . . . . . . . . . . . . . . . . 159
4.19.2.8 MATLAB_PUTINTEGER . . . . . . . . . . . . . . . . 159
4.19.2.9 MATLAB_PUTREALVECTOR . . . . . . . . . . . . . 160
4.20 primitive_forces Module Reference . . . . . . . . . . . . . . . . . . 160
4.20.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 161
4.20.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 161
4.20.2.1 TSDA . . . . . . . . . . . . . . . . . . . . . . . . . 161
4.20.2.2 TSDA_q . . . . . . . . . . . . . . . . . . . . . . . . 162
4.20.2.3 TSDA_qp . . . . . . . . . . . . . . . . . . . . . . . 163
4.21 restric Module Reference . . . . . . . . . . . . . . . . . . . . . . . 163
4.21.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . 165
4.21.2 Function/Subroutine Documentation . . . . . . . . . . . . . . 165
4.21.2.1 deallocfi . . . . . . . . . . . . . . . . . . . . . . . 165
4.21.2.2 r_dot1 . . . . . . . . . . . . . . . . . . . . . . . . 165
4.21.2.3 r_dot1GB . . . . . . . . . . . . . . . . . . . . . . . 165
4.21.2.4 r_Drive_dist . . . . . . . . . . . . . . . . . . . . . 166
4.21.2.5 r_Drive_distGB . . . . . . . . . . . . . . . . . . . 167
4.21.2.6 r_Drive_rgEul . . . . . . . . . . . . . . . . . . . . 167
4.21.2.7 r_revolute . . . . . . . . . . . . . . . . . . . . . . 168
4.21.2.8 r_revoluteGB . . . . . . . . . . . . . . . . . . . . 168
4.21.2.9 r_spherical . . . . . . . . . . . . . . . . . . . . . 169
4.21.2.10 r_sphericalGB . . . . . . . . . . . . . . . . . . . . 170
4.21.2.11 r_trans . . . . . . . . . . . . . . . . . . . . . . . 170
4.21.2.12 r_transGB . . . . . . . . . . . . . . . . . . . . . . 171
4.21.2.13 r_UnitEulParam . . . . . . . . . . . . . . . . . . . 172
4.21.2.14 restric_Setup . . . . . . . . . . . . . . . . . . . . 172
4.21.3 Variable Documentation . . . . . . . . . . . . . . . . . . . . 172
4.21.3.1 fi . . . . . . . . . . . . . . . . . . . . . . . . . . 172
4.21.3.2 PROTECTED . . . . . . . . . . . . . . . . . . . . . 172
4.22 SOLIDS Module Reference . . . . . . . . . . . . . . . . . . . . . . . 172

4.22.1 Detailed Description . . . . . 173
4.22.2 Function/Subroutine Documentation . . . . . 173
4.22.2.1 ADDbody . . . . . 173
4.22.2.2 ADDmassgeom . . . . . 174
4.22.2.3 MatTrans . . . . . 174
4.22.2.4 r . . . . . 175
4.22.2.5 rp . . . . . 175
4.22.3 Variable Documentation . . . . . 176
4.22.3.1 ground . . . . . 176
4.22.3.2 nSOLID . . . . . 176
4.22.3.3 PROTECTED . . . . . 176
4.22.3.4 SOLIDlist . . . . . 176
4.23 STATE Module Reference . . . . . 176
4.23.1 Detailed Description . . . . . 177
4.23.2 Function/Subroutine Documentation . . . . . 178
4.23.2.1 assembleMM . . . . . 178
4.23.2.2 Eul . . . . . 178
4.23.2.3 Eulp . . . . . 179
4.23.2.4 Euls . . . . . 179
4.23.2.5 rg . . . . . 180
4.23.2.6 rgp . . . . . 180
4.23.2.7 rgs . . . . . 181
4.23.2.8 rgx . . . . . 181
4.23.2.9 rgy . . . . . 182
4.23.2.10 rgz . . . . . 182
4.23.2.11 setDOF . . . . . 183
4.23.2.12 STATE_Setup . . . . . 183
4.23.3 Variable Documentation . . . . . 183
4.23.3.1 ace . . . . . 183
4.23.3.2 gdl . . . . . 183
4.23.3.3 lambda . . . . . 184
4.23.3.4 MM . . . . . 184
4.23.3.5 pos . . . . . 184
4.23.3.6 q . . . . . 184

4.23.3.7 qp . . . 184
4.23.3.8 qp_g . . . 184
4.23.3.9 qs . . . 184
4.23.3.10 qs_g . . . 184
4.23.3.11 vel . . . 184
4.23.3.12 zp . . . 184
4.23.3.13 zs . . . 184

**5 Data Type Documentation 185**

5.1 DERIVED_TYPES::callback_AdjInit Interface Reference . . . 185
5.1.1 Constructor & Destructor Documentation . . . 185
5.1.1.1 callback_AdjInit . . . 185
5.2 DERIVED_TYPES::callback_damping Interface Reference . . . 185
5.2.1 Constructor & Destructor Documentation . . . 185
5.2.1.1 callback_damping . . . 185
5.3 DERIVED_TYPES::callback_dgdp Interface Reference . . . 186
5.3.1 Constructor & Destructor Documentation . . . 186
5.3.1.1 callback_dgdp . . . 186
5.4 DERIVED_TYPES::callback_dgdy Interface Reference . . . 186
5.4.1 Constructor & Destructor Documentation . . . 186
5.4.1.1 callback_dgdy . . . 186
5.5 DERIVED_TYPES::callback_forces Interface Reference . . . 186
5.5.1 Constructor & Destructor Documentation . . . 187
5.5.1.1 callback_forces . . . 187
5.6 DERIVED_TYPES::callback_gfun Interface Reference . . . 187
5.6.1 Constructor & Destructor Documentation . . . 187
5.6.1.1 callback_gfun . . . 187
5.7 DERIVED_TYPES::callback_PMbarPrhoVdot Interface Reference . . . 187
5.7.1 Constructor & Destructor Documentation . . . 187
5.7.1.1 callback_PMbarPrhoVdot . . . 187
5.8 DERIVED_TYPES::callback_PQbarPrho Interface Reference . . . 188
5.8.1 Constructor & Destructor Documentation . . . 188
5.8.1.1 callback_PQbarPrho . . . 188
5.9 DERIVED_TYPES::callback_stiffness Interface Reference . . . 188

5.9.1 Constructor & Destructor Documentation . . . . . . . . . . . . . 188
5.9.1.1 callback_stiffness . . . . . . . . . . . . . . . . . . . 188
5.10 DERIVED_TYPES::MATRIXTRANSFORM Type Reference . . . . . . . 188
5.10.1 Member Data Documentation . . . . . . . . . . . . . . . . . . . 188
5.10.1.1 iscalc . . . . . . . . . . . . . . . . . . . . . . . . . 188
5.10.1.2 uvwr . . . . . . . . . . . . . . . . . . . . . . . . . 189
5.11 DERIVED_TYPES::POINT Type Reference . . . . . . . . . . . . . . 189
5.11.1 Member Data Documentation . . . . . . . . . . . . . . . . . . . 189
5.11.1.1 idpt . . . . . . . . . . . . . . . . . . . . . . . . . . 189
5.11.1.2 pt . . . . . . . . . . . . . . . . . . . . . . . . . . . 189
5.11.1.3 rpt . . . . . . . . . . . . . . . . . . . . . . . . . . 189
5.11.1.4 rptp . . . . . . . . . . . . . . . . . . . . . . . . . . 189
5.12 DERIVED_TYPES::SOLID Type Reference . . . . . . . . . . . . . . . 189
5.12.1 Member Data Documentation . . . . . . . . . . . . . . . . . . . 190
5.12.1.1 id . . . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.2 mass . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.3 Mt . . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.4 Mtp . . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.5 npt_loc . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.6 nvc_loc . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.7 nz . . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.8 pt_loc . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.9 rg_loc . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.10 v_lg . . . . . . . . . . . . . . . . . . . . . . . . . 190
5.12.1.11 vc_loc . . . . . . . . . . . . . . . . . . . . . . . . 190
5.13 DERIVED_TYPES::typeConstr_dot1 Type Reference . . . . . . . . . . 190
5.13.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . 191
5.13.2 Member Data Documentation . . . . . . . . . . . . . . . . . . . 191
5.13.2.1 iEul1 . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.13.2.2 iEul2 . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.13.2.3 ir . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.13.2.4 u . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.13.2.5 v . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.14 DERIVED_TYPES::typeconstr_Drive_Dist Type Reference . . . . . . . 191

5.14.1 Detailed Description . . . 191
5.14.2 Member Data Documentation . . . 192
5.14.2.1 i_MOTOR . . . 192
5.14.2.2 iEul1 . . . 192
5.14.2.3 ieul2 . . . 192
5.14.2.4 ir . . . 192
5.14.2.5 irg1 . . . 192
5.14.2.6 irg2 . . . 192
5.14.2.7 pt1 . . . 192
5.14.2.8 pt2 . . . 192
5.15 DERIVED_TYPES::typeconstr_Drive_rgEul Type Reference . . . 192
5.15.1 Detailed Description . . . 192
5.15.2 Member Data Documentation . . . 192
5.15.2.1 i_MOTOR . . . 192
5.15.2.2 ind . . . 192
5.15.2.3 ir . . . 193
5.16 DERIVED_TYPES::typeConstr_RevJoint Type Reference . . . 193
5.16.1 Detailed Description . . . 193
5.16.2 Member Data Documentation . . . 193
5.16.2.1 iEul1 . . . 193
5.16.2.2 iEul2 . . . 193
5.16.2.3 ir . . . 193
5.16.2.4 irg1 . . . 193
5.16.2.5 irg2 . . . 193
5.16.2.6 pt1 . . . 193
5.16.2.7 pt2 . . . 194
5.16.2.8 uPerp1 . . . 194
5.16.2.9 vec1 . . . 194
5.16.2.10 vec2 . . . 194
5.16.2.11 vPerp1 . . . 194
5.17 DERIVED_TYPES::typeConstr_SpheJoint Type Reference . . . 194
5.17.1 Detailed Description . . . 194
5.17.2 Member Data Documentation . . . 194
5.17.2.1 iEul1 . . . 194

5.17.2.2 iEul2 . . . . . . . . . . . . . . . . . . . . . . . . 194
5.17.2.3 ir . . . . . . . . . . . . . . . . . . . . . . . . . . 194
5.17.2.4 irg1 . . . . . . . . . . . . . . . . . . . . . . . . . 194
5.17.2.5 irg2 . . . . . . . . . . . . . . . . . . . . . . . . . 194
5.17.2.6 pt1 . . . . . . . . . . . . . . . . . . . . . . . . . 195
5.17.2.7 pt2 . . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18 DERIVED_TYPES::typeConstr_TransJoint Type Reference . . . . . . . 195
5.18.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 195
5.18.2 Member Data Documentation . . . . . . . . . . . . . . . . . . . 195
5.18.2.1 iEul1 . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18.2.2 iEul2 . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18.2.3 ir . . . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18.2.4 irg1 . . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18.2.5 irg2 . . . . . . . . . . . . . . . . . . . . . . . . . 195
5.18.2.6 pt1 . . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.7 pt2 . . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.8 vec1x . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.9 vec1y . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.10 vec2x . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.11 vec2y . . . . . . . . . . . . . . . . . . . . . . . . 196
5.18.2.12 vec2z . . . . . . . . . . . . . . . . . . . . . . . . 196
5.19 DERIVED_TYPES::typeConstr_UnitEulParam Type Reference . . . . . 196
5.19.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 196
5.19.2 Member Data Documentation . . . . . . . . . . . . . . . . . . . 196
5.19.2.1 iEul . . . . . . . . . . . . . . . . . . . . . . . . . 196
5.19.2.2 ir . . . . . . . . . . . . . . . . . . . . . . . . . . 196
5.20 DERIVED_TYPES::typeforce_TSDA Type Reference . . . . . . . . . . 197
5.20.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . 197
5.20.2 Member Data Documentation . . . . . . . . . . . . . . . . . . . 197
5.20.2.1 body1 . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.2 body2 . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.3 c . . . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.4 F1 . . . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.5 F2 . . . . . . . . . . . . . . . . . . . . . . . . . . 197

5.20.2.6 k . . . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.7 pt1 . . . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.8 pt2 . . . . . . . . . . . . . . . . . . . . . . . . . . 197
5.20.2.9 s0 . . . . . . . . . . . . . . . . . . . . . . . . . . 197

**6 File Documentation 199**

6.1 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/constants.f90 File Reference . . . . . . . . . . . . . . . . . . 199
6.2 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/constraints.f90 File Reference . . . . . . . . . . . . . . . . . 200
6.3 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/d2jacobdt2.f90 File Reference . . . . . . . . . . . . . . . . . 203
6.4 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . 204
6.5 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/djacobdt.f90 File Reference . . . . . . . . . . . . . . . . . . 205
6.6 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/djacobdt_-qp.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 206
6.7 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/forceold.f90 File Reference . . . . . . . . . . . . . . . . . . 207
6.8 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/forces.f90 File Reference . . . . . . . . . . . . . . . . . . . 208
6.9 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-dynamics.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . 208
6.10 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-kinematics.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . 209
6.11 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-sensitivity.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . 209
6.12 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/formulations.f90 File Reference . . . . . . . . . . . . . . . . 210
6.13 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/generalized_-forces.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . 210
6.14 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/jacob.f90 File Reference . . . . . . . . . . . . . . . . . . . . 211
6.15 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_-djacobdt_qp.f90 File Reference . . . . . . . . . . . . . . . . . . . . . 212
6.16 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_-jacob.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 213
6.17 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacobT_-jacob.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 214

6.18 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/lbfgsb/blas.f File Reference . . . . . . . . . . . . . . . . . 215
6.18.1 Function Documentation . . . . . . . . . . . . . . . . . . . . . 215
6.18.1.1 daxpy . . . . . . . . . . . . . . . . . . . . . . . . 215
6.18.1.2 dcopy . . . . . . . . . . . . . . . . . . . . . . . . 216
6.18.1.3 ddot . . . . . . . . . . . . . . . . . . . . . . . . 216
6.18.1.4 dnrm2 . . . . . . . . . . . . . . . . . . . . . . . . 216
6.18.1.5 dscal . . . . . . . . . . . . . . . . . . . . . . . . 216
6.19 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/lbfgsb/lbfgsb.f File Reference . . . . . . . . . . . . . . . . . 216
6.19.1 Function Documentation . . . . . . . . . . . . . . . . . . . . . 217
6.19.1.1 active . . . . . . . . . . . . . . . . . . . . . . . . 217
6.19.1.2 bmv . . . . . . . . . . . . . . . . . . . . . . . . 217
6.19.1.3 cauchy . . . . . . . . . . . . . . . . . . . . . . . . 218
6.19.1.4 cmprlb . . . . . . . . . . . . . . . . . . . . . . . . 219
6.19.1.5 dcsrch . . . . . . . . . . . . . . . . . . . . . . . . 219
6.19.1.6 dcstep . . . . . . . . . . . . . . . . . . . . . . . . 219
6.19.1.7 errclb . . . . . . . . . . . . . . . . . . . . . . . . 219
6.19.1.8 formk . . . . . . . . . . . . . . . . . . . . . . . . 220
6.19.1.9 formt . . . . . . . . . . . . . . . . . . . . . . . . 220
6.19.1.10 freev . . . . . . . . . . . . . . . . . . . . . . . . 220
6.19.1.11 hpsolb . . . . . . . . . . . . . . . . . . . . . . . . 221
6.19.1.12 lnsrlb . . . . . . . . . . . . . . . . . . . . . . . . 221
6.19.1.13 mainlb . . . . . . . . . . . . . . . . . . . . . . . . 222
6.19.1.14 matupd . . . . . . . . . . . . . . . . . . . . . . . . 223
6.19.1.15 prn1lb . . . . . . . . . . . . . . . . . . . . . . . . 223
6.19.1.16 prn2lb . . . . . . . . . . . . . . . . . . . . . . . . 223
6.19.1.17 prn3lb . . . . . . . . . . . . . . . . . . . . . . . . 223
6.19.1.18 projgr . . . . . . . . . . . . . . . . . . . . . . . . 223
6.19.1.19 setulb . . . . . . . . . . . . . . . . . . . . . . . . 224
6.19.1.20 subsm . . . . . . . . . . . . . . . . . . . . . . . . 225
6.20 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/lbfgsb/linpack.f File Reference . . . . . . . . . . . . . . . . . 225
6.20.1 Function Documentation . . . . . . . . . . . . . . . . . . . . . 225
6.20.1.1 dpofa . . . . . . . . . . . . . . . . . . . . . . . . 225

6.20.1.2 dtrsl . . . . . . . . . . . . . . . . . . . . . . . . . 226

6.21 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/lbfgsb/timer.f File Reference . . . . . . . . . . . . . . . . . . . 226

6.21.1 Function Documentation . . . . . . . . . . . . . . . . . . . . 226

6.21.1.1 timer . . . . . . . . . . . . . . . . . . . . . . . . . 226

6.22 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/mass_-massq.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . 226

6.23 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/math_-oper.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 227

6.24 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/matlab_-caller.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 229

6.25 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/primitive_-forces.f90 File Reference . . . . . . . . . . . . . . . . . . . . . . . . 230

6.26 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/restric.f90 File Reference . . . . . . . . . . . . . . . . . . . 230

6.27 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/solids.f90 File Reference . . . . . . . . . . . . . . . . . . . . 232

6.28 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MB-SVT/trunk/state.F90 File Reference . . . . . . . . . . . . . . . . . . . . 233

# Chapter 1

# Modules Index

## 1.1 Modules List

Here is a list of all modules with brief descriptions:


CONSTANTS (Module of solver parameters ) . . . . . . . . . . . . . . . . . . 7
CONSTRAINTS (Module that manages the constraints ) . . . . . . . . . . . . 11
d2Jacobdt2 (Module of second derivatives of the Jacobian. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . 31
DERIVED_TYPES (Module of solver derived types definition and subroutines/-functions to manage the derived types ) . . . . . . . . . . . . . . . . 40
dJacobdt (Module of total derivatives of the Jacobian. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . 43
djacobdt_qp (Module of derivatives of the Jacobian mutiplied by the velocity vector. It's NOT a user module, it's used by the solver ) . . . . . . . 52
forces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
formulation_Dynamics (Dynamic simulation module ) . . . . . . . . . . . . . 72
formulation_Kinematics (Kinematic simulation module ) . . . . . . . . . . . 77
formulation_Sensitivity (Sensitivity analyis module ) . . . . . . . . . . . . . 79
formulations (Module of generic formulations. Contains the generic functions that manage the use of different formulations ) . . . . . . . . . . . . 80
generalized_forces (Generalized forces module ) . . . . . . . . . . . . . . . 86
Jacob (Module of primitive jacobians. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
jacob_djacobdt_qp (Module of $(\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}$. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
jacob_jacob (Module of $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$, which is the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver ) 115
jacobT_jacob (Module of $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$, which is the transpose of the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . . . 129
mass_massq (Module of $\mathbf{M}_{\mathbf{q}}\mathbf{V}$, which is the jacobian of the mass matrix multiplied by a vector. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142

math_oper (Module of non-intrinsic mathematical operations. Contains all the operations necessary for multi body dynamics computations not supported by the Fortran 2003 standard ) . . . . . . . . . . . . . . . . 145
matlab_caller (Managment of sessions of MATLAB engine: This is a part of the matlab_caller module of MBSLIM ) . . . . . . . . . . . . . . . . 156
primitive_forces (Primitive forces module ) . . . . . . . . . . . . . . . . . . . . 160
restric (Module of primitive constraints. It's NOT a user module, it's used by the solver ) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 163
SOLIDS (Solids module that adds and manages the bodies of the system ) . . 172
STATE (Module of solver state variables, subroutines and functions. It creates, manages and updates the state variables of the model ) . . . . . . . . 176

# Chapter 2

# Data Type Index

## 2.1 Class List

Here are the data types with brief descriptions:


DERIVED_TYPES::callback_AdjInit . . . . . . . . . . . . . . . . . . . . . . . 185
DERIVED_TYPES::callback_damping . . . . . . . . . . . . . . . . . . . . . . 185
DERIVED_TYPES::callback_dgdp . . . . . . . . . . . . . . . . . . . . . . . . 186
DERIVED_TYPES::callback_dgdy . . . . . . . . . . . . . . . . . . . . . . . . 186
DERIVED_TYPES::callback_forces . . . . . . . . . . . . . . . . . . . . . . . 186
DERIVED_TYPES::callback_gfun . . . . . . . . . . . . . . . . . . . . . . . . 187
DERIVED_TYPES::callback_PMbarPrhoVdot . . . . . . . . . . . . . . . . . 187
DERIVED_TYPES::callback_PQbarPrho . . . . . . . . . . . . . . . . . . . . 188
DERIVED_TYPES::callback_stiffness . . . . . . . . . . . . . . . . . . . . . . 188
DERIVED_TYPES::MATRIXTRANSFORM . . . . . . . . . . . . . . . . . . . 188
DERIVED_TYPES::POINT . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189
DERIVED_TYPES::SOLID . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189
DERIVED_TYPES::typeConstr_dot1 (Dot-1 constraints ) . . . . . . . . . . . 190
DERIVED_TYPES::typeconstr_Drive_Dist (Driving distance constraints ) . . . 191
DERIVED_TYPES::typeconstr_Drive_rgEul (Driving constraints coordinates ) . 192
DERIVED_TYPES::typeConstr_RevJoint (Revolute joint constraints ) . . . . . 193
DERIVED_TYPES::typeConstr_SpheJoint (Spherical joint constraints ) . . . . 194
DERIVED_TYPES::typeConstr_TransJoint (Translational joint constraints ) . . 195
DERIVED_TYPES::typeConstr_UnitEulParam (Euler parameters constraints ) 196
DERIVED_TYPES::typeforce_TSDA (TSDA forces ) . . . . . . . . . . . . . . 197

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:


D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/constants.f90 199
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/constraints.f90 200
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/d2jacobdt2.f90 203
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/djacobdt.f90 205
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/djacobdt_-qp.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/forceold.f90 207
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/forces.f90 208
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-dynamics.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-kinematics.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . 209
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_-sensitivity.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulations.f90 210
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/generalized_-forces.f90 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob.f90 211

D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_-djacobdt_qp.f90 . . . . 212
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_-jacob.f90 . . . . 213
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacobT_-jacob.f90 . . . . 214
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/mass_-massq.f90 . . . . 226
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/math_-oper.f90 . . . . 227
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/matlab_-caller.f90 . . . . 229
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/primitive_-forces.f90 . . . . 230
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/restric.f90 230
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/solids.f90 232
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/state.F90 233
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk-/lbfgsb/blas.f . . . . 215
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk-/lbfgsb/lbfgsb.f . . . . 216
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk-/lbfgsb/linpack.f . . . . 225
D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk-/lbfgsb/timer.f . . . . 226

# Chapter 4

# Module Documentation

## 4.1 CONSTANTS Module Reference

Module of solver parameters.

### Functions/Subroutines

- subroutine initialize_CONSTANTS (formulation, integrator, time_step, penaltycoef, psicoef, omegacoef, gravity)

  *Inicialization of solver constants and parameters.*
- subroutine initialize_CALLBACKS (forces, stiffness, damping, PQbarPrho, PMbarPrhoVdot, dgdy, dgdp, AdjInit, gfun)

  *Initialization of solver callbacks: the user provides the subtoutines that the solver calls if necessary. It needs a previous call to constants::initialize_CONSTANTS.*
- subroutine setDIM (newDIM)
- subroutine setNRT (newNRT)
- subroutine setNMT (newNMT)
- subroutine setNIN (newNIN)

### Variables

- REAL(8), dimension(3), pointer PROTECTED
- REAL(8), dimension(3) g = (/0.d0,0.d0,-9.81d0/)
- REAL(8) dt = 1.d-2
- REAL(8) alfa = 1.d9
- REAL(8) psi = 1.d0
- REAL(8) omega = 10.d0
- REAL(8) tolNRppos = 1.d-10
- REAL(8) pivgdlval = 1.d15
- INTEGER maxiteppos = 1000
- INTEGER, parameter Dynamics = 1

- INTEGER, parameter Kinematics = 2
- INTEGER, parameter Sensitivity_ADJ = 3
- INTEGER, parameter Sensitivity_TLM = 4
- INTEGER, parameter E_RK = 1
- INTEGER, parameter E_RK2 = 2
- INTEGER, parameter I_RK = 3
- INTEGER, parameter I_RK_ADJ = 4
- INTEGER, parameter I_RK_TLM = 5
- INTEGER SWFORM = Dynamics
- INTEGER SWINT = E_RK
- INTEGER DIM = 0
- INTEGER NRT = 0
- INTEGER NIN = 0
- INTEGER NMT = 0
- PROCEDURE(callback_forces), pointer pforces_user
- PROCEDURE(callback_stiffness), pointer pstiffness_user
- PROCEDURE(callback_damping), pointer pdamping_user
- PROCEDURE(callback_PQbarPrho), pointer pqro_user
- PROCEDURE(callback_PMbarPrhoVdot), pointer pmpv_user
- PROCEDURE(callback_dgdy), pointer pdgdy_user
- PROCEDURE(callback_dgdp), pointer pdgdp_user
- PROCEDURE(callback_AdjInit), pointer padjinit_user
- PROCEDURE(callback_gfun), pointer pgfun_user

### 4.1.1 Detailed Description

Module of solver parameters.

### 4.1.2 Function/Subroutine Documentation

#### 4.1.2.1 subroutine CONSTANTS::initialize_CALLBACKS ( PROCEDURE(callback_forces),optional *forces*, PROCEDURE(callback_stiffness),optional *stiffness*, PROCEDURE(callback_damping),optional *damping*, PROCEDURE(callback_PQbarPrho),optional *PQbarPrho*, PROCEDURE(callback_PMbarPrhoVdot),optional *PMbarPrhoVdot*, PROCEDURE(callback_dgdy),optional *dgdy*, PROCEDURE(callback_dgdp),optional *dgdp*, PROCEDURE(callback_AdjInit),optional *AdjInit*, PROCEDURE(callback_gfun),optional *gfun* )

Initialization of solver callbacks: the user provides the subtoutines that the solver calls if necessary. It needs a previous call to constants::initialize_CONSTANTS.

**Parameters**

| | |
|---|---|
| *forces* | pointer to the subrutine of user forces. If it is not provided, MBSVT understands that user forces are not present and no call is performed. |
| *PQbarPrho* | $\bar{Q}_\rho$. |
| *PMbarPrhoVdot* | $\bar{M}_\rho \dot{v}$. |

#### 4.1.2.2 subroutine CONSTANTS::initialize_CONSTANTS ( INTEGER,intent(in),optional *formulation,* INTEGER,intent(in),optional *integrator,* REAL(8),intent(in),optional *time_step,* REAL(8),intent(in),optional *penaltycoef,* REAL(8),intent(in),optional *psicoef,* REAL(8),intent(in),optional *omegacoef,* REAL(8),dimension(3),intent(in),optional *gravity* )

Inicialization of solver constants and parameters.

**Parameters**

| | |
|---|---|
| *formulation* | there is only constants::Kinematics and constants::Dynamics. |
| *integrator* | numerical integrator for the dynamics. MBSVT uses FATODE integrators. The supported integrators are ERK so far. |
| *time_step* | paso de tiempo. |
| *penalty-coef,psicoef,o* | coeficientes para los términos de penalización. |
| *gravity* | aceleración de la gravedad (vector). |

#### 4.1.2.3 subroutine CONSTANTS::setDIM ( INTEGER,intent(in) *newDIM* )

#### 4.1.2.4 subroutine CONSTANTS::setNIN ( INTEGER,intent(in) *newNIN* )

#### 4.1.2.5 subroutine CONSTANTS::setNMT ( INTEGER,intent(in) *newNMT* )

#### 4.1.2.6 subroutine CONSTANTS::setNRT ( INTEGER,intent(in) *newNRT* )

### 4.1.3 Variable Documentation

#### 4.1.3.1 REAL(8) CONSTANTS::alfa = 1.d9

#### 4.1.3.2 INTEGER CONSTANTS::DIM = 0

#### 4.1.3.3 REAL(8) CONSTANTS::dt = 1.d-2

#### 4.1.3.4 INTEGER,parameter CONSTANTS::Dynamics = 1

#### 4.1.3.5 INTEGER,parameter CONSTANTS::E_RK = 1

#### 4.1.3.6 INTEGER,parameter CONSTANTS::E_RK2 = 2

#### 4.1.3.7 REAL(8),dimension(3) CONSTANTS::g = (/0.d0,0.d0,-9.81d0/)

#### 4.1.3.8 INTEGER,parameter CONSTANTS::I_RK = 3

#### 4.1.3.9 INTEGER,parameter CONSTANTS::I_RK_ADJ = 4

#### 4.1.3.10 INTEGER,parameter CONSTANTS::I_RK_TLM = 5

#### 4.1.3.11 INTEGER,parameter CONSTANTS::Kinematics = 2

#### 4.1.3.12 INTEGER CONSTANTS::maxiteppos = 1000

#### 4.1.3.13 INTEGER CONSTANTS::NIN = 0

#### 4.1.3.14 INTEGER CONSTANTS::NMT = 0

#### 4.1.3.15 INTEGER CONSTANTS::NRT = 0

#### 4.1.3.16 REAL(8) CONSTANTS::omega = 10.d0

#### 4.1.3.17 PROCEDURE(callback_AdjInit),pointer CONSTANTS::padjinit_user

#### 4.1.3.18 PROCEDURE(callback_damping),pointer CONSTANTS::pdamping_user

#### 4.1.3.19 PROCEDURE(callback_dgdp),pointer CONSTANTS::pdgdp_user

#### 4.1.3.20 PROCEDURE(callback_dgdy),pointer CONSTANTS::pdgdy_user

#### 4.1.3.21 PROCEDURE(callback_forces),pointer CONSTANTS::pforces_user

#### 4.1.3.22 PROCEDURE(callback_gfun),pointer CONSTANTS::pgfun_user

#### 4.1.3.23 REAL(8) CONSTANTS::pivgdlval = 1.d15

#### 4.1.3.24 PROCEDURE(callback_PMbarPrhoVdot),pointer CONSTANTS::pmpv_user

#### 4.1.3.25 PROCEDURE(callback_PQbarPrho),pointer CONSTANTS::pqro_user

#### 4.1.3.26 PROCEDURE(callback_gfun),dimension(3),pointer CONSTANTS::PROTECTED

#### 4.1.3.27 REAL(8) CONSTANTS::psi = 1.d0

#### 4.1.3.28 PROCEDURE(callback_stiffness),pointer CONSTANTS::pstiffness_user

#### 4.1.3.29 INTEGER,parameter CONSTANTS::Sensitivity_ADJ = 3

#### 4.1.3.30 INTEGER,parameter CONSTANTS::Sensitivity_TLM = 4

#### 4.1.3.31 INTEGER CONSTANTS::SWFORM = Dynamics

#### 4.1.3.32 INTEGER CONSTANTS::SWINT = E_RK

#### 4.1.3.33 REAL(8) CONSTANTS::tolNRppos = 1.d-10

## 4.2 CONSTRAINTS Module Reference

Module that manages the constraints.

### Functions/Subroutines

- subroutine, public ADDConstr_UnitEulParam (body)

  *Adds an unitary vector constraint to the Euler parameters of one body. It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public ADDConstr_dot1 (body1, body2, vector1, vector2)

  *Adds a dot-1 joint between two bodies or between the ground and one body.*
- subroutine, public ADDConstr_SpheJoint (body1, body2, point1, point2)

  *Adds a spehrcal joint between two bodies or between the ground and one body.*
- subroutine, public ADDConstr_RevJoint (body1, body2, point1, point2, vect1, vect2)

  *Adds a revolute joint between two bodies or between the ground and one body.*
- subroutine, public ADDConstr_TransJoint (body1, body2, point1, point2, vect1x, vect1y, vect2x, vect2y)

  *Adds a translational joint between two bodies or between the ground and one body.*
- subroutine, public ADDConstr_Drive_rgx (body, i_MOTOR)

  *Adds a driving constraint to the x-coordinate of the CDM of a body.*
- subroutine, public ADDConstr_Drive_rgy (body, i_MOTOR)

  *Adds a driving constraint to the y-coordinate of the CDM of a body.*
- subroutine, public ADDConstr_Drive_rgz (body, i_MOTOR)

  *Adds a driving constraint to the z-coordinate of the CDM of a body.*
- subroutine, public ADDConstr_Drive_eu0 (body, i_MOTOR)

  *Adds a driving constraint to the 1st Euler parameter of a body.*
- subroutine, public ADDConstr_Drive_eu1 (body, i_MOTOR)

  *Adds a driving constraint to the 2nd Euler parameter of a body.*
- subroutine, public ADDConstr_Drive_eu2 (body, i_MOTOR)

  *Adds a driving constraint to the 3rd Euler parameter of a body.*
- subroutine, public ADDConstr_Drive_eu3 (body, i_MOTOR)

  *Adds a driving constraint to the 4rd Euler parameter of a body.*
- subroutine ADDConstr_Drive_rgEul (ind, i_MOTOR)

  *Adds a driving constraint to any body coordinate. It's NOT a user function, it's intended to be called by other user constraints.*
- subroutine, public ADDConstr_Drive_dist (body1, body2, point1, point2, i_MOTOR)

  *Adds a driving constraint to a distance between two points in two different bodies.*
- subroutine, public evalconstraints

  *Subroutine that evaluates the constraints vector ( $\mathbf{\Phi}$). It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public evaljacobian

*Subroutine that evaluates the jacobian of the constraints vector ( $\mathbf{\Phi_q}$). It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public evalderjacobianqp

  *Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}}$. It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public evalderjacobian

  *Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_\mathbf{q}$. It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public evalderderjacobian

  *Subroutine that evaluates the term $\ddot{\mathbf{\Phi}}_\mathbf{q}$. It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public evaljacobderjacobianqp
- subroutine, public evalfit
- subroutine, public evalfitp
- subroutine, public evaljacob_jacob (lb)
- subroutine, public evaljacobT_jacob (lb)
- subroutine, public CONSTRAINTS_Setup

## Variables

- INTEGER nConstr_UnitEulParam = 0
- INTEGER nConstr_dot1GB = 0
- INTEGER nConstr_dot1 = 0
- INTEGER nConstr_SpheJointGB = 0
- INTEGER nConstr_SpheJoint = 0
- INTEGER nConstr_RevJointGB = 0
- INTEGER nConstr_RevJoint = 0
- INTEGER nConstr_TransJointGB = 0
- INTEGER nConstr_TransJoint = 0
- INTEGER nConstr_Drive_rgEul = 0
- INTEGER nConstr_Drive_distGB = 0
- INTEGER nConstr_Drive_dist = 0
- TYPE(typeConstr_UnitEulParam), dimension(:), allocatable Constr_UnitEulParam
- TYPE(typeConstr_dot1), dimension(:), allocatable Constr_dot1GB
- TYPE(typeConstr_dot1), dimension(:), allocatable Constr_dot1
- TYPE(typeConstr_SpheJoint), dimension(:), allocatable Constr_SpheJointGB
- TYPE(typeConstr_SpheJoint), dimension(:), allocatable Constr_SpheJoint
- TYPE(typeConstr_RevJoint), dimension(:), allocatable Constr_RevJointGB
- TYPE(typeConstr_RevJoint), dimension(:), allocatable Constr_RevJoint
- TYPE(typeConstr_TransJoint), dimension(:), allocatable Constr_TransJointGB
- TYPE(typeConstr_TransJoint), dimension(:), allocatable Constr_TransJoint
- TYPE(typeconstr_Drive_rgEul), dimension(:), allocatable Constr_Drive_rgEul
- TYPE(typeconstr_Drive_Dist), dimension(:), allocatable Constr_Drive_DistGB
- TYPE(typeconstr_Drive_Dist), dimension(:), allocatable Constr_Drive_Dist

### 4.2.1 Detailed Description

Module that manages the constraints. This module:

1)Authomatically creates the rigid body constraints, associated to the redundant Euler parameters.

2)Add joints to the model (requested by the user), translating them into primitive constraints for each type of joint.

3)Manages the evaluation of the constraints vector, its jacobian matrix and all its associated derivatives.

### 4.2.2 Function/Subroutine Documentation

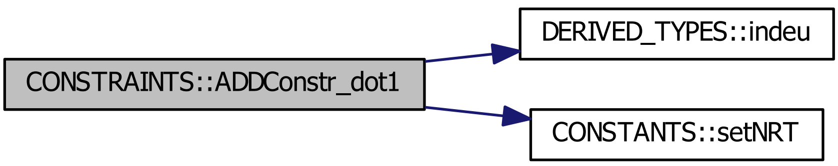
#### 4.2.2.1 subroutine,public CONSTRAINTS::ADDConstr_dot1 ( INTEGER,intent(in) *body1*, INTEGER,intent(in) *body2*, real(8),dimension(3),intent(in) *vector1*, real(8),dimension(3),intent(in) *vector2* )

Adds a dot-1 joint between two bodies or between the ground and one body.

**Parameters**

| | |
|---|---|
| *body1* | first body involved. It can be "SOLIDS::ground" if the second body is attached to the ground. |
| *body2* | second body involved. It cannot be the ground. |
| *vector1* | vector in the first body/ground in the body reference frame. |
| *vector2* | vector in the second body in the body reference frame. |

Here is the call graph for this function:



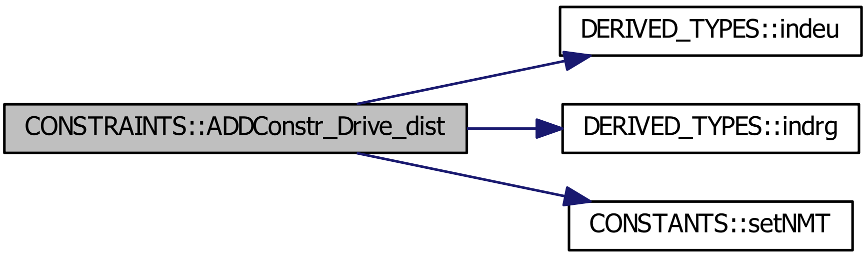
#### 4.2.2.2 subroutine,public CONSTRAINTS::ADDConstr_Drive_dist ( INTEGER,intent(in) *body1,* INTEGER,intent(in) *body2,* REAL(8),dimension(3),intent(in) *point1,* REAL(8),dimension(3),intent(in) *point2,* INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to a distance between two points in two different bodies.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_-MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable eu3. |

Here is the call graph for this function:



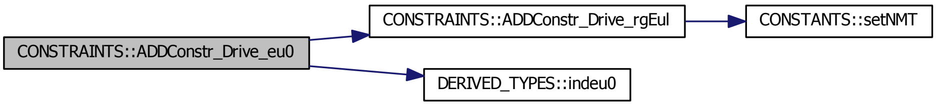

#### 4.2.2.3 subroutine,public CONSTRAINTS::ADDConstr_Drive_eu0 ( INTEGER,intent(in) *body,* INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the 1st Euler parameter of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_-MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable eu0. |

Here is the call graph for this function:



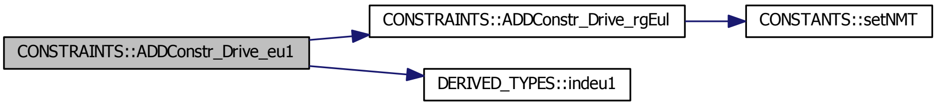

#### 4.2.2.4 subroutine,public CONSTRAINTS::ADDConstr_Drive_eu1 ( INTEGER,intent(in) *body*, INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the 2nd Euler parameter of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable eu1. |

Here is the call graph for this function:




#### 4.2.2.5 subroutine,public CONSTRAINTS::ADDConstr_Drive_eu2 ( INTEGER,intent(in) *body*, INTEGER,intent(in) *i_MOTOR* )

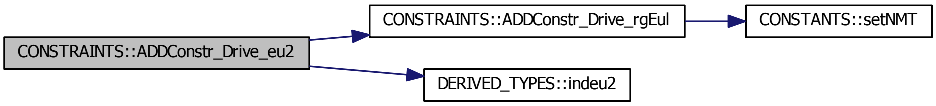
Adds a driving constraint to the 3rd Euler parameter of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

| | |
|---|---|
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable eu2. |

Here is the call graph for this function:



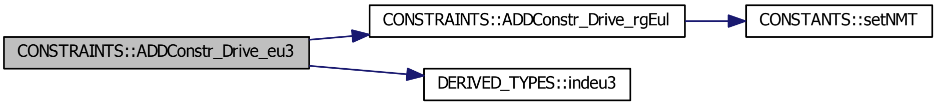

#### 4.2.2.6 subroutine,public CONSTRAINTS::ADDConstr_Drive_eu3 ( INTEGER,intent(in) *body*, INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the 4rd Euler parameter of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable eu3. |

Here is the call graph for this function:



#### 4.2.2.7 subroutine CONSTRAINTS::ADDConstr_Drive_rgEul ( INTEGER,intent(in) *ind*, INTEGER,intent(in) *i_MOTOR* ) [private]

Adds a driving constraint to any body coordinate. It's NOT a user function, it's intended to be called by other user constraints.

**Parameters**

| | |
|---|---|
| *ind* | index of the coordinate involved. |
| *i_MOTOR* | index of the kinematic actuator. |

Here is the call graph for this function:

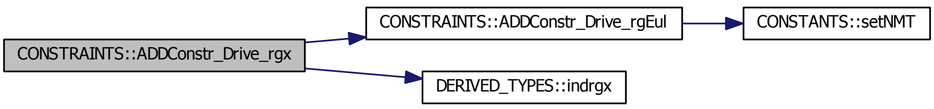

#### 4.2.2.8 subroutine,public CONSTRAINTS::ADDConstr_Drive_rgx ( INTEGER,intent(in) *body*, INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the x-coordinate of the CDM of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_-MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable rgx. |

Here is the call graph for this function:



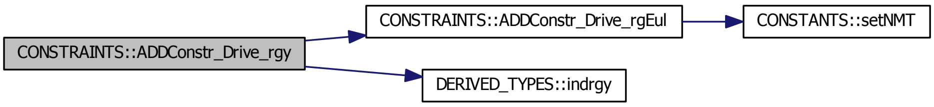
#### 4.2.2.9 subroutine,public CONSTRAINTS::ADDConstr_Drive_rgy ( INTEGER,intent(in) *body,* INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the y-coordinate of the CDM of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_-MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable rgy. |

Here is the call graph for this function:



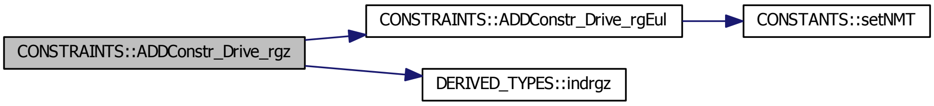

#### 4.2.2.10 subroutine,public CONSTRAINTS::ADDConstr_Drive_rgz ( INTEGER,intent(in) *body,* INTEGER,intent(in) *i_MOTOR* )

Adds a driving constraint to the z-coordinate of the CDM of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *i_MOTOR* | index of the kinematic actuator. The user must evaluate the kinematic guidance function and introduce the function and the derivatives in the vectors STATE::pos,STATE::vel, STATE::ace in each time step: STATE::pos(i_-MOTOR),STATE::vel(i_MOTOR),STATE::ace(i_MOTOR) contain the kinematic guidance function and its derivatives for the variable rgz. |

Here is the call graph for this function:



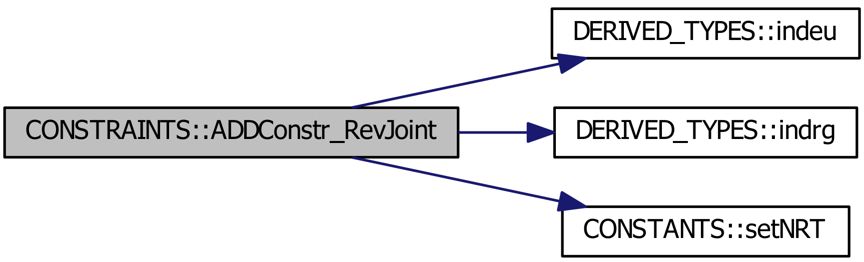

#### 4.2.2.11 subroutine,public CONSTRAINTS::ADDConstr_RevJoint ( INTEGER,intent(in) *body1*, INTEGER,intent(in) *body2*, REAL(8),dimension(3),intent(in) *point1*, REAL(8),dimension(3),intent(in) *point2*, REAL(8),dimension(3),intent(in) *vect1*, REAL(8),dimension(3),intent(in) *vect2* )

Adds a revolute joint between two bodies or between the ground and one body.

**Parameters**

| | |
|---|---|
| *body1* | first body involved. It can be "SOLIDS::ground" if the second body is attached to the ground. |
| *body2* | second body involved. It cannot be the ground. |
| *point1* | point in the first body/ground in the body reference frame. |
| *point2* | point in the second body in the body reference frame. |
| *vect1* | vector in the first body/ground in the body reference frame. |
| *vect2* | vector in the second body in the body reference frame. |

Here is the call graph for this function:



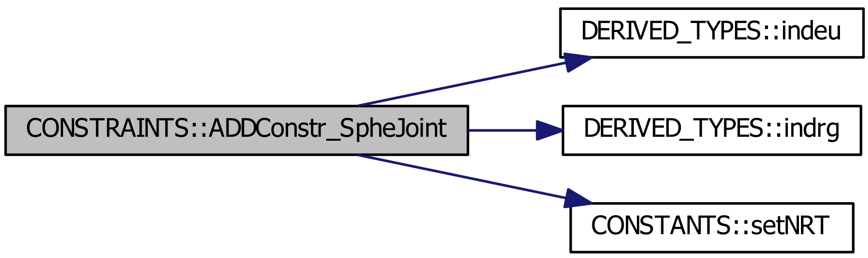
#### 4.2.2.12 subroutine,public CONSTRAINTS::ADDConstr_SpheJoint ( INTEGER,intent(in) *body1*, INTEGER,intent(in) *body2*, real(8),dimension(3),intent(in) *point1*, real(8),dimension(3),intent(in) *point2* )

Adds a sphercal joint between two bodies or between the ground and one body.

**Parameters**

| | |
|---|---|
| *body1* | first body involved. It can be "SOLIDS::ground" if the second body is attached to the ground. |
| *body2* | second body involved. It cannot be the ground. |
| *point1* | point in the first body/ground in the body reference frame. |
| *point2* | point in the second body in the body reference frame. |

Here is the call graph for this function:




#### 4.2.2.13 subroutine,public CONSTRAINTS::ADDConstr_TransJoint ( INTEGER,intent(in) *body1*, INTEGER,intent(in) *body2*, REAL(8),dimension(3),intent(in) *point1*, REAL(8),dimension(3),intent(in) *point2*, REAL(8),dimension(3),intent(in) *vect1x*, REAL(8),dimension(3),intent(in) *vect1y*, REAL(8),dimension(3),intent(in) *vect2x*, REAL(8),dimension(3),intent(in) *vect2y* )

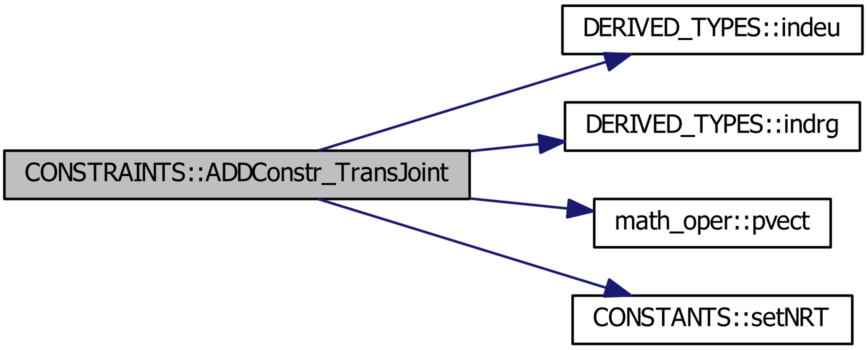
Adds a translational joint between two bodies or between the ground and one body.

**Parameters**

| | |
|---|---|
| *body1* | first body involved. It can be "SOLIDS::ground" if the second body is attached to the ground. |
| *body2* | second body involved. It cannot be the ground. |
| *point1* | point in the first body/ground in the body reference frame. |
| *point2* | point in the second body in the body reference frame. |
| *vect1y* | vector along the tranlating direction in the first body/ground in the body reference frame. |

| | |
|---|---|
| *vect1x* | vector perpendicular to vect1y in the first body/ground in the body reference frame. |
| *vect2y* | vector along the tranlating direction in the second body in the body reference frame. |
| *vect2x* | vector perpendicular to vect1y and parallel to vect1x in the second body in the body reference frame. |

Here is the call graph for this function:



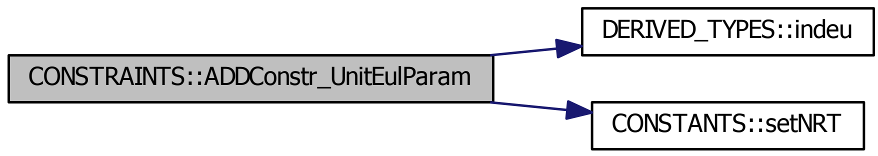

#### 4.2.2.14 subroutine,public CONSTRAINTS::ADDConstr_UnitEulParam ( INTEGER,intent(in) *body* )

Adds an unitary vector constraint to the Euler parameters of one body. It's NOT a user function, it's intended to be called by the solver.

**Parameters**

| | |
|---|---|
| *body* | corresponding body |

Here is the call graph for this function:



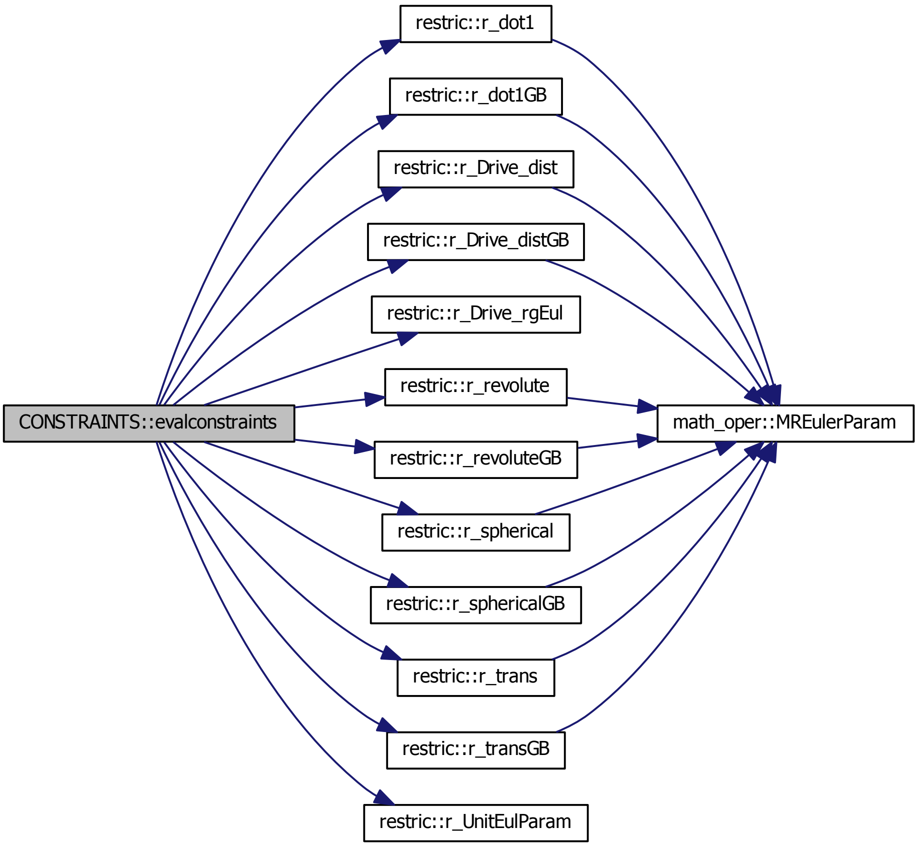

#### 4.2.2.15 subroutine,public CONSTRAINTS::CONSTRAINTS_Setup ( )

Here is the call graph for this function:


#### 4.2.2.16 subroutine,public CONSTRAINTS::evalconstraints ( )

Subroutine that evaluates the constraints vector ( $\mathbf{\Phi}$). It's NOT a user function, it's intended to be called by the solver.

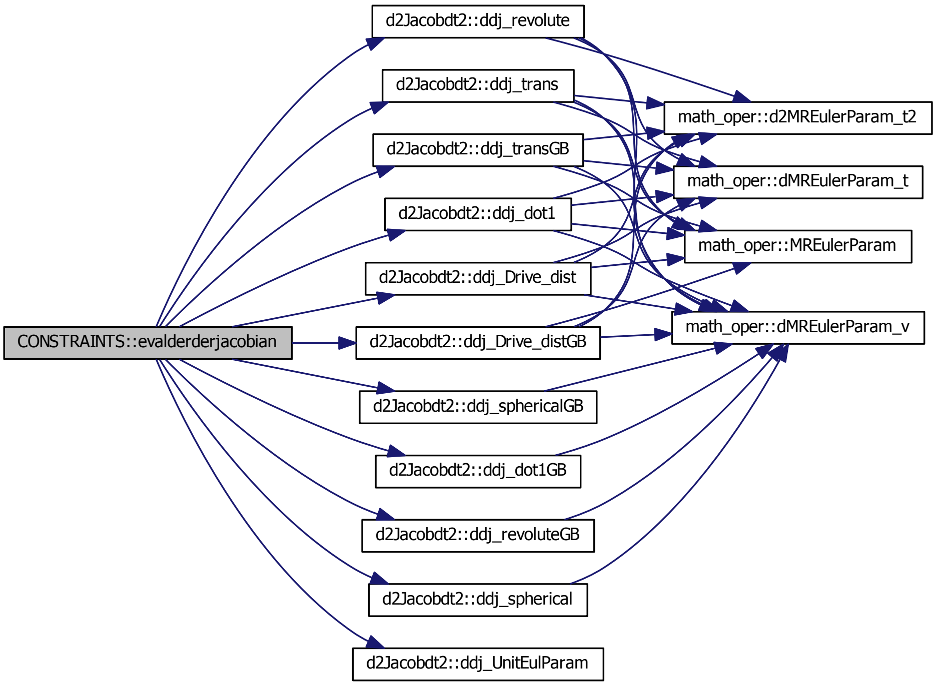
Here is the call graph for this function:




#### 4.2.2.17 subroutine,public CONSTRAINTS::evalderderjacobian ( )

Subroutine that evaluates the term $\ddot{\mathbf{\Phi}}_{\mathbf{q}}$. It's NOT a user function, it's intended to be called by the solver.

Here is the call graph for this function:



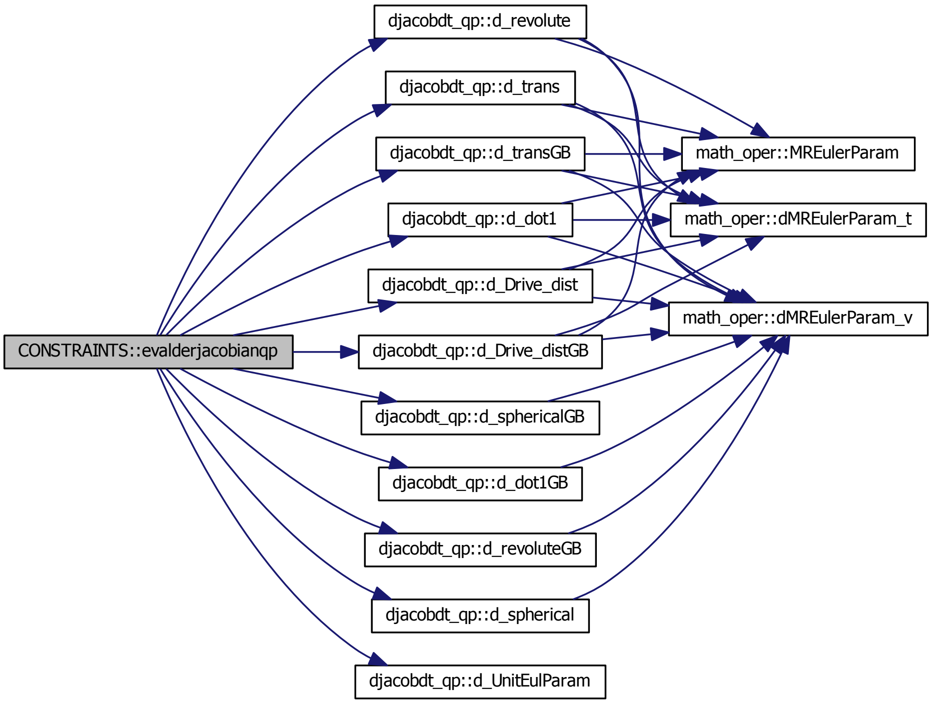

#### 4.2.2.18 subroutine,public CONSTRAINTS::evalderjacobian ( )

Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_{\mathbf{q}}$. It's NOT a user function, it's intended to be called by the solver.

#### 4.2.2.19 subroutine,public CONSTRAINTS::evalderjacobianqp ( )

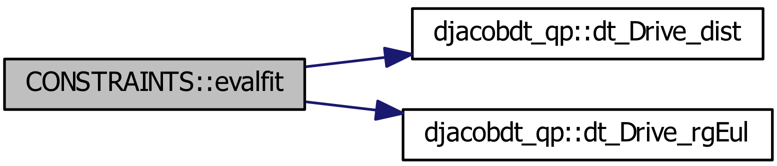
Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$. It's NOT a user function, it's intended to be called by the solver.

Here is the call graph for this function:



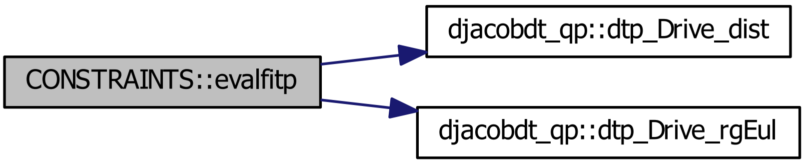

#### 4.2.2.20 subroutine,public CONSTRAINTS::evalfit ( )

Here is the call graph for this function:



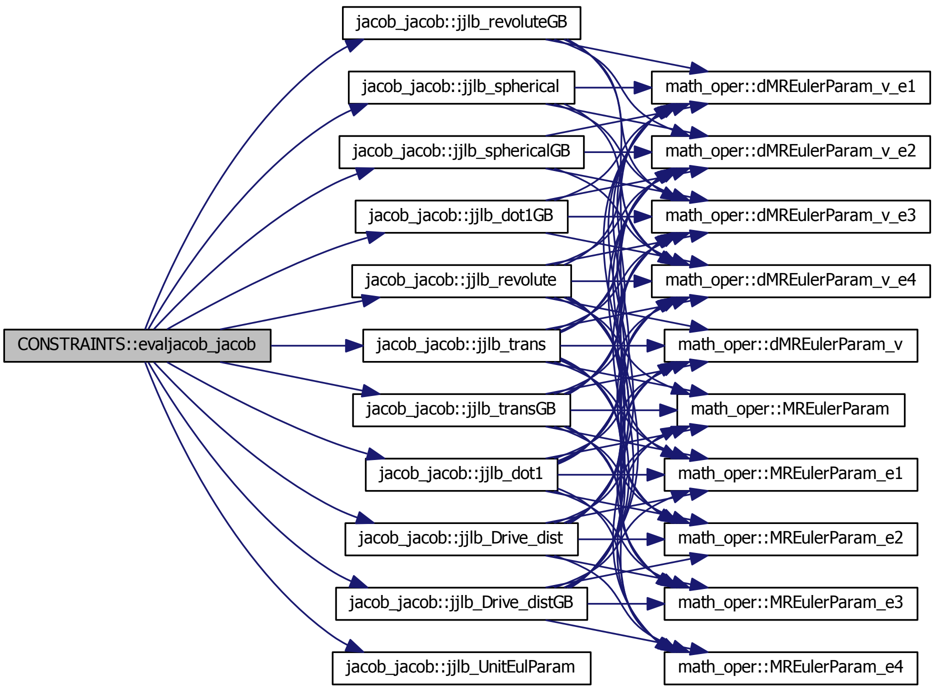
#### 4.2.2.21 subroutine,public CONSTRAINTS::evalfitp ( )

Here is the call graph for this function:



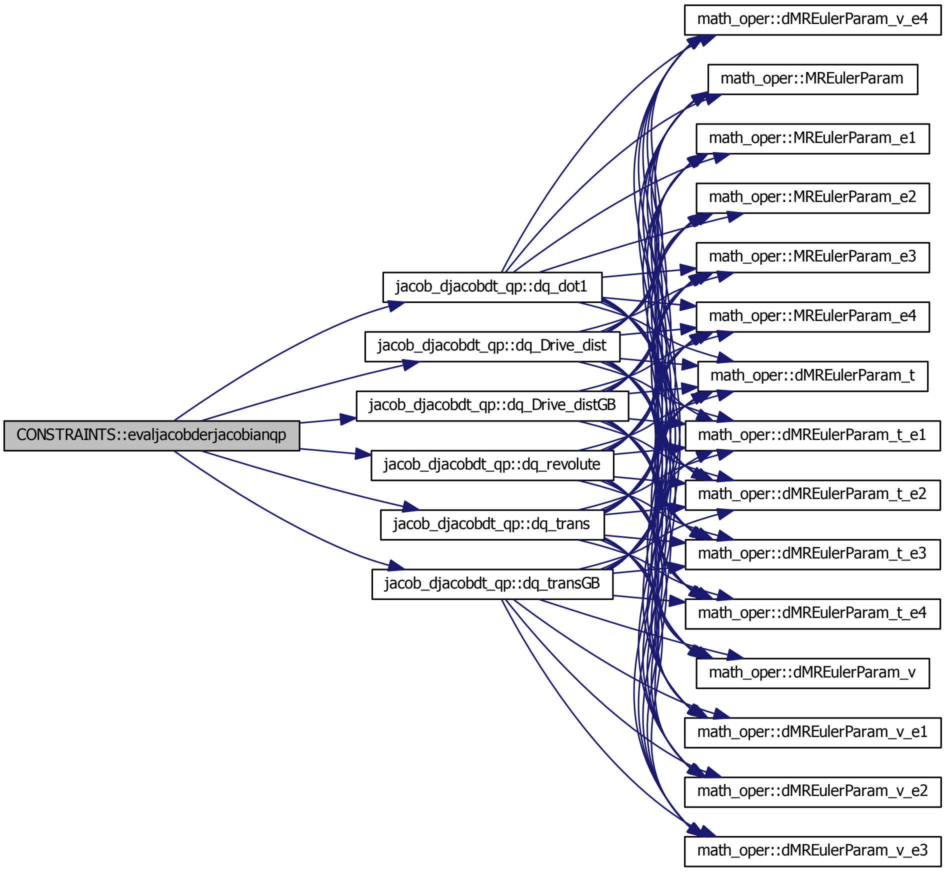
#### 4.2.2.22 subroutine,public CONSTRAINTS::evaljacob_jacob ( real(8),dimension(dim),intent(in) *lb* )

Here is the call graph for this function:



#### 4.2.2.23 subroutine,public CONSTRAINTS::evaljacobderjacobianqp ( )

Here is the call graph for this function:

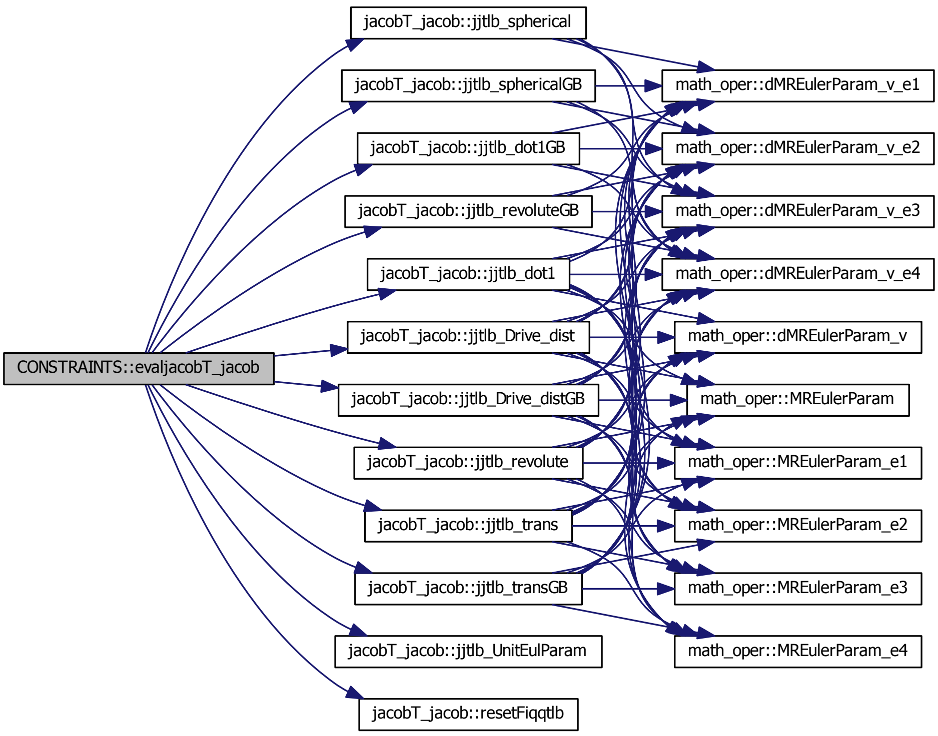



#### 4.2.2.24 subroutine,public CONSTRAINTS::evaljacobian ( )

Subroutine that evaluates the jacobian of the constraints vector ( $\mathbf{\Phi_q}$). It's NOT a user function, it's intended to be called by the solver.

#### 4.2.2.25 subroutine,public CONSTRAINTS::evaljacobT_jacob ( real(8),dimension(nrt),intent(in) *lb* )

Here is the call graph for this function:




### 4.2.3 Variable Documentation

#### 4.2.3.1 TYPE (typeConstr_dot1),dimension(:),allocatable CONSTRAINTS::Constr_dot1

#### 4.2.3.2 TYPE (typeConstr_dot1),dimension(:),allocatable CONSTRAINTS::Constr_dot1GB

#### 4.2.3.3 TYPE (typeconstr_Drive_Dist),dimension(:),allocatable CONSTRAINTS::Constr_Drive_Dist

#### 4.2.3.4 TYPE (typeconstr_Drive_Dist),dimension(:),allocatable CONSTRAINTS::Constr_Drive_DistGB

#### 4.2.3.5 TYPE (typeconstr_Drive_rgEul),dimension(:),allocatable CONSTRAINTS::Constr_Drive_rgEul

#### 4.2.3.6 TYPE (typeConstr_RevJoint),dimension(:),allocatable CONSTRAINTS::Constr_RevJoint

#### 4.2.3.7 TYPE (typeConstr_RevJoint),dimension(:),allocatable CONSTRAINTS::Constr_RevJointGB

#### 4.2.3.8 TYPE (typeConstr_SpheJoint),dimension(:),allocatable CONSTRAINTS::Constr_SpheJoint

#### 4.2.3.9 TYPE (typeConstr_SpheJoint),dimension(:),allocatable CONSTRAINTS::Constr_SpheJointGB

#### 4.2.3.10 TYPE (typeConstr_TransJoint),dimension(:),allocatable CONSTRAINTS::Constr_TransJoint

#### 4.2.3.11 TYPE (typeConstr_TransJoint),dimension(:),allocatable CONSTRAINTS::Constr_TransJointGB

#### 4.2.3.12 TYPE (typeConstr_UnitEulParam),dimension(:),allocatable CONSTRAINTS::Constr_UnitEulParam

#### 4.2.3.13 INTEGER CONSTRAINTS::nConstr_dot1 = 0

#### 4.2.3.14 INTEGER CONSTRAINTS::nConstr_dot1GB = 0

#### 4.2.3.15 INTEGER CONSTRAINTS::nConstr_Drive_dist = 0

#### 4.2.3.16 INTEGER CONSTRAINTS::nConstr_Drive_distGB = 0

#### 4.2.3.17 INTEGER CONSTRAINTS::nConstr_Drive_rgEul = 0

#### 4.2.3.18 INTEGER CONSTRAINTS::nConstr_RevJoint = 0

#### 4.2.3.19 INTEGER CONSTRAINTS::nConstr_RevJointGB = 0

#### 4.2.3.20 INTEGER CONSTRAINTS::nConstr_SpheJoint = 0

#### 4.2.3.21 INTEGER CONSTRAINTS::nConstr_SpheJointGB = 0

#### 4.2.3.22 INTEGER CONSTRAINTS::nConstr_TransJoint = 0

#### 4.2.3.23 INTEGER CONSTRAINTS::nConstr_TransJointGB = 0

#### 4.2.3.24 INTEGER CONSTRAINTS::nConstr_UnitEulParam = 0

## 4.3 d2Jacobdt2 Module Reference

Module of second derivatives of the Jacobian. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine d2Jacobdt2_Setup
- subroutine deallocFiqpp
- subroutine ddj_UnitEulParam (ir, iEul)

  *The second derivatives of the jacobians of unitary Euler parameters.*
- subroutine ddj_dot1GB (ir, iEul2, u, v)

  *The second derivative of the jacobians of a dot-1 constraint attached on the ground.*
- subroutine ddj_dot1 (ir, iEul1, iEul2, u, v)

  *The second derivatives of the jacobians of the jacobians of a dot-1 constraint.*
- subroutine ddj_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *The second derivatives of the jacobians of of a spherical joint of a body attached to the ground.*
- subroutine ddj_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *The second derivatives of the jacobians of of a spherical joint between two bodies.*
- subroutine ddj_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *The second derivatives of the jacobians of of a revolute joint of a body attached to the ground.*
- subroutine ddj_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *The second derivatives of the jacobians of of a revolute joint between two bodies.*
- subroutine ddj_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The second derivatives of the jacobians of of a translational joint of a body attached to the ground.*
- subroutine ddj_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The second derivatives of the jacobians of a translational joint between two bodies.*
- subroutine ddj_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *The second derivatives of the jacobians for a distance between a point in the ground and a point of one body.*
- subroutine ddj_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *The second derivatives of the jacobians for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiqpp

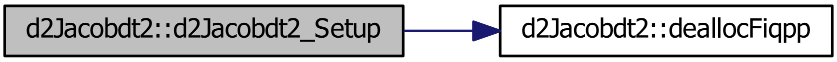
### 4.3.1 Detailed Description

Module of second derivatives of the Jacobian. It's NOT a user module, it's used by the solver.

### 4.3.2 Function/Subroutine Documentation

#### 4.3.2.1 subroutine d2Jacobdt2::d2Jacobdt2_Setup ( )

Here is the call graph for this function:



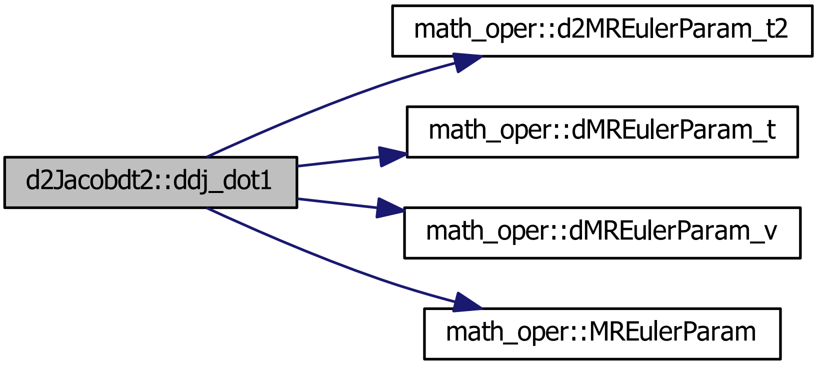

#### 4.3.2.2 subroutine d2Jacobdt2::ddj_dot1 ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

The second derivatives of the jacobians of the jacobians of a dot-1 constraint.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



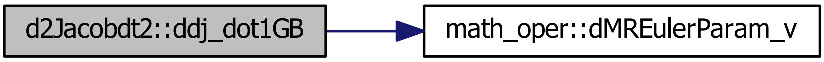

#### 4.3.2.3 subroutine d2Jacobdt2::ddj_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

The second derivative of the jacobians of a dot-1 constraint attached on the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



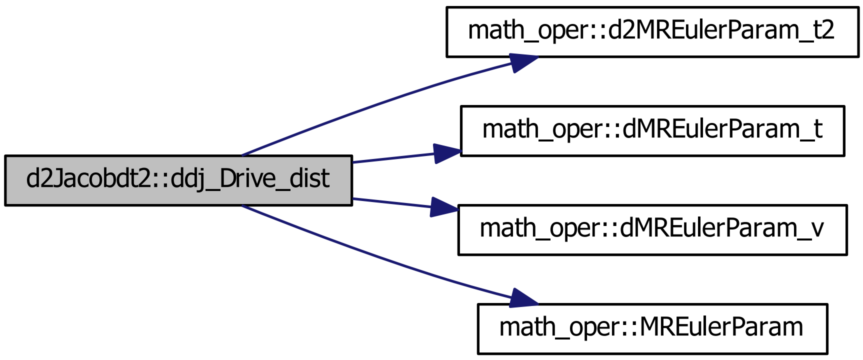
#### 4.3.2.4 subroutine d2Jacobdt2::ddj_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

The second derivatives of the jacobians for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



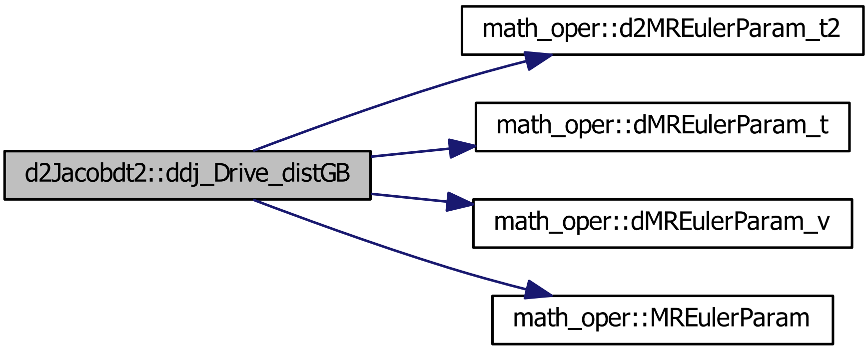

#### 4.3.2.5 subroutine d2Jacobdt2::ddj_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

The second derivatives of the jacobians for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



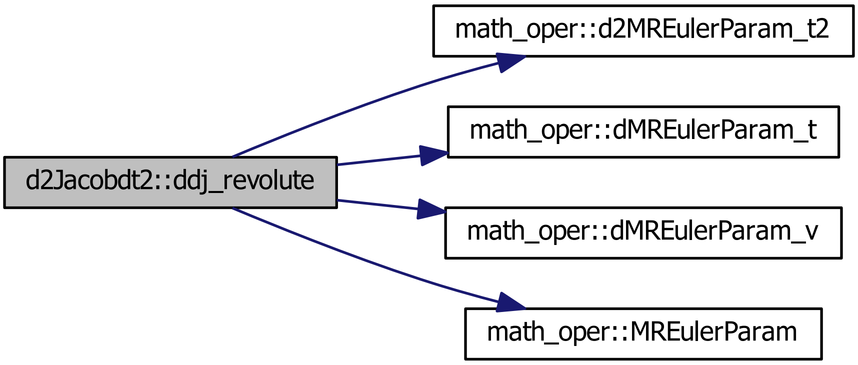

#### 4.3.2.6 subroutine d2Jacobdt2::ddj_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg1*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),dimension(3),intent(in) *u1*, real(8),dimension(3),intent(in) *v1*, real(8),dimension(3),intent(in) *vec2* )

The second derivatives of the jacobians of of a revolute joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



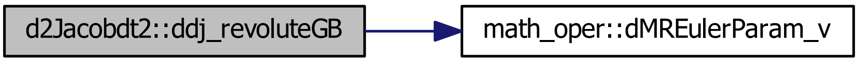

#### 4.3.2.7 subroutine d2Jacobdt2::ddj_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2* )

The second derivatives of the jacobians of of a revolute joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,u2* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |

Here is the call graph for this function:



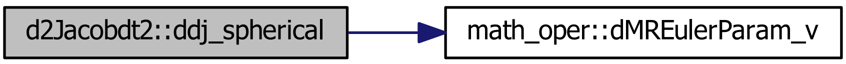

#### 4.3.2.8 subroutine d2Jacobdt2::ddj_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

The second derivatives of the jacobians of of a spherical joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |

Here is the call graph for this function:



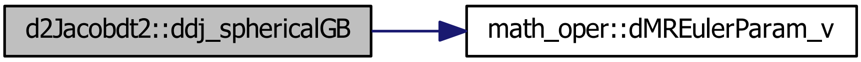

#### 4.3.2.9 subroutine d2Jacobdt2::ddj_sphericalGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

The second derivatives of the jacobians of of a spherical joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

Here is the call graph for this function:



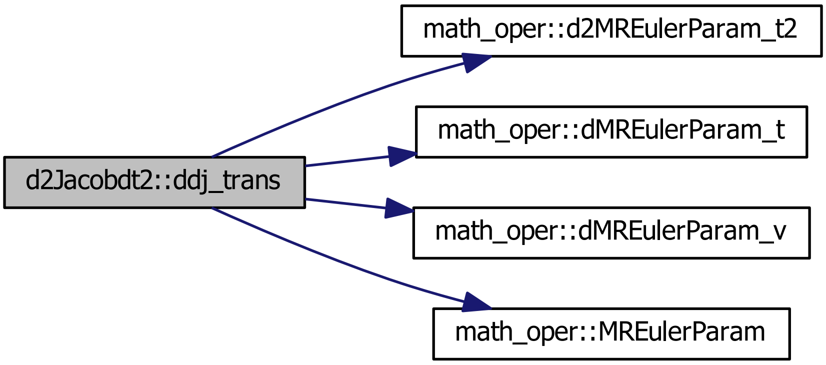

#### 4.3.2.10 subroutine d2Jacobdt2::ddj_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

The second derivatives of the jacobians of a translational joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



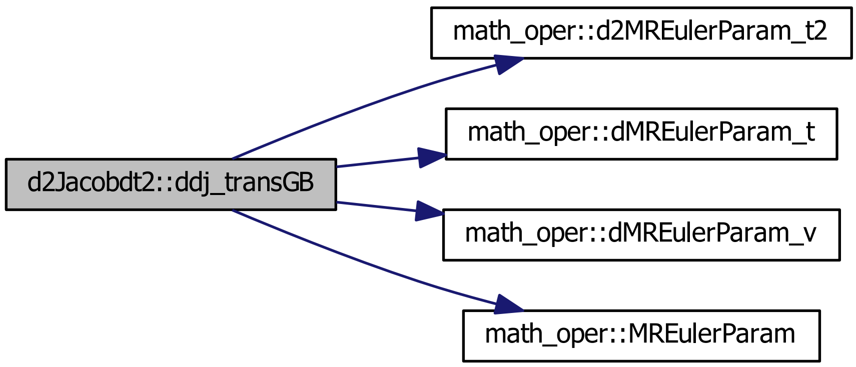

#### 4.3.2.11 subroutine d2Jacobdt2::ddj_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

The second derivatives of the jacobians of of a translational joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:




#### 4.3.2.12 subroutine d2Jacobdt2::ddj_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul* )

The second derivatives of the jacobians of unitary Euler parameters.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |

#### 4.3.2.13 subroutine d2Jacobdt2::deallocFiqpp ( )

### 4.3.3 Variable Documentation

#### 4.3.3.1 REAL(8),dimension(:,:),allocatable d2Jacobdt2::Fiqpp

#### 4.3.3.2 REAL(8),dimension(:,:),allocatable d2Jacobdt2::PROTECTED

## 4.4 DERIVED_TYPES Module Reference

Module of solver derived types definition and subroutines/functions to manage the derived types.

## Data Types

- type MATRIXTRANSFORM
- type POINT
- type SOLID
- type typeConstr_UnitEulParam

  *Euler parameters constraints.*
- type typeConstr_dot1

  *Dot-1 constraints.*
- type typeConstr_SpheJoint

  *Spherical joint constraints.*
- type typeConstr_RevJoint

  *Revolute joint constraints.*
- type typeConstr_TransJoint

  *Translational joint constraints.*
- type typeconstr_Drive_rgEul

  *Driving constraints coordinates.*
- type typeconstr_Drive_Dist

  *Driving distance constraints.*
- type typeforce_TSDA

  *TSDA forces.*
- interface callback_forces
- interface callback_stiffness
- interface callback_damping
- interface callback_PQbarPrho
- interface callback_PMbarPrhoVdot
- interface callback_dgdy
- interface callback_dgdp
- interface callback_AdjInit
- interface callback_gfun

## Functions/Subroutines

- INTEGER, dimension(7) indre (nSOLID)

  *Function that returns the index for all the states of the body.*
- INTEGER, dimension(3) indrg (nSOLID)

  *Function that returns the index for the CDM of the body.*
- INTEGER indrgx (nSOLID)
- INTEGER indrgy (nSOLID)
- INTEGER indrgz (nSOLID)
- INTEGER, dimension(4) indeu (nSOLID)

  *Function that returns the index for the Euler parameters of the body.*
- INTEGER indeu0 (nSOLID)
- INTEGER indeu1 (nSOLID)
- INTEGER indeu2 (nSOLID)
- INTEGER indeu3 (nSOLID)

### 4.4.1 Detailed Description

Module of solver derived types definition and subroutines/functions to manage the derived types.

### 4.4.2 Function/Subroutine Documentation

#### 4.4.2.1 INTEGER,dimension(4) DERIVED_TYPES::indeu ( INTEGER,intent(in) *nSOLID* )

Function that returns the index for the Euler parameters of the body.

**Parameters**

| | |
|---|---|
| *nSOLID* | numer of the body |

#### 4.4.2.2 INTEGER DERIVED_TYPES::indeu0 ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.3 INTEGER DERIVED_TYPES::indeu1 ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.4 INTEGER DERIVED_TYPES::indeu2 ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.5 INTEGER DERIVED_TYPES::indeu3 ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.6 INTEGER,dimension(7) DERIVED_TYPES::indre ( INTEGER,intent(in) *nSOLID* )

Function that returns the index for all the states of the body.

**Parameters**

| | |
|---|---|
| *nSOLID* | numer of the body |

#### 4.4.2.7 INTEGER,dimension(3) DERIVED_TYPES::indrg ( INTEGER,intent(in) *nSOLID* )

Function that returns the index for the CDM of the body.

**Parameters**

| | |
|---|---|
| *nSOLID* | numer of the body |

#### 4.4.2.8 INTEGER DERIVED_TYPES::indrgx ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.9 INTEGER DERIVED_TYPES::indrgy ( INTEGER,intent(in) *nSOLID* )

#### 4.4.2.10 INTEGER DERIVED_TYPES::indrgz ( INTEGER,intent(in) *nSOLID* )

## 4.5 dJacobdt Module Reference

Module of total derivatives of the Jacobian. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine dJacobdt_Setup
- subroutine deallocFiqp
- subroutine dj_UnitEulParam (ir, iEul)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC.*
- subroutine dj_dot1GB (ir, iEul2, u, v)

  *The first derivative of the jacobians of a dot-1 constraint attached on the ground.*
- subroutine dj_dot1 (ir, iEul1, iEul2, u, v)

  *The first derivative of the jacobians of a dot-1 constraint.*
- subroutine dj_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *The first derivative of the jacobians of a spherical joint of a body attached to the ground.*
- subroutine dj_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *The first derivative of the jacobians of a spherical joint between two bodies.*
- subroutine dj_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *The first derivative of the jacobians of a revolute joint of a body attached to the ground.*
- subroutine dj_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *The first derivative of the jacobians of a revolute joint between two bodies.*
- subroutine dj_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The first derivative of the jacobians of a translational joint of a body attached to the ground.*
- subroutine dj_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The first derivative of the jacobians of a translational joint between two bodies.*
- subroutine dj_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *The first derivative of the jacobians for a distance between a point in the ground and a point of one body.*
- subroutine dj_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *The first derivative of the jacobians for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiqp

### 4.5.1 Detailed Description

Module of total derivatives of the Jacobian. It's NOT a user module, it's used by the solver.

### 4.5.2 Function/Subroutine Documentation

#### 4.5.2.1 subroutine dJacobdt::deallocFiqp ( )

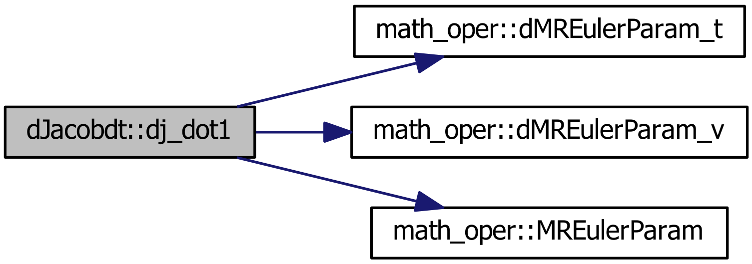
#### 4.5.2.2 subroutine dJacobdt::dj_dot1 ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

The first derivative of the jacobians of a dot-1 constraint.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



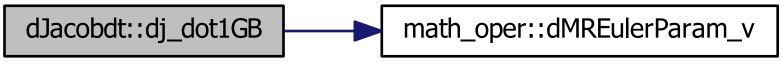

#### 4.5.2.3 subroutine dJacobdt::dj_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

The first derivative of the jacobians of a dot-1 constraint attached on the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



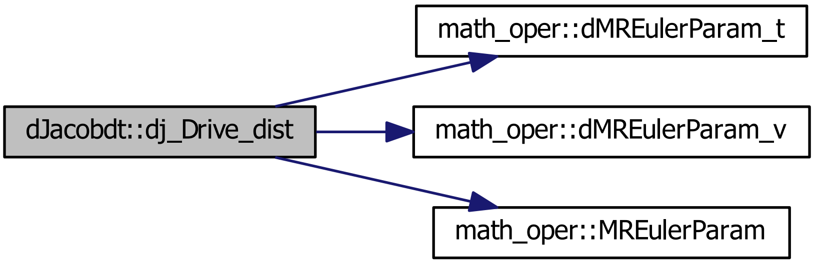

#### 4.5.2.4 subroutine dJacobdt::dj_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

The first derivative of the jacobians for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



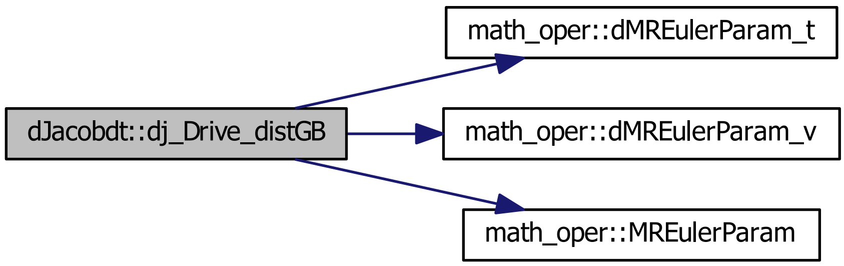
#### 4.5.2.5 subroutine dJacobdt::dj_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

The first derivative of the jacobians for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:




#### 4.5.2.6 subroutine dJacobdt::dj_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg1*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),dimension(3),intent(in) *u1*, real(8),dimension(3),intent(in) *v1*, real(8),dimension(3),intent(in) *vec2* )

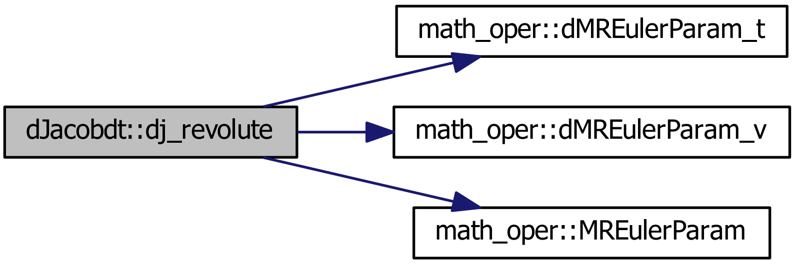
The first derivative of the jacobians of a revolute joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |

| | |
|---|---|
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



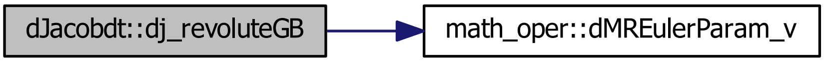

#### 4.5.2.7 subroutine dJacobdt::dj_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2* )

The first derivative of the jacobians of a revolute joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,u2* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |

Here is the call graph for this function:



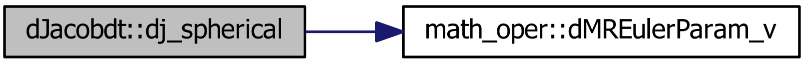

#### 4.5.2.8 subroutine dJacobdt::dj_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

The first derivative of the jacobians of a spherical joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |

Here is the call graph for this function:



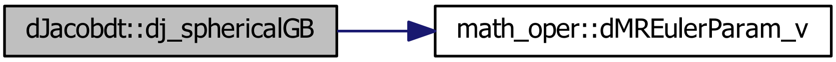

#### 4.5.2.9 subroutine dJacobdt::dj_sphericalGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

The first derivative of the jacobians of a spherical joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

Here is the call graph for this function:



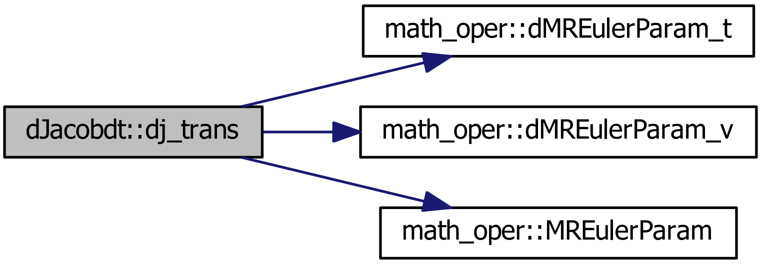

#### 4.5.2.10 subroutine dJacobdt::dj_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

The first derivative of the jacobians of a translational joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



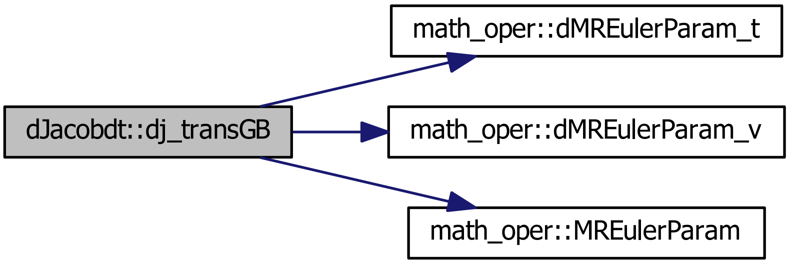

#### 4.5.2.11 subroutine dJacobdt::dj_transGB ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *vec1y,* REAL(8),dimension(3),intent(in) *vec1x,* REAL(8),dimension(3),intent(in) *vec2x,* REAL(8),dimension(3),intent(in) *vec2z* )

The first derivative of the jacobians of a translational joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:




#### 4.5.2.12 subroutine dJacobdt::dj_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul* )

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

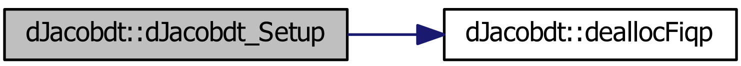
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
The first derivative of the jacobians of unitary Euler parameters

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |

#### 4.5.2.13 subroutine dJacobdt::dJacobdt_Setup ( )

Here is the call graph for this function:



### 4.5.3 Variable Documentation

#### 4.5.3.1 REAL(8),dimension(:,:),allocatable dJacobdt::Fiqp

#### 4.5.3.2 REAL(8),dimension(:,:),allocatable dJacobdt::PROTECTED

## 4.6 djacobdt_qp Module Reference

Module of derivatives of the Jacobian mutiplied by the velocity vector. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine djacobdt_qp_Setup
- subroutine deallocfiqpqp
- subroutine deallocfit
- subroutine d_UnitEulParam (ir, iEul)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$*, which is the derevative of jacobian with respect to time multiplies the velocity vector* $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of unitary Euler parameters*
- subroutine d_dot1GB (ir, iEul2, u, v)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a dot-1 constraint attached on the ground*
- subroutine d_dot1 (ir, iEul1, iEul2, u, v)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a dot-1 constraint.*
- subroutine d_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a spherical joint of a body attached to the ground*
- subroutine d_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a spherical joint between two bodies*
- subroutine d_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a revolute joint of a body attached to the ground*
- subroutine d_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a revolute joint between two bodies*
- subroutine d_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a translational joint of a body attached to the ground*
- subroutine d_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a translational joint between two bodies*
- subroutine d_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance between a point in the ground and a point of one body.*
- subroutine d_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance between two points of two bodies.*
- subroutine dt_Drive_rgEul (ir, ind, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a generalized coordinate of the system.*
- subroutine dtp_Drive_rgEul (ir, ind, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a generalized coordinate of the system.*

- subroutine dt_Drive_dist (ir, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance.*
- subroutine dtp_Drive_dist (ir, i_MOTOR)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance.*

### Variables

- REAL(8), dimension(:), allocatable PROTECTED
- REAL(8), dimension(:), allocatable fiqpqp
- REAL(8), dimension(:), allocatable fit
- REAL(8), dimension(:), allocatable fitp

## 4.6.1 Detailed Description

Module of derivatives of the Jacobian mutiplied by the velocity vector. It's NOT a user module, it's used by the solver.

## 4.6.2 Function/Subroutine Documentation

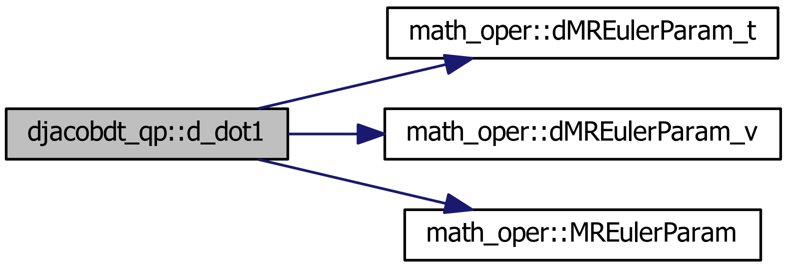
### 4.6.2.1 subroutine djacobdt_qp::d_dot1 ( integer,intent(in) *ir,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* real(8),dimension(3),intent(in) *u,* real(8),dimension(3),intent(in) *v* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a dot-1 constraint.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



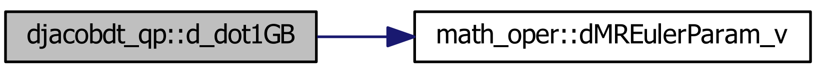

#### 4.6.2.2 subroutine djacobdt_qp::d_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a dot-1 constraint attached on the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



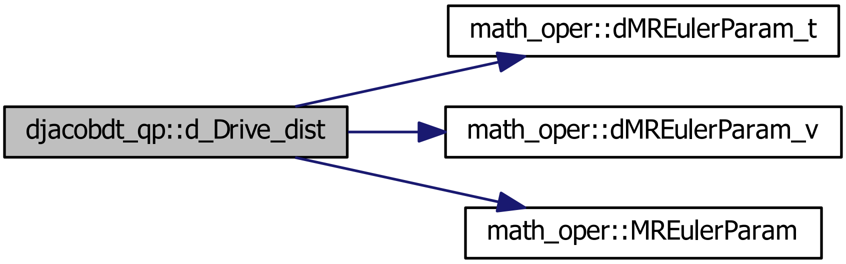
#### 4.6.2.3 subroutine djacobdt_qp::d_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:




#### 4.6.2.4 subroutine djacobdt_qp::d_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

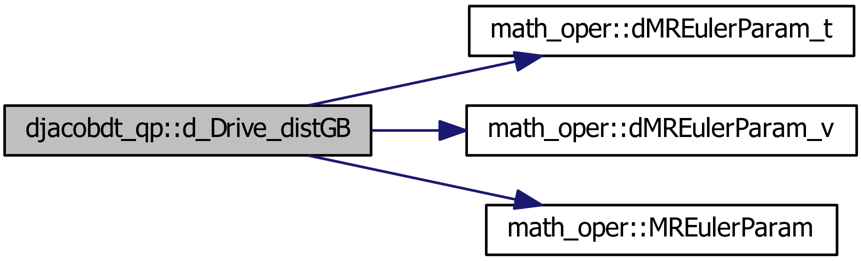
$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |

| | |
|---|---|
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



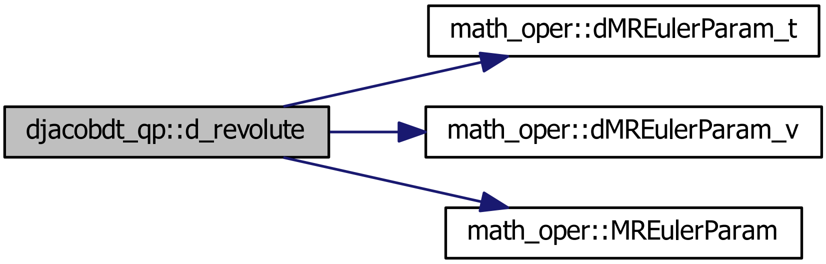

#### 4.6.2.5 subroutine djacobdt_qp::d_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg1*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),dimension(3),intent(in) *u1*, real(8),dimension(3),intent(in) *v1*, real(8),dimension(3),intent(in) *vec2* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a revolute joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



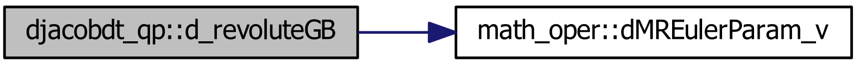

#### 4.6.2.6 subroutine djacobdt_qp::d_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in),optional *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),dimension(3),intent(in) *u1*, real(8),dimension(3),intent(in) *v1*, real(8),dimension(3),intent(in) *vec2* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a revolute joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,u2* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |

Here is the call graph for this function:



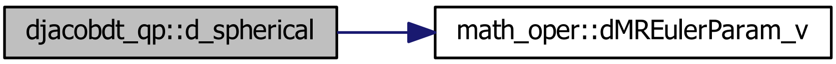
#### 4.6.2.7 subroutine djacobdt_qp::d_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg1*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a spherical joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |

Here is the call graph for this function:



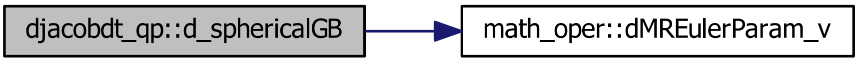

#### 4.6.2.8 subroutine djacobdt_qp::d_sphericalGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in),optional *pt1*, real(8),dimension(3),intent(in) *pt2* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a spherical joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

Here is the call graph for this function:



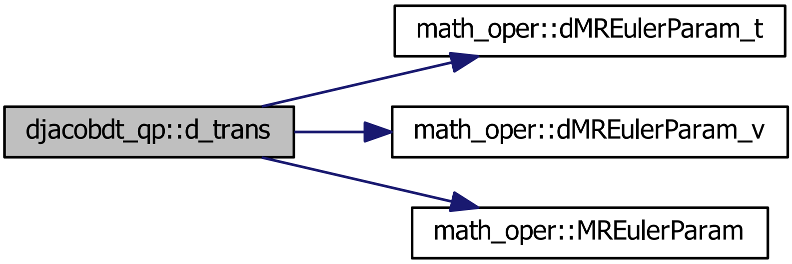

#### 4.6.2.9 subroutine djacobdt_qp::d_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a translational joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



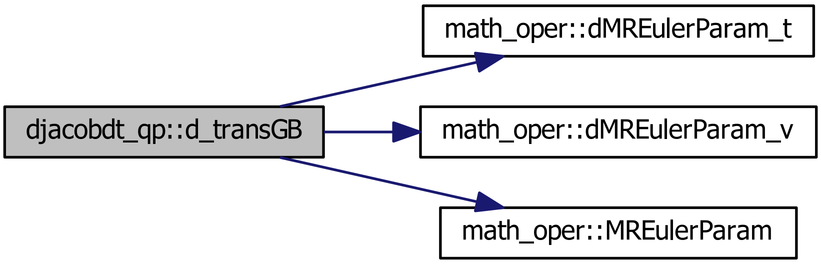

#### 4.6.2.10 subroutine djacobdt_qp::d_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of a translational joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:




#### 4.6.2.11 subroutine djacobdt_qp::d_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$, which is the derevative of jacobian with respect to time multiplies the velocity vector $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ of unitary Euler parameters

**Parameters**

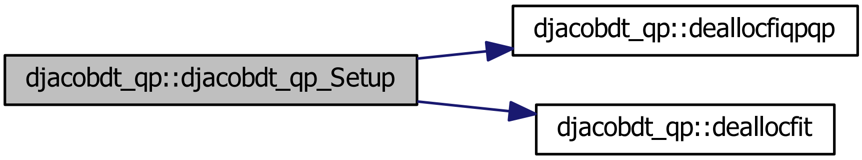
| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |

#### 4.6.2.12 subroutine djacobdt_qp::deallocfiqpqp ( )

#### 4.6.2.13 subroutine djacobdt_qp::deallocfit ( )

#### 4.6.2.14 subroutine djacobdt_qp::djacobdt_qp_Setup ( )

Here is the call graph for this function:




#### 4.6.2.15 subroutine djacobdt_qp::dt_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a distance.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

#### 4.6.2.16 subroutine djacobdt_qp::dt_Drive_rgEul ( INTEGER,intent(in) *ir*, INTEGER,intent(in) *ind*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a generalized coordinate of the system.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *ind* | index of the driven generalized coordinate. It is not necessary here, but it is kept for compatibility of the interfaces (less easy to make mistakes) |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

#### 4.6.2.17 subroutine djacobdt_qp::dtp_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a distance.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

#### 4.6.2.18 subroutine djacobdt_qp::dtp_Drive_rgEul ( INTEGER,intent(in) *ir*, INTEGER,intent(in) *ind*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ for a generalized coordinate of the system.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *ind* | index of the driven generalized coordinate. It is not necessary here, but it is kept for compatibility of the interfaces (less easy to make mistakes) |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

### 4.6.3 Variable Documentation

#### 4.6.3.1 REAL(8),dimension(:),allocatable djacobdt_qp::fiqpqp

#### 4.6.3.2 REAL(8),dimension(:),allocatable djacobdt_qp::fit

#### 4.6.3.3 REAL(8),dimension(:),allocatable djacobdt_qp::fitp

#### 4.6.3.4 REAL(8),dimension(:),allocatable djacobdt_qp::PROTECTED

## 4.7 forces Module Reference

### Functions/Subroutines

- subroutine force (t, n, F, p, Q)

  *Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.*
- subroutine TSDA (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized forces of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA_q (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA_qp (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized damping of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA (r1, r2, r1p, r2p, s0, k, c, F1, F2)

  *Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.*

- subroutine TSDA_q (r1, r2, r1p, r2p, s0, k, c, df1dr1, df1dr2, df2dr1, df2dr2)

  *Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA_qp (r1, r2, c, df1dr1p, df1dr2p, df2dr1p, df2dr2p)

  *Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.*

### 4.7.1 Function/Subroutine Documentation

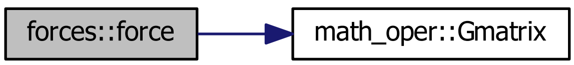
#### 4.7.1.1 subroutine forces::force ( REAL(8) *t*, REAL(8),dimension(3) *n*, REAL(8),dimension(3) *F*, REAL(8),dimension(4) *p*, REAL(8),dimension(7) *Q* )

Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *n* | torque act on the given body. |
| *F* | force act on the given body. |
| *P* | Euler parameters of the given body. |
| *Q* | Euler parameters of the given body. |

Here is the call graph for this function:




#### 4.7.1.2 subroutine forces::TSDA ( real(8),intent(in) *t*, integer,intent(in) *body1*, integer,intent(in) *body2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),intent(in) *s0*, real(8),intent(in) *k*, real(8),intent(in) *c*, real(8),dimension(7),intent(out) *Q1*, real(8),dimension(7),intent(out) *Q2* )

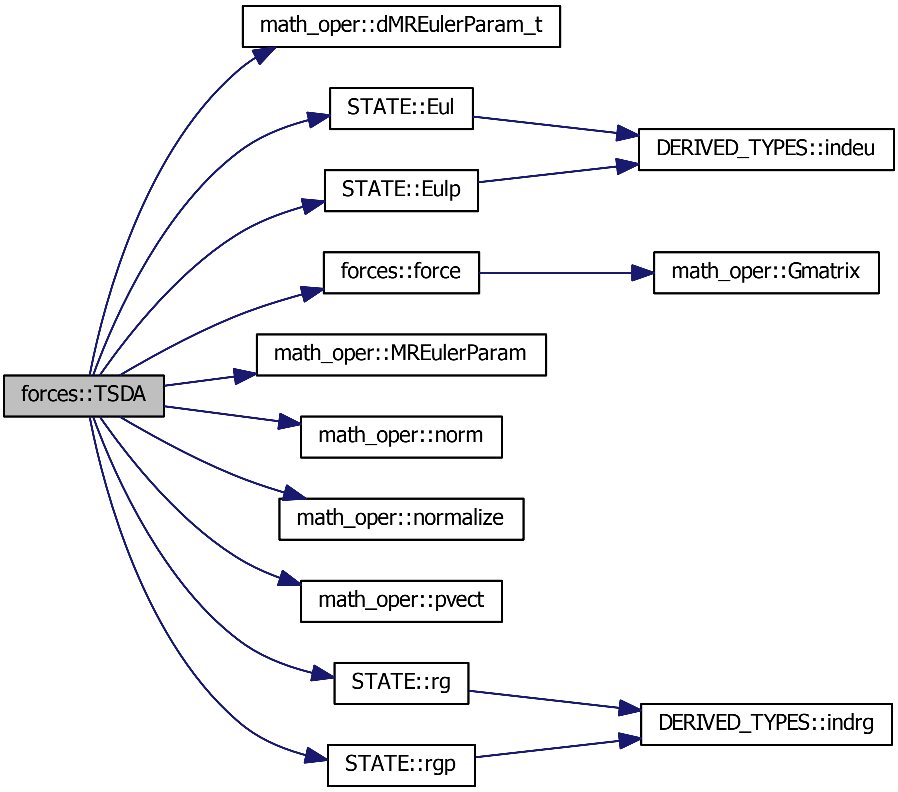
Function to get the generalized forces of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |

| | |
|---|---|
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *Q1* | return the generalized force acting on the first body |
| *Q2* | return the generalized force acting on the second body |

Here is the call graph for this function:



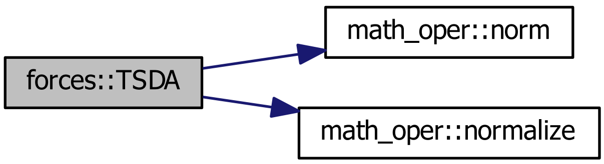
#### 4.7.1.3 subroutine forces::TSDA ( REAL(8),dimension(3),intent(in) *r1*, REAL(8),dimension(3),intent(in) *r2*, REAL(8),dimension(3),intent(in) *r1p*, REAL(8),dimension(3),intent(in) *r2p*, REAL(8),intent(in) *s0*, REAL(8),intent(in) *k*, REAL(8),intent(in) *c*, REAL(8),dimension(3),intent(out) *F1*, REAL(8),dimension(3),intent(out) *F2* )

Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *F1* | return the primitive force acting on the first body |
| *F2* | return the primitive force acting on the second body |

Here is the call graph for this function:



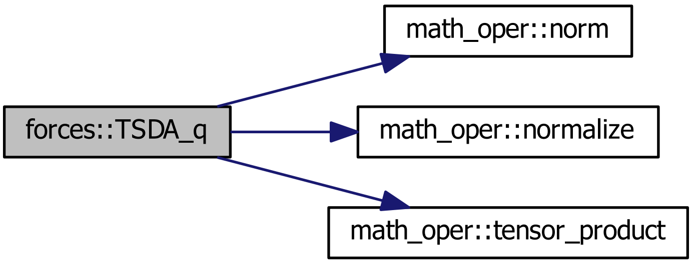

#### 4.7.1.4 subroutine forces::TSDA_q ( real(8),dimension(3),intent(in) *r1*, real(8),dimension(3),intent(in) *r2*, real(8),dimension(3),intent(in) *r1p*, real(8),dimension(3),intent(in) *r2p*, real(8),intent(in) *s0*, real(8),intent(in) *k*, real(8),intent(in) *c*, real(8),dimension(3,3),intent(out) *df1dr1*, real(8),dimension(3,3),intent(out) *df1dr2*, real(8),dimension(3,3),intent(out) *df2dr1*, real(8),dimension(3,3),intent(out) *df2dr2* )

Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *df1dr1,df1dr2* | return the primitive stiffness acting on the first body |
| *df2dr1,df2dr2* | return the primitive stiffness acting on the second body |

Here is the call graph for this function:




#### 4.7.1.5 subroutine forces::TSDA_q ( real(8),intent(in) *t*, integer,intent(in) *body1*, integer,intent(in) *body2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),intent(in) *s0*, real(8),intent(in) *k*, real(8),intent(in) *c*, real(8),dimension(7,7),intent(out) *Q1*, real(8),dimension(7,7),intent(out) *Q2* )

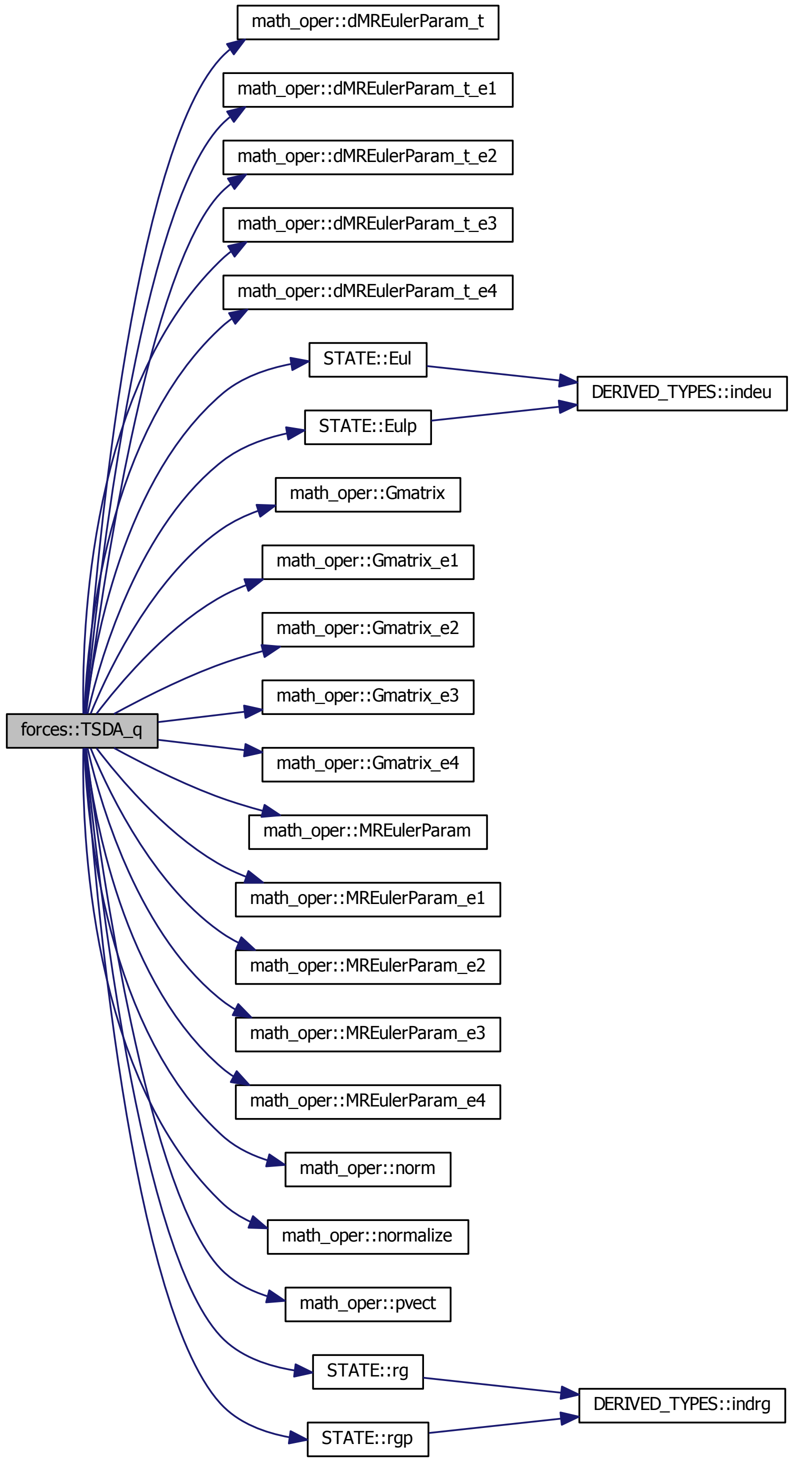
Function to get the generalized stiffness of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |

| | |
|---|---|
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *Q1* | return the generalized stiffness acting on the first body |
| *Q2* | return the generalized stiffness acting on the second body |

Here is the call graph for this function:



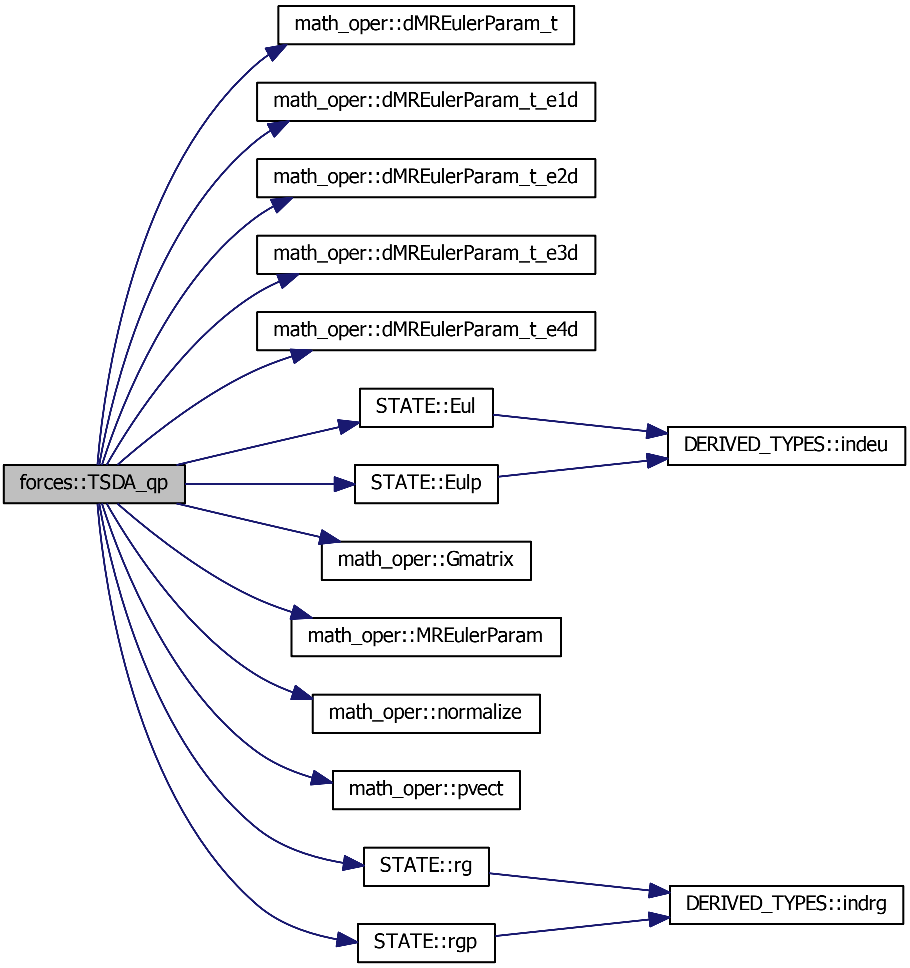
#### 4.7.1.6 subroutine forces::TSDA_qp ( real(8),intent(in) *t*, integer,intent(in) *body1*, integer,intent(in) *body2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),intent(in) *s0*, real(8),intent(in) *k*, real(8),intent(in) *c*, real(8),dimension(7,7),intent(out) *Q1*, real(8),dimension(7,7),intent(out) *Q2* )

Function to get the generalized damping of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *Q1* | return the generalized damping acting on the first body |
| *Q2* | return the generalized damping acting on the second body |

Here is the call graph for this function:



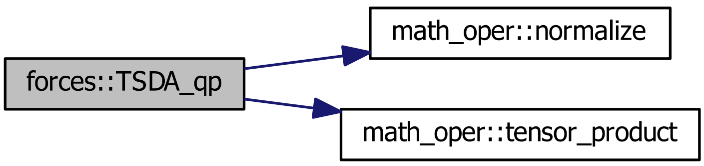

#### 4.7.1.7 subroutine forces::TSDA_qp ( real(8),dimension(3),intent(in) *r1*, real(8),dimension(3),intent(in) *r2*, real(8),intent(in) *c*, real(8),dimension(3,3),intent(out) *df1dr1p*, real(8),dimension(3,3),intent(out) *df1dr2p*, real(8),dimension(3,3),intent(out) *df2dr1p*, real(8),dimension(3,3),intent(out) *df2dr2p* )

Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *df1dr1p,df1dr2* | return the primitive damping acting on the first body |
| *df2dr1p,df2dr2* | return the primitive damping acting on the second body |

Here is the call graph for this function:




## 4.8 formulation_Dynamics Module Reference

Dynamic simulation module.

### Functions/Subroutines

- subroutine Acceleration_penalty (t)

  *Subrutine that solves the equations of motion for the acceleration using penalty (Partially taken from MBSLIM)*
- subroutine Penalty_fun (NVAR, t, y, yp)
- subroutine Penalty_Tang (N, T, Y, Fy)

### 4.8.1 Detailed Description

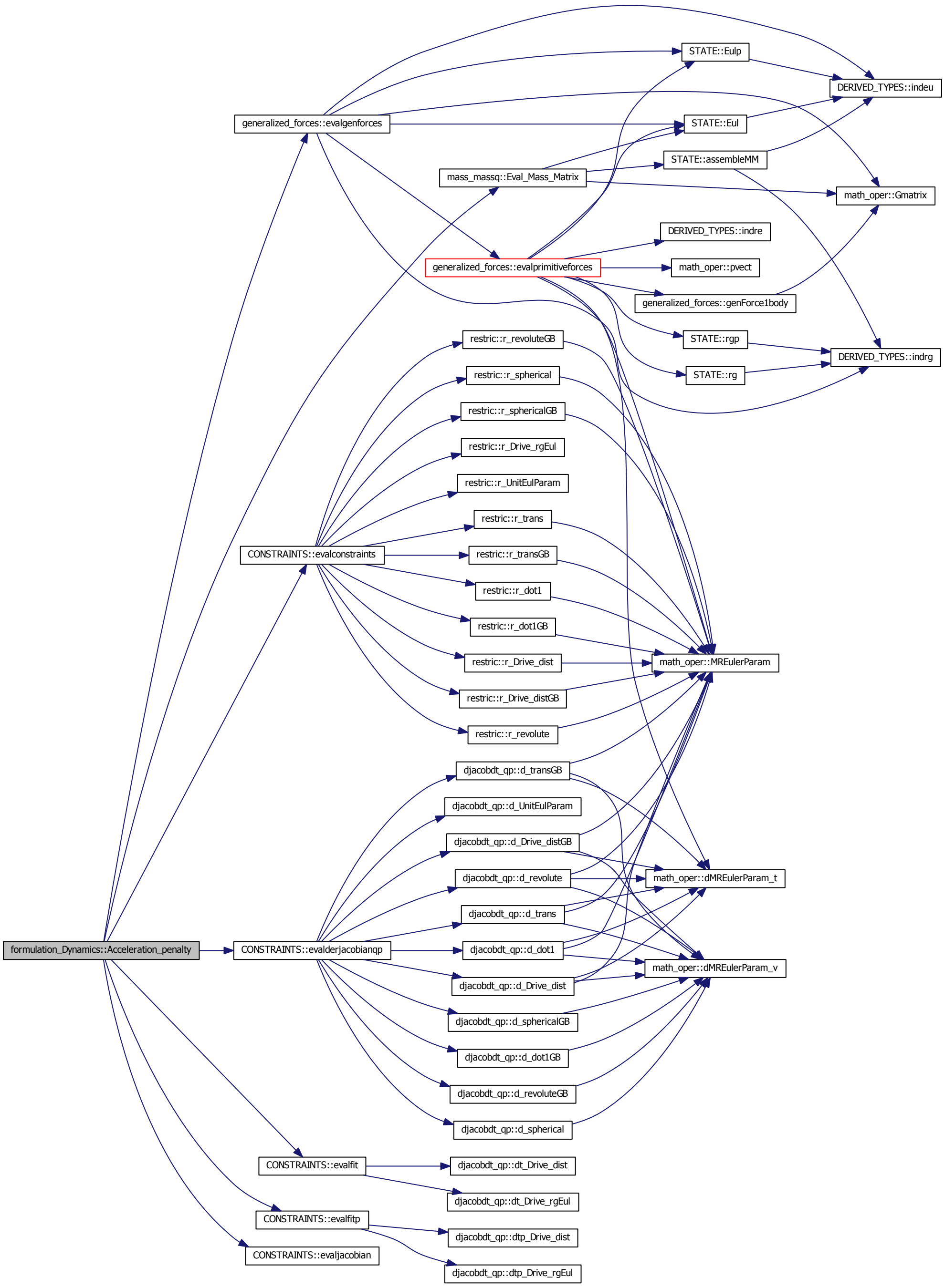
Dynamic simulation module.

### 4.8.2 Function/Subroutine Documentation

#### 4.8.2.1 subroutine formulation_Dynamics::Acceleration_penalty ( REAL(8),intent(in) *t* )

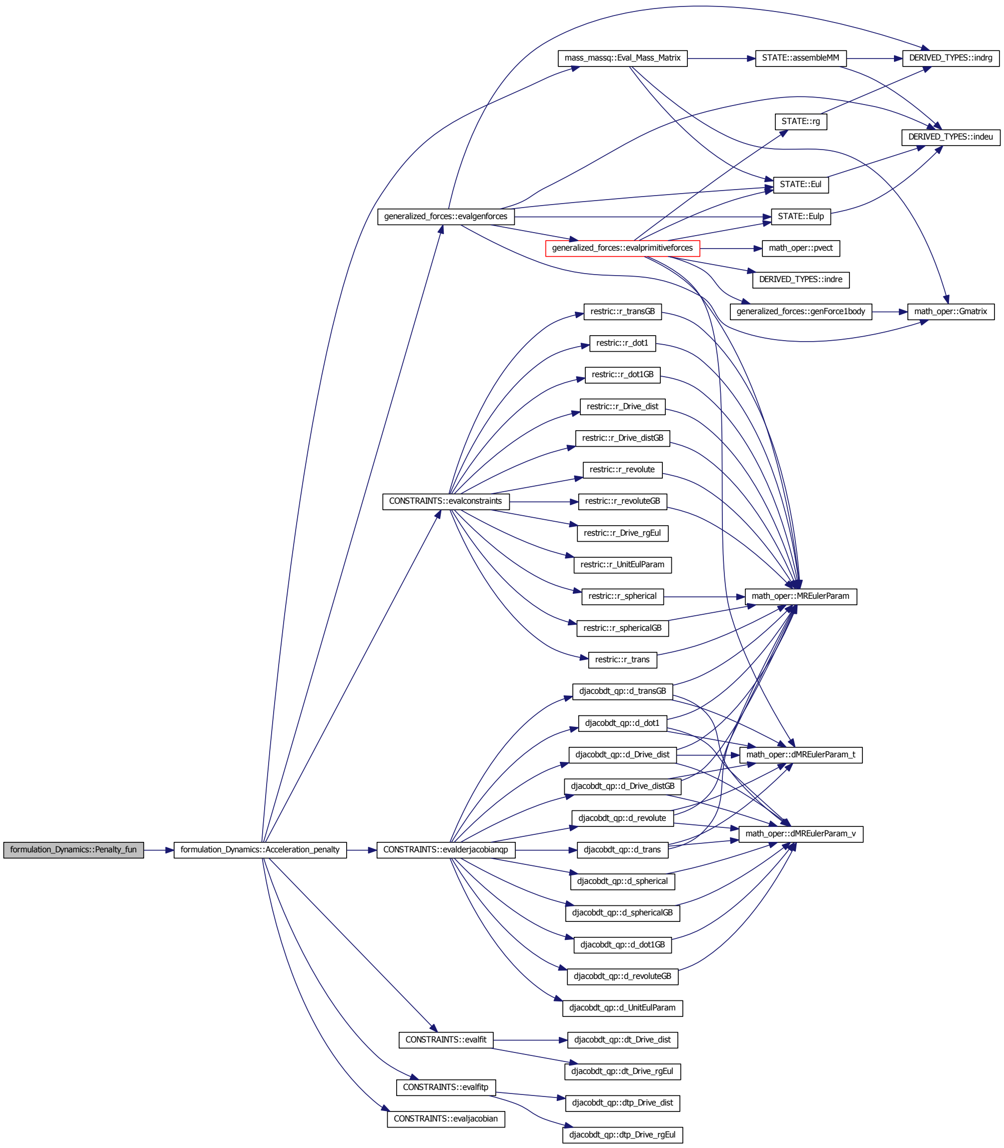
Subrutine that solves the equations of motion for the acceleration using penalty (Partially taken from MBSLIM)

Here is the call graph for this function:



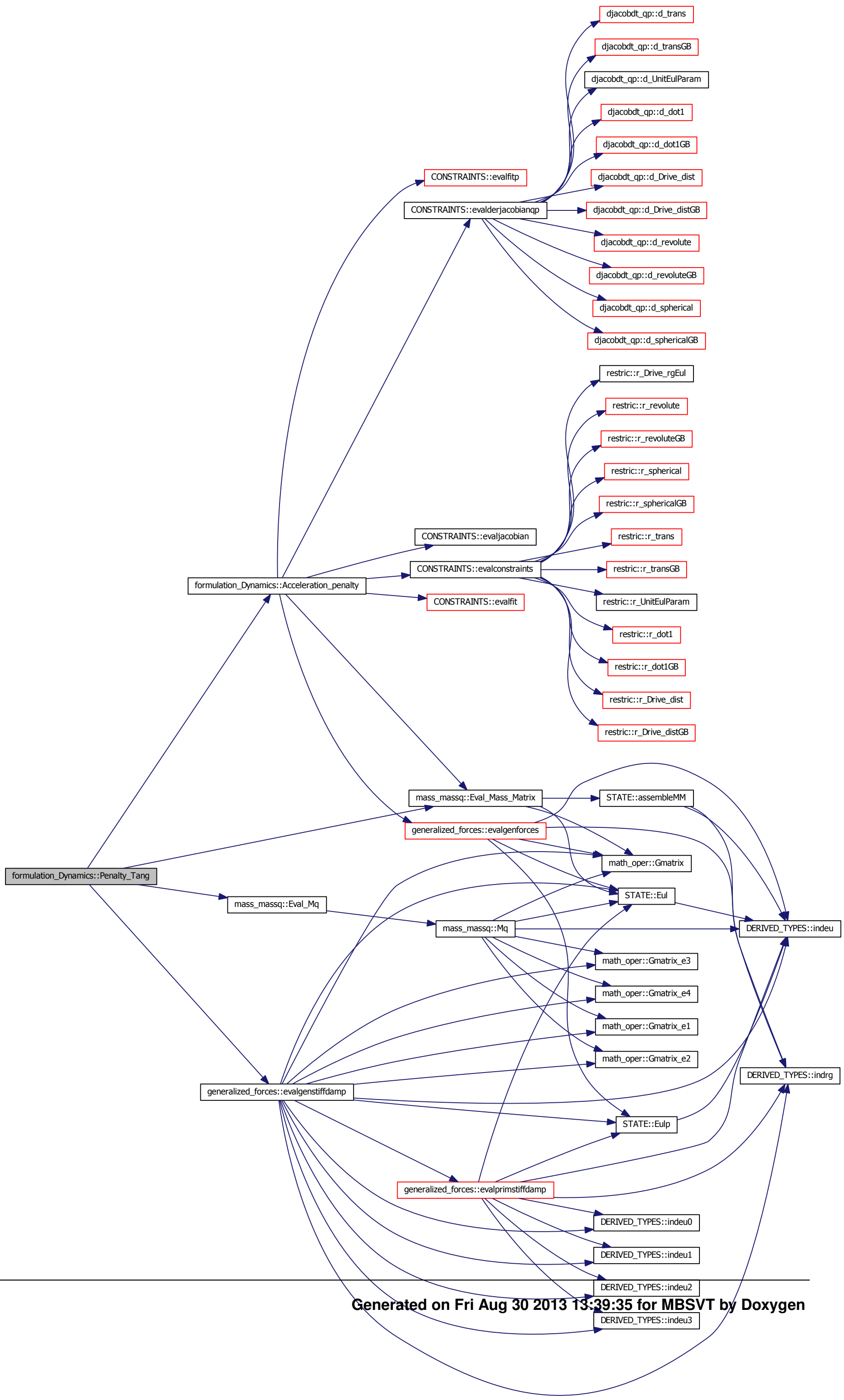
#### 4.8.2.2 subroutine formulation_Dynamics::Penalty_fun ( INTEGER,intent(in) *NVAR*, REAL(8),intent(in) *t*, REAL(8),dimension(nvar) *y*, REAL(8),dimension(nvar) *yp* )

Here is the call graph for this function:



#### 4.8.2.3 subroutine formulation_Dynamics::Penalty_Tang ( integer,intent(in) *N*, DOUBLE PRECISION,intent(in) *T*, DOUBLE PRECISION,dimension(n),intent(in) *Y*, DOUBLE PRECISION,dimension(n,n),intent(out) *Fy* )

Here is the call graph for this function:



## 4.9 formulation_Kinematics Module Reference

Kinematic simulation module.

### Functions/Subroutines

- subroutine position_kinematics (C, name)

  *Solves the position problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*

- subroutine velocity_kinematics (C, name)

  *Solves the velocity problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*

- subroutine acceleration_kinematics (C, name)

  *Solves the acceleration problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*

### 4.9.1 Detailed Description

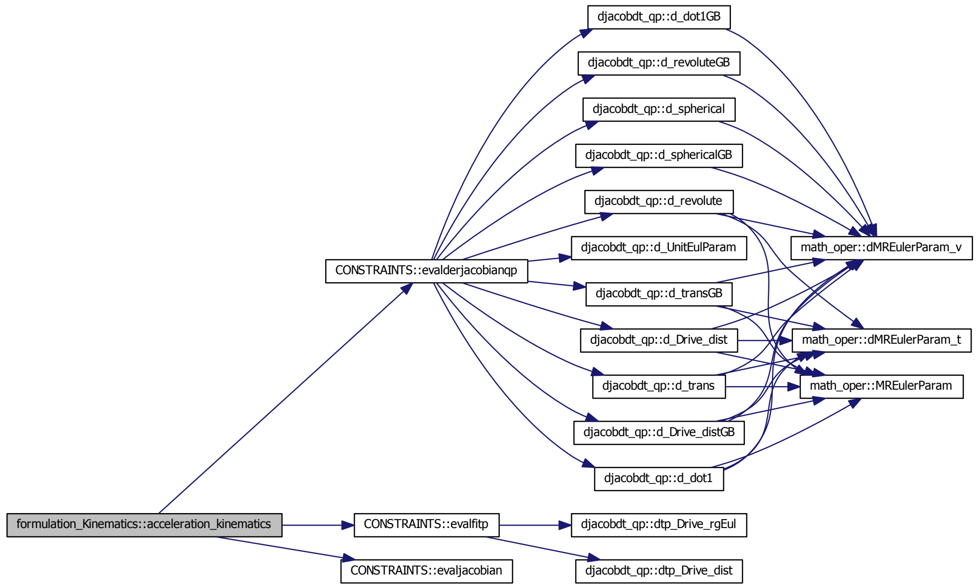
Kinematic simulation module.

### 4.9.2 Function/Subroutine Documentation

#### 4.9.2.1 subroutine formulation_Kinematics::acceleration_kinematics ( *BIND, C, name* )

Solves the acceleration problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)

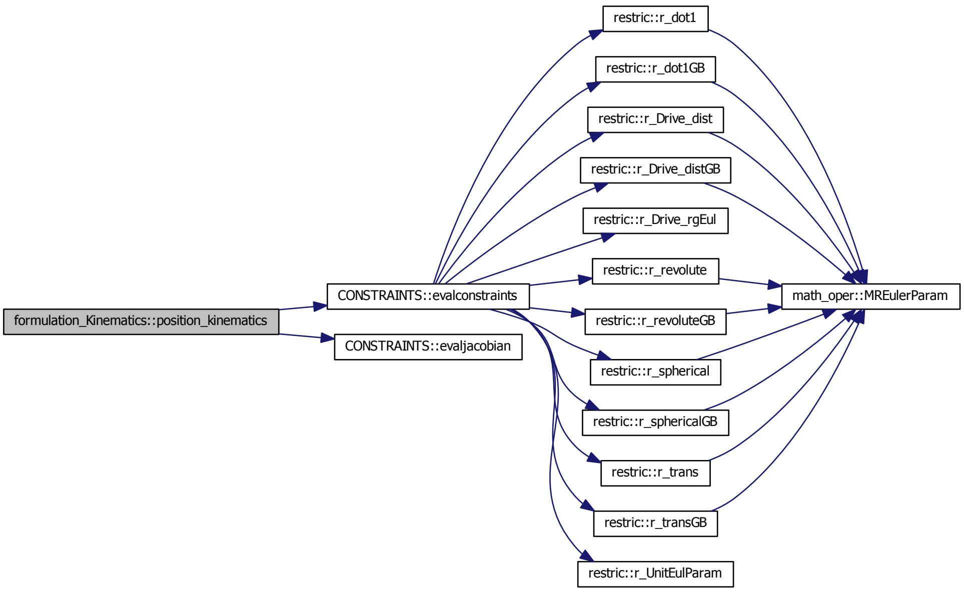
Here is the call graph for this function:




#### 4.9.2.2 subroutine formulation_Kinematics::position_kinematics ( *BIND*, *C*, *name* )

Solves the position problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)

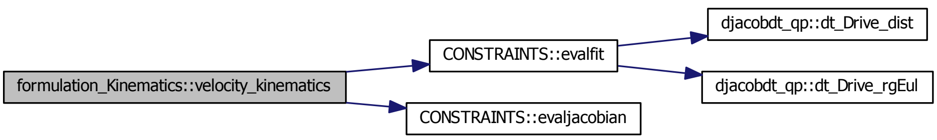
Here is the call graph for this function:




#### 4.9.2.3 subroutine formulation_Kinematics::velocity_kinematics ( *BIND,* *C,* *name* )

Solves the velocity problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)

Here is the call graph for this function:




## 4.10 formulation_Sensitivity Module Reference

Sensitivity analyis module.

### Functions/Subroutines

- subroutine Penalty_Jacp (N, NP, T, Y, FPJAC)

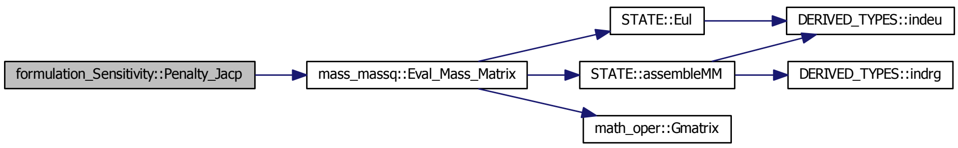
### 4.10.1 Detailed Description

Sensitivity analyis module.

### 4.10.2 Function/Subroutine Documentation

#### 4.10.2.1 subroutine formulation_Sensitivity::Penalty_Jacp ( integer *N*, integer *NP*, DOUBLE PRECISION *T*, DOUBLE PRECISION,dimension(n) *Y*, DOUBLE PRECISION,dimension(n,np) *FPJAC* )

Here is the call graph for this function:




## 4.11 formulations Module Reference

Module of generic formulations. Contains the generic functions that manage the use of different formulations.

### Functions/Subroutines

- subroutine acceleration_dynamics (t)

  *Generic subroutine for the acceleration calculation.*
- subroutine integration_dynamics (TIN, TOUT, RTOL, ATOL, POSTSTEP)

  *Generic subroutine for the integration of the equations of motion.*
- subroutine integration_sensitivity (NP, NADJ, NNZERO, VAR, Lambda, TIN, TOUT, ATOL_adj, RTOL_adj, ATOL, RTOL, Mu, objval)

  *Generic subroutine for the integration of the equations of motion.*
- subroutine Model_Setup

  *Generic subroutine to set up the models.*

### 4.11.1 Detailed Description

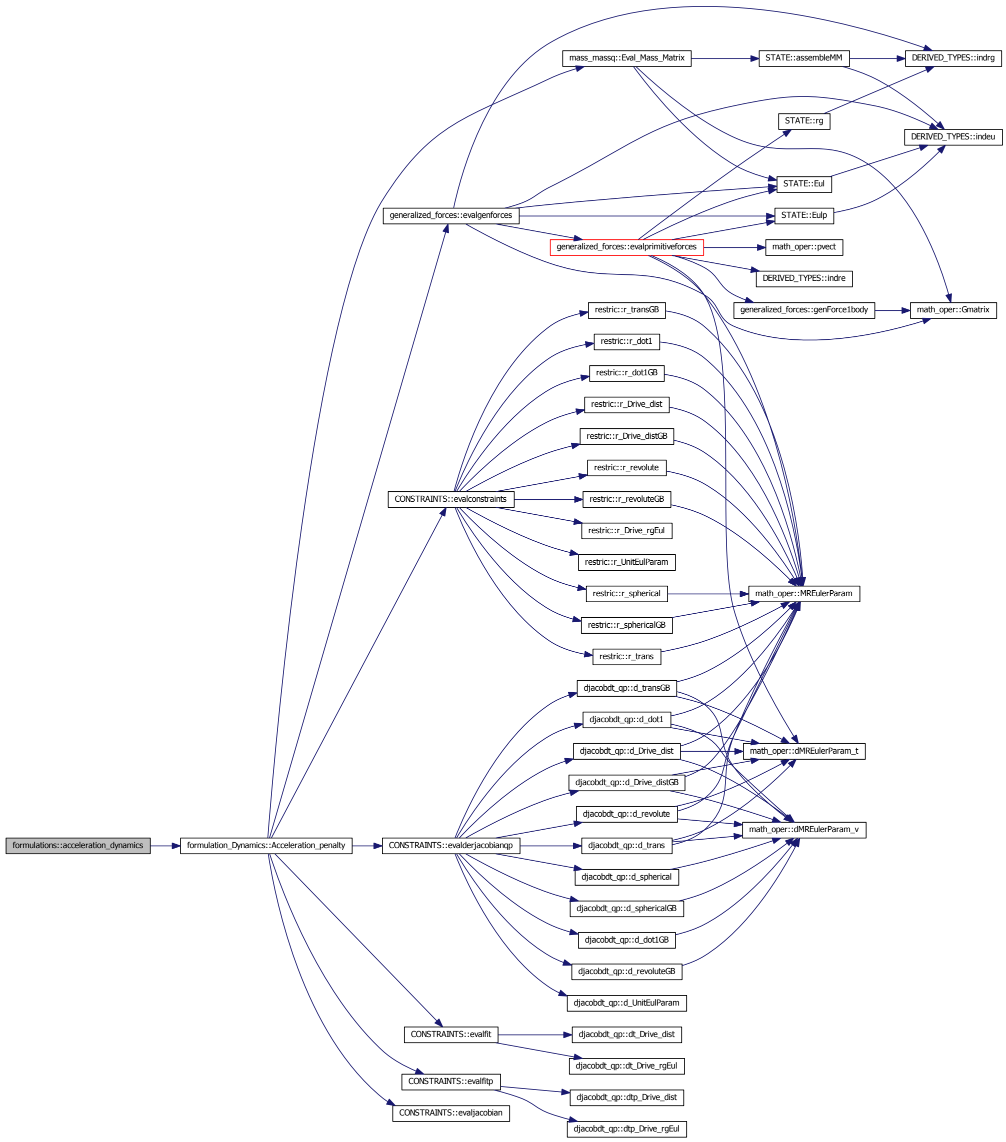
Module of generic formulations. Contains the generic functions that manage the use of different formulations.

### 4.11.2 Function/Subroutine Documentation

#### 4.11.2.1 subroutine formulations::acceleration_dynamics ( REAL(C_DOUBLE),intent(in) *t* )

Generic subroutine for the acceleration calculation.

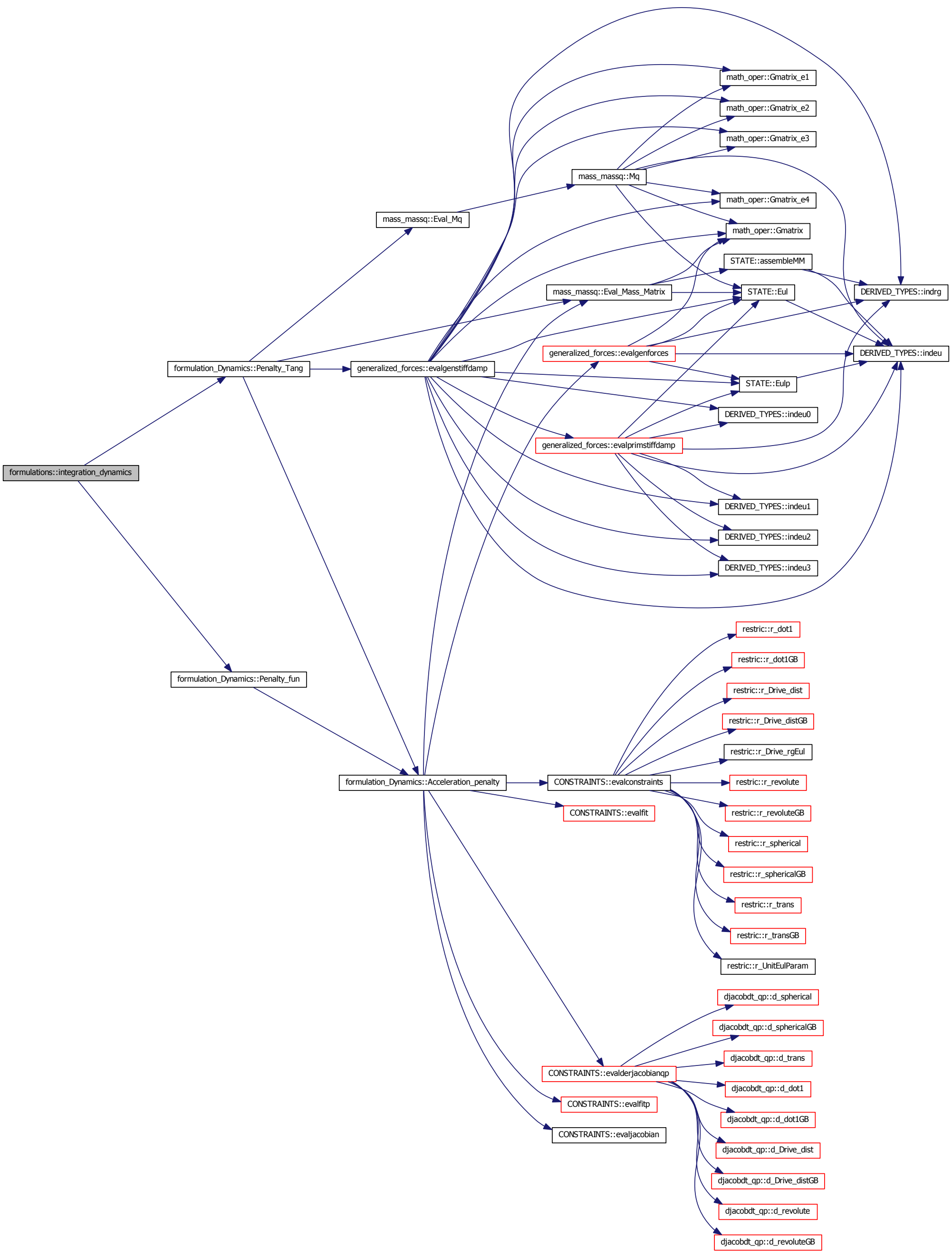
Here is the call graph for this function:




#### 4.11.2.2 subroutine formulations::integration_dynamics ( REAL(8),intent(in) *TIN*, REAL(8),intent(in) *TOUT*, REAL(8),dimension(2∗dim),intent(in) *RTOL*, REAL(8),dimension(2∗dim),intent(in) *ATOL*, ,optional,external *POSTSTEP* )

Generic subroutine for the integration of the equations of motion.

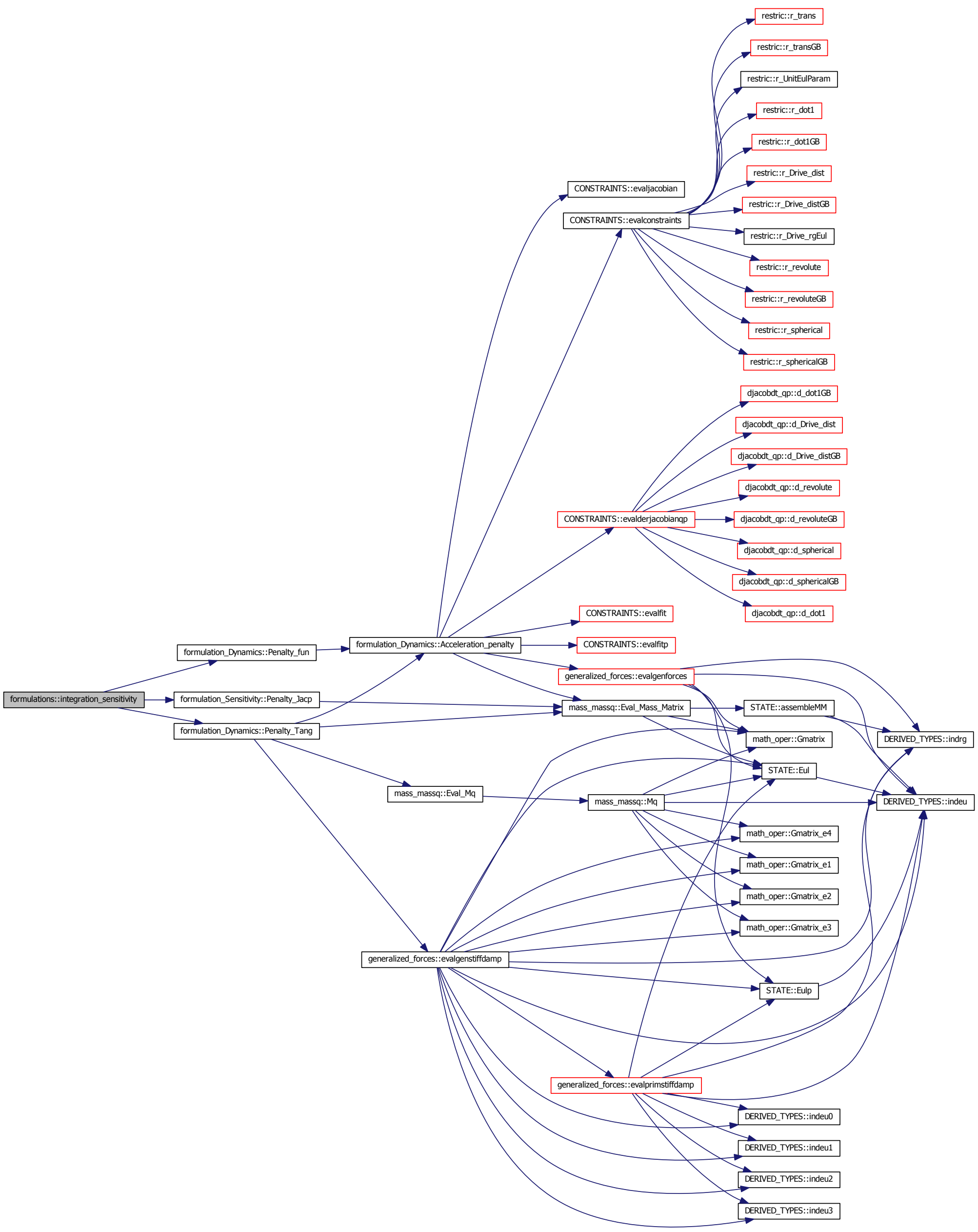
Here is the call graph for this function:



#### 4.11.2.3 subroutine formulations::integration_sensitivity ( integer *NP,* integer *NADJ,* integer *NNZERO,* real(8),dimension(2∗dim) *VAR,* real(8),dimension(2∗dim,nadj) *Lambda,* real(8) *TIN,* real(8) *TOUT,* real(8),dimension(2∗dim,nadj) *ATOL_adj,* real(8),dimension(2∗dim,nadj) *RTOL_adj,* real(8),dimension(2∗dim) *ATOL,* real(8),dimension(2∗dim) *RTOL,* real(8),dimension(np,nadj) *Mu,* real(8),dimension(nadj) *objval* )

Generic subroutine for the integration of the equations of motion.

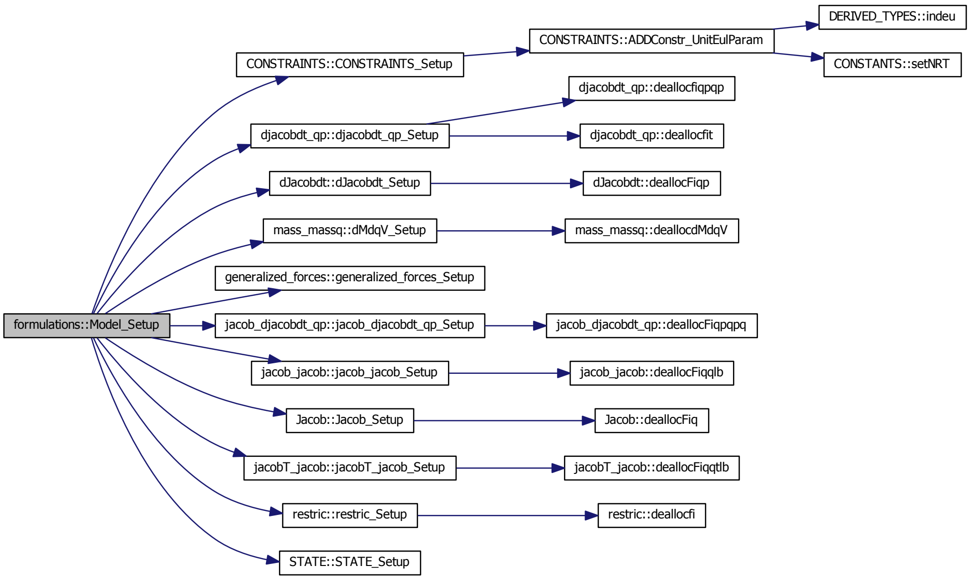
Here is the call graph for this function:




#### 4.11.2.4 subroutine formulations::Model_Setup ( )

Generic subroutine to set up the models.

Here is the call graph for this function:




## 4.12 generalized_forces Module Reference

Generalized forces module.

### Functions/Subroutines

- subroutine, public ADDforce_TSDA (body1, body2, pt1, pt2, k, c, s0)
- subroutine, public evalgenforces (t)

  *Subroutine to evaluate the generalized forces of the system.*
- subroutine, public evalgenstiffdamp (t)

  *Subroutine to evaluate the generalized stiffness and damping of the system.*
- subroutine evalprimitiveforces (t, Q_Prim)

  *Subroutine to evaluate the primitive forces of the system.*
- subroutine evalprimstiffdamp (t, K_Prim, C_Prim)

  *Subroutine to evaluate the primitive stiffness and damping of the system.*
- subroutine genForce1body (n, F, p, Q)

  *Subroutine to form the local generalized force over one body Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.*
- subroutine, public generalized_forces_Setup

  *Generalized forces module setup.*

## Variables

- REAL(8), dimension(:,:), allocatable, public PROTECTED
- REAL(8), dimension(:), allocatable, public Qgen
- REAL(8), dimension(:,:), allocatable, public Kgen
- REAL(8), dimension(:,:), allocatable, public Cgen
- REAL(8), dimension(:), allocatable Qgrav
- REAL(8), dimension(:,:), allocatable Kgrav
- REAL(8), dimension(:,:), allocatable Cgrav
- INTEGER nforce_TSDA = 0
- TYPE(typeforce_TSDA), dimension(:), allocatable force_TSDA

### 4.12.1 Detailed Description

Generalized forces module. This module:

1)Add forces to the model.

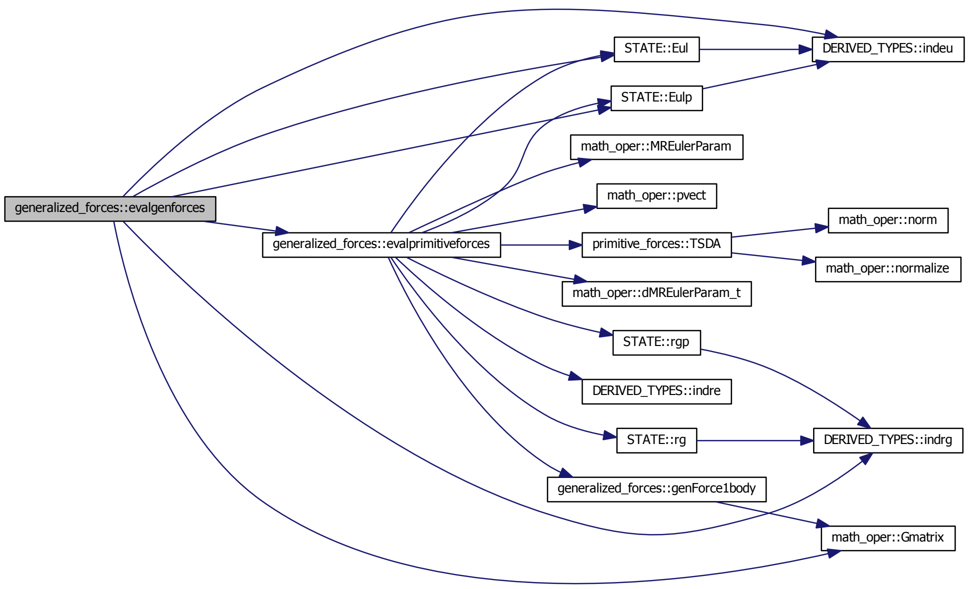
### 4.12.2 Function/Subroutine Documentation

#### 4.12.2.1 subroutine,public generalized_forces::ADDforce_TSDA ( INTEGER,intent(in) *body1*, INTEGER,intent(in) *body2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),intent(in) *k*, REAL(8),intent(in) *c*, REAL(8),intent(in) *s0* )

#### 4.12.2.2 subroutine,public generalized_forces::evalgenforces ( REAL(8),intent(in) *t* )

Subroutine to evaluate the generalized forces of the system.

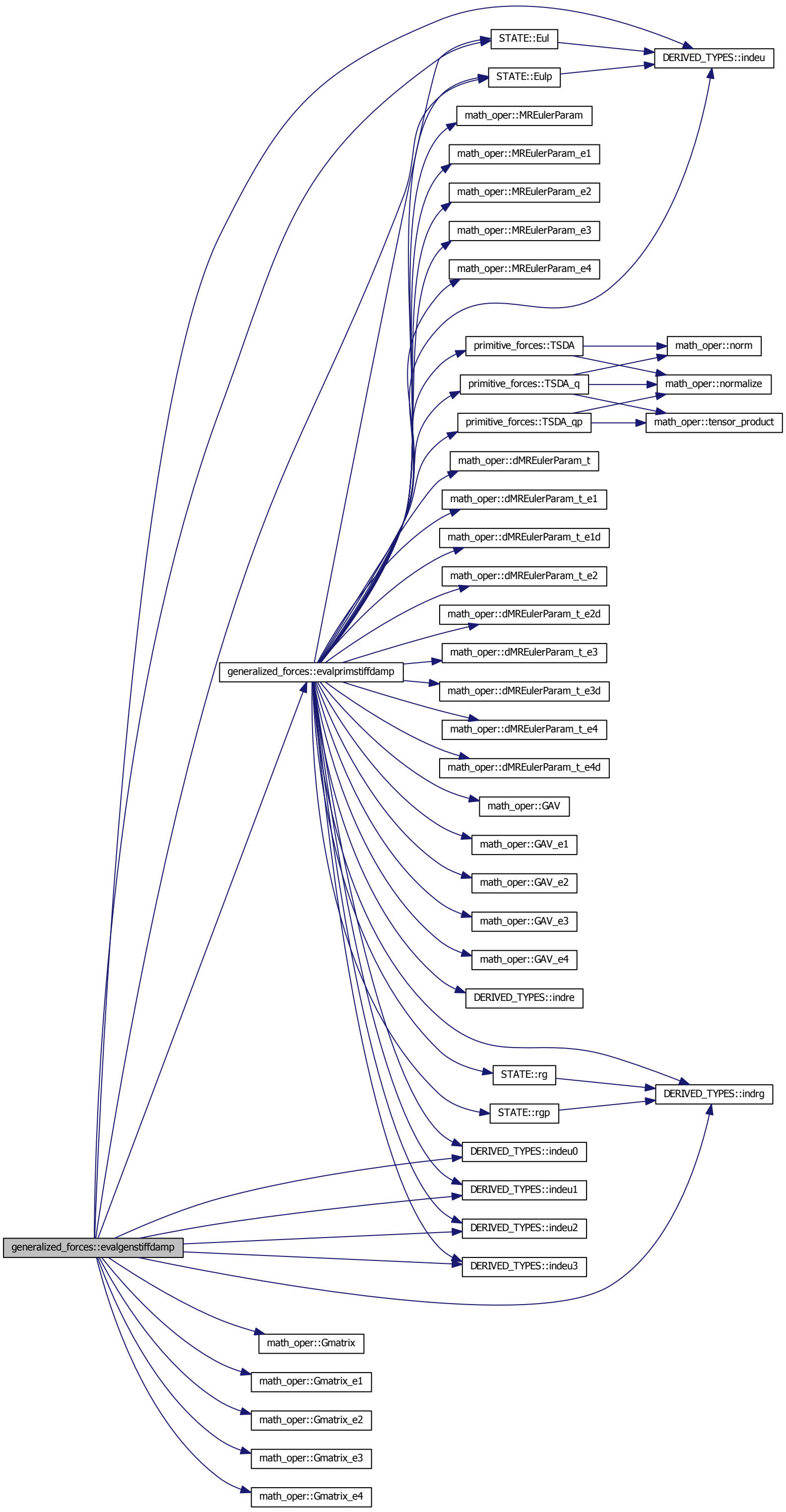
Here is the call graph for this function:




#### 4.12.2.3 subroutine,public generalized_forces::evalgenstiffdamp ( REAL(8),intent(in) *t* )

Subroutine to evaluate the generalized stiffness and damping of the system.

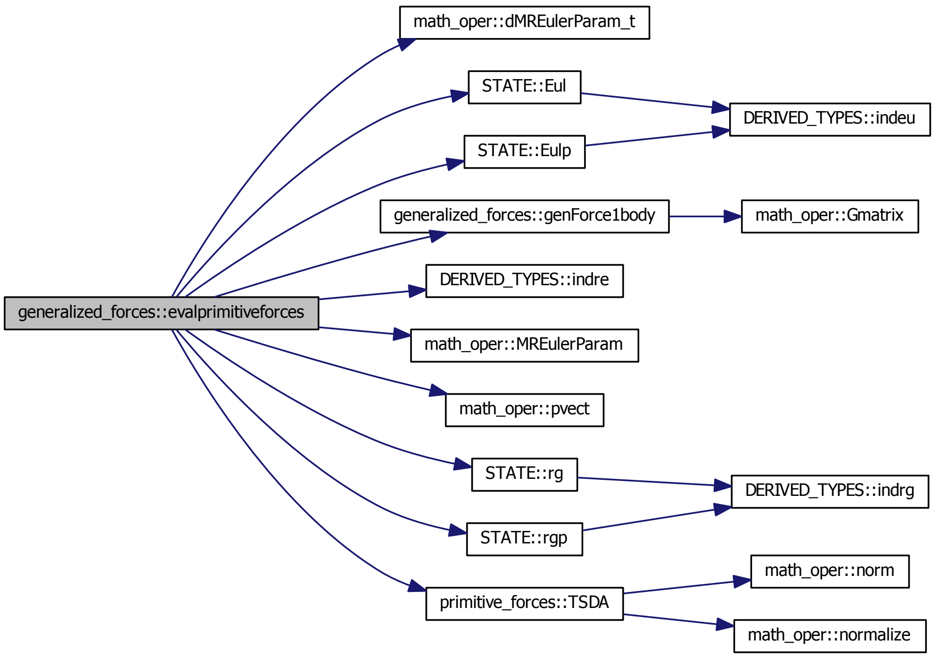
Here is the call graph for this function:



#### 4.12.2.4 subroutine generalized_forces::evalprimitiveforces ( REAL(8),intent(in) *t*, REAL(8),dimension(dim),intent(out) *Q_Prim* )

Subroutine to evaluate the primitive forces of the system.

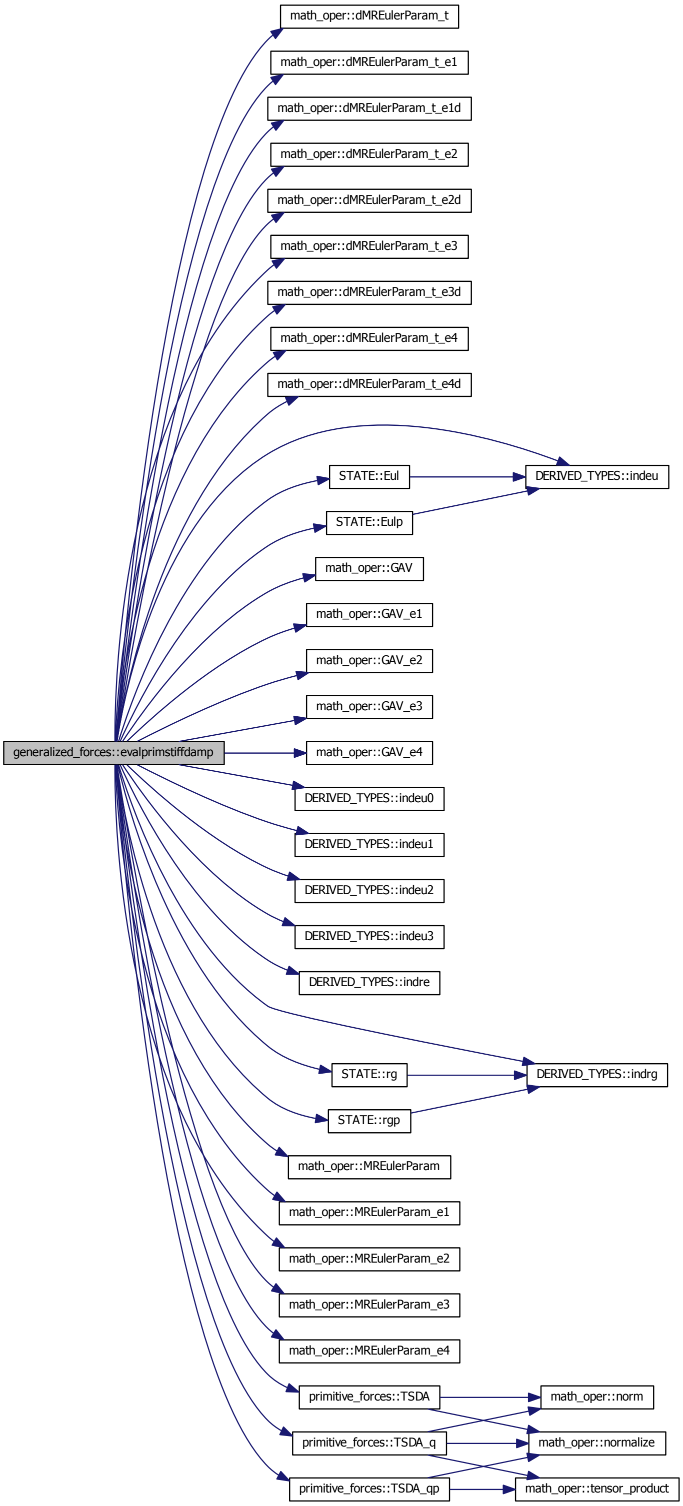
Here is the call graph for this function:




#### 4.12.2.5 subroutine generalized_forces::evalprimstiffdamp ( REAL(8),intent(in) *t*, REAL(8),dimension(dim,dim),intent(out) *K_Prim*, REAL(8),dimension(dim,dim),intent(out) *C_Prim* )

Subroutine to evaluate the primitive stiffness and damping of the system.

Here is the call graph for this function:



#### 4.12.2.6 subroutine,public generalized_forces::generalized_forces_Setup ( )

Generalized forces module setup.

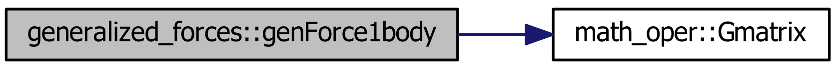
#### 4.12.2.7 subroutine generalized_forces::genForce1body ( REAL(8),dimension(3),intent(in) *n,* REAL(8),dimension(3),intent(in) *F,* REAL(8),dimension(4),intent(in) *p,* REAL(8),dimension(7),intent(out) *Q* )

Subroutine to form the local generalized force over one body Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *n* | torque act on the given body. |
| *F* | force act on the given body. |
| *P* | Euler parameters of the given body. |
| *Q* | Euler parameters of the given body. |

Here is the call graph for this function:




### 4.12.3 Variable Documentation

#### 4.12.3.1 REAL(8),dimension(:,:),allocatable,public generalized_forces::Cgen

#### 4.12.3.2 REAL(8),dimension(:,:),allocatable generalized_forces::Cgrav

#### 4.12.3.3 TYPE(typeforce_TSDA),dimension(:),allocatable generalized_forces::force_TSDA

#### 4.12.3.4 REAL(8),dimension(:,:),allocatable,public generalized_forces::Kgen

#### 4.12.3.5 REAL(8),dimension(:,:),allocatable generalized_forces::Kgrav

#### 4.12.3.6 INTEGER generalized_forces::nforce_TSDA = 0

#### 4.12.3.7 REAL(8),dimension(:,:),allocatable,public generalized_forces::PROTECTED

#### 4.12.3.8 REAL(8),dimension(:),allocatable,public generalized_forces::Qgen

#### 4.12.3.9 REAL(8),dimension(:),allocatable generalized_forces::Qgrav

## 4.13 Jacob Module Reference

Module of primitive jacobians. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine Jacob_Setup
- subroutine deallocFiq
- subroutine j_UnitEulParam (ir, iEul)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC Primitive jacobian of unitary Euler parameters.*
- subroutine j_dot1GB (ir, iEul2, u, v)

  *Primitive dot-1 jacobian of a body attached on the ground.*
- subroutine j_dot1 (ir, iEul1, iEul2, u, v)

  *Primitive dot-1 jacobian.*
- subroutine j_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *Primitive jacobians of a spherical joint of a body attached to the ground.*
- subroutine j_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *Primitive jacobians of a spherical joint between two bodies.*
- subroutine j_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *Primitive jacobians of a revolute joint of a body attached to the ground.*
- subroutine j_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *Primitive jacobians of a revolute joint between two bodies.*
- subroutine j_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive jacobians of a translational joint of a body attached to the ground.*
- subroutine j_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive jacobians of a translational joint between two bodies.*
- subroutine j_Drive_rgEul (ir, ind, i_MOTOR)

  *Primitive driving jacobians for a generalized coordinate of the system.*
- subroutine j_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *Primitive driving jacobians for a distance between a point in the ground and a point of one body.*
- subroutine j_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *Primitive driving constraints for a distance between two points of two bodies.*

## Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiq

### 4.13.1 Detailed Description

Module of primitive jacobians. It's NOT a user module, it's used by the solver.

### 4.13.2 Function/Subroutine Documentation

#### 4.13.2.1 subroutine Jacob::deallocFiq ( )

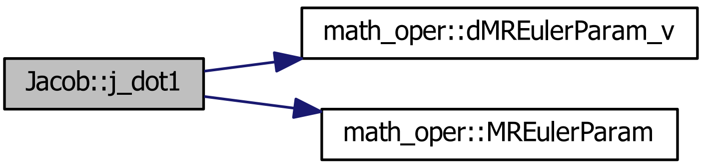
#### 4.13.2.2 subroutine Jacob::j_dot1 ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

Primitive dot-1 jacobian.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:




#### 4.13.2.3 subroutine Jacob::j_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

Primitive dot-1 jacobian of a body attached on the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:

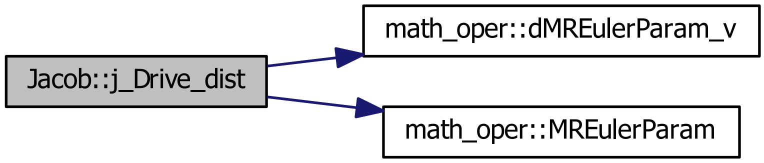

#### 4.13.2.4 subroutine Jacob::j_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

Primitive driving constraints for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



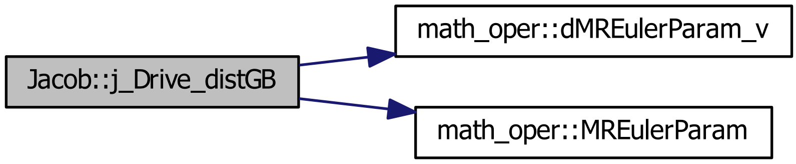

#### 4.13.2.5 subroutine Jacob::j_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

Primitive driving jacobians for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



#### 4.13.2.6 subroutine Jacob::j_Drive_rgEul ( INTEGER,intent(in) *ir,* INTEGER,intent(in) *ind,* INTEGER,intent(in) *i_MOTOR* )

Primitive driving jacobians for a generalized coordinate of the system.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *ind* | index of the driven generalized coordinate. |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. It is not necessary here, but it is kept for compatibility of the interfaces (less easy to make mistakes) |

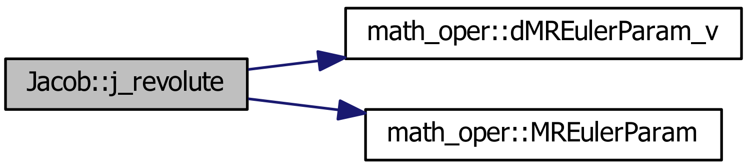
#### 4.13.2.7 subroutine Jacob::j_revolute ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg1,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *u1,* REAL(8),dimension(3),intent(in) *v1,* REAL(8),dimension(3),intent(in) *vec2* )

Primitive jacobians of a revolute joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



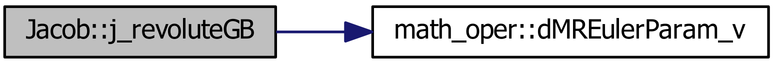
#### 4.13.2.8 subroutine Jacob::j_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2* )

Primitive jacobians of a revolute joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,u2* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |

Here is the call graph for this function:



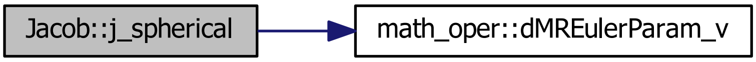

#### 4.13.2.9 subroutine Jacob::j_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

Primitive jacobians of a spherical joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |

Here is the call graph for this function:



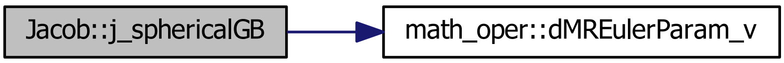

#### 4.13.2.10 subroutine Jacob::j_sphericalGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

Primitive jacobians of a spherical joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

Here is the call graph for this function:



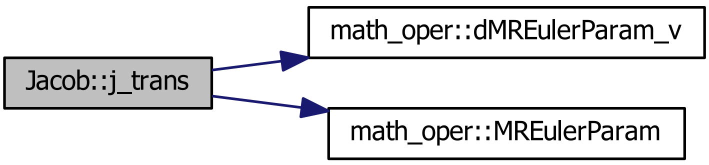

#### 4.13.2.11 subroutine Jacob::j_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

Primitive jacobians of a translational joint between two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



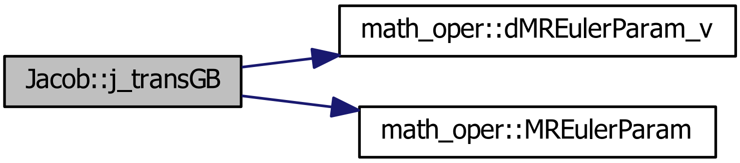

#### 4.13.2.12 subroutine Jacob::j_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

Primitive jacobians of a translational joint of a body attached to the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:




#### 4.13.2.13 subroutine Jacob::j_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul* )

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Primitive jacobian of unitary Euler parameters.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |

#### 4.13.2.14 subroutine Jacob::Jacob_Setup ( )

Here is the call graph for this function:


### 4.13.3 Variable Documentation

#### 4.13.3.1 REAL(8),dimension(:,:),allocatable Jacob::Fiq

#### 4.13.3.2 REAL(8),dimension(:,:),allocatable Jacob::PROTECTED

## 4.14 jacob_djacobdt_qp Module Reference

Module of $(\dot{\Phi}_q \dot{q})_q$. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine jacob_djacobdt_qp_Setup
- subroutine deallocFiqpqpq
- subroutine dq_dot1 (ir, iEul1, iEul2, u, v)

  *$\dot{\Phi}q\dot{q}$ of a dot-1 constraint.*
- subroutine dq_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *$(\blacksquare q\dot{q})q$ of a revolute joint between two bodies*
- subroutine dq_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *$(\blacksquare q\dot{q})q$ of a translational joint of a body attached to the ground*
- subroutine dq_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *$(\blacksquare q\dot{q})q$ of a translational joint between two bodies*
- subroutine dq_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *$\dot{\Phi}q\dot{q}$ for a distance between a point in the ground and a point of one body.*
- subroutine dq_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *$\dot{\Phi}q\dot{q}$ for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiqpqpq

### 4.14.1 Detailed Description

Module of $(\dot{\Phi}_q \dot{q})_q$. It's NOT a user module, it's used by the solver.

### 4.14.2 Function/Subroutine Documentation

#### 4.14.2.1 subroutine jacob_djacobdt_qp::deallocFiqpqpq ( )

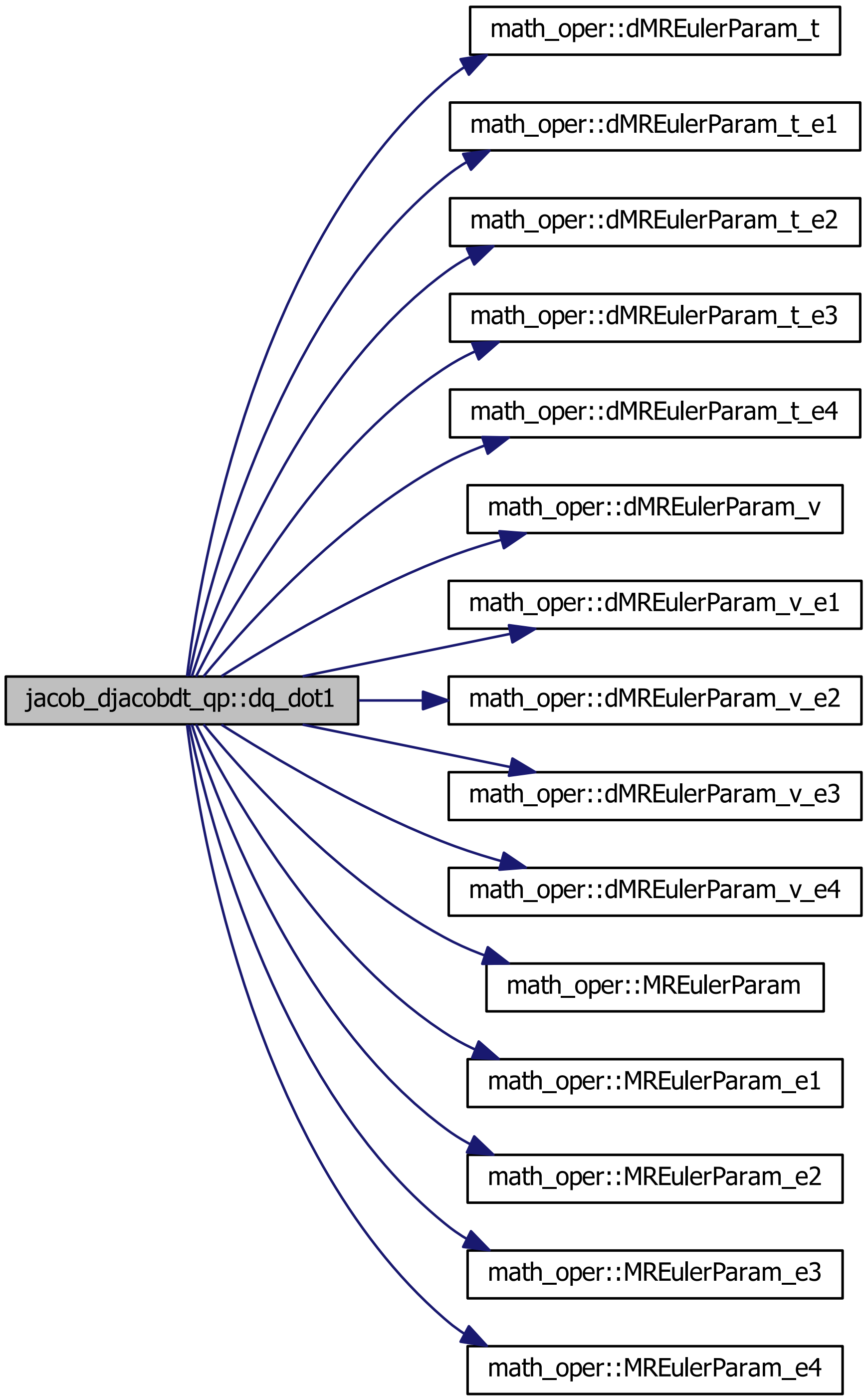
#### 4.14.2.2 subroutine jacob_djacobdt_qp::dq_dot1 ( integer,intent(in) *ir,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* real(8),dimension(3),intent(in) *u,* real(8),dimension(3),intent(in) *v* )

$\dot{\Phi}q\dot{q}$ of a dot-1 constraint.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



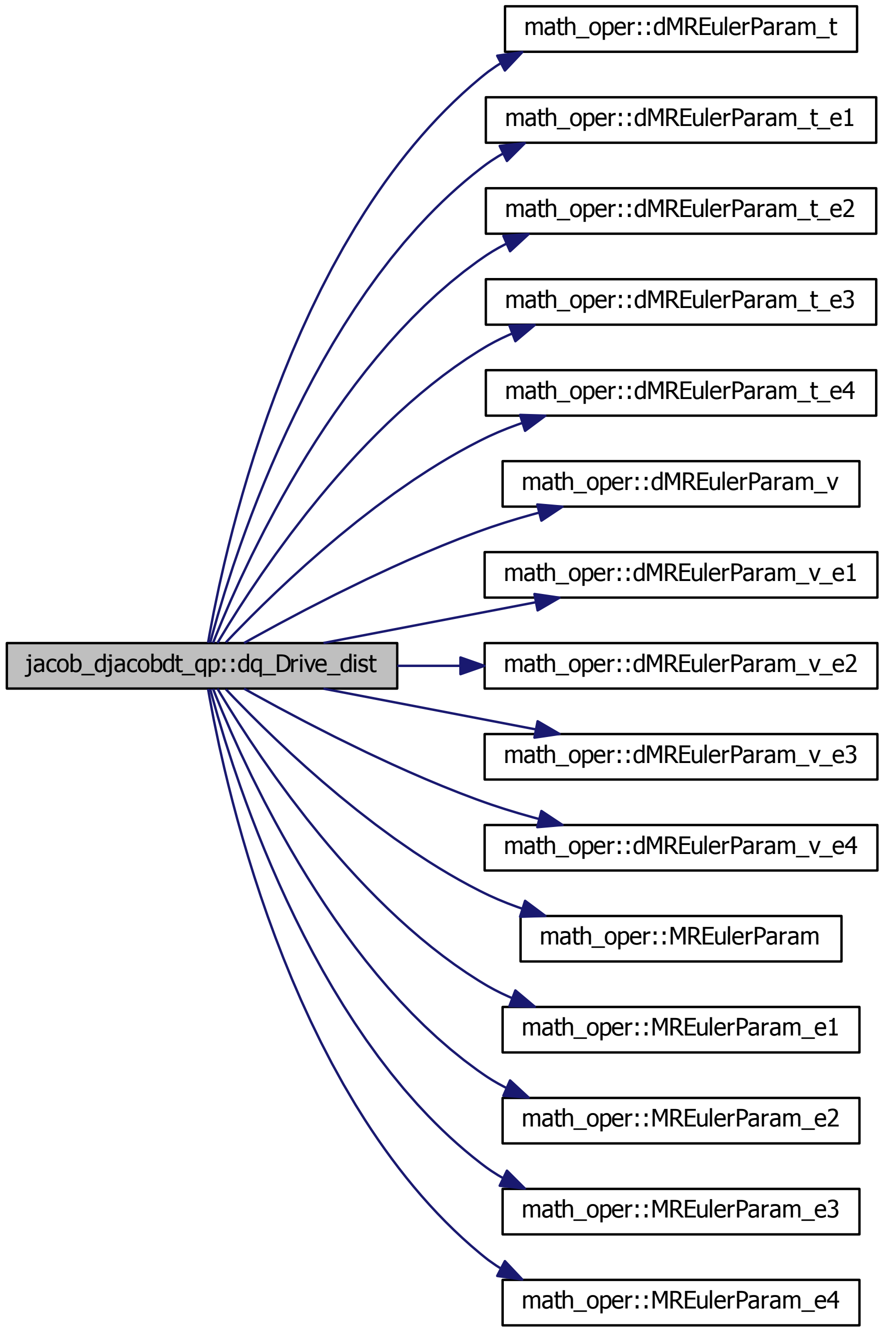
#### 4.14.2.3 subroutine jacob_djacobdt_qp::dq_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}\mathbf{q}\dot{\mathbf{q}}$ for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



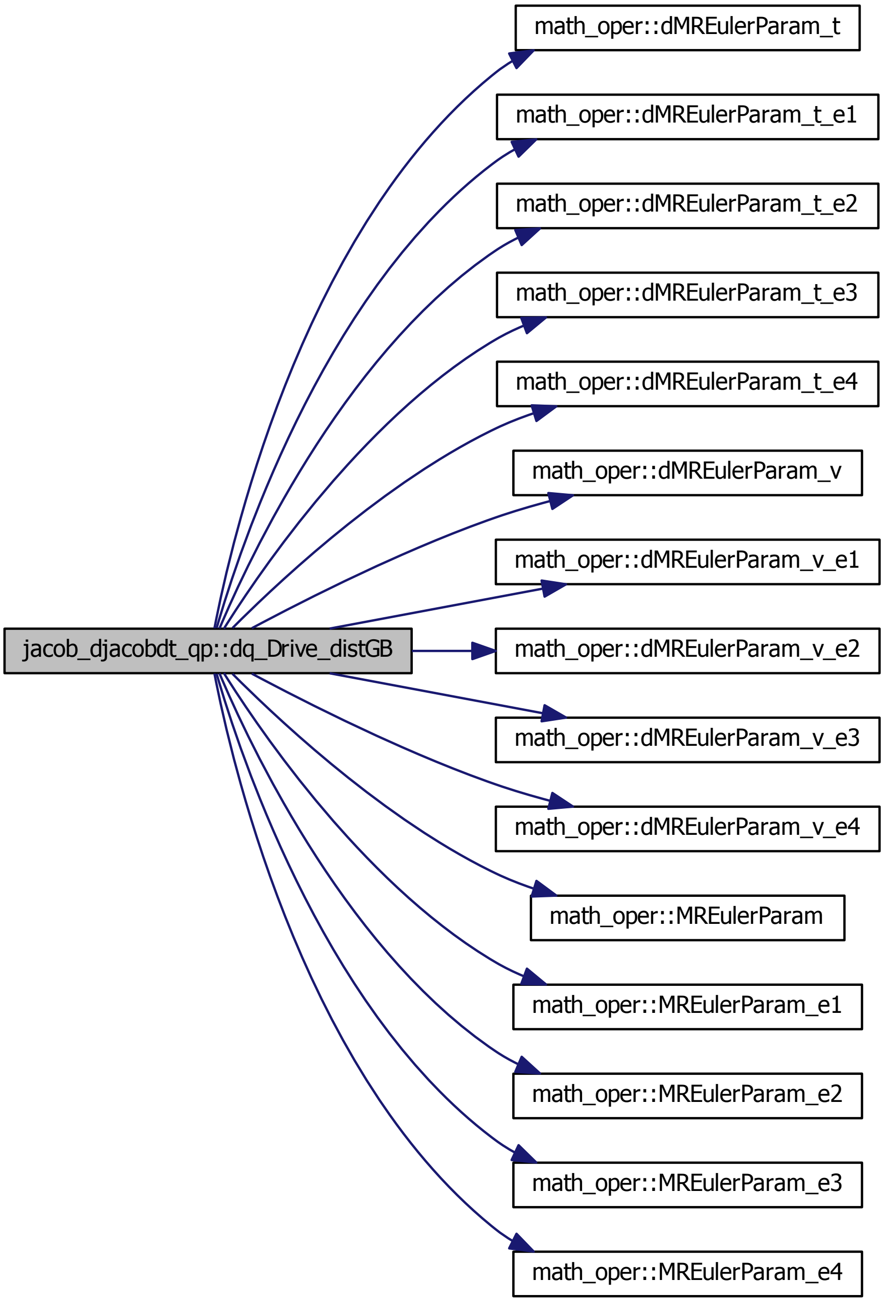
#### 4.14.2.4 subroutine jacob_djacobdt_qp::dq_Drive_distGB ( INTEGER,intent(in) *ir,* INTEGER,dimension(3),intent(in) *irg2,* INTEGER,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2_loc,* INTEGER,intent(in) *i_MOTOR* )

$\dot{\mathbf{\Phi}}\mathbf{q}\dot{\mathbf{q}}$ for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



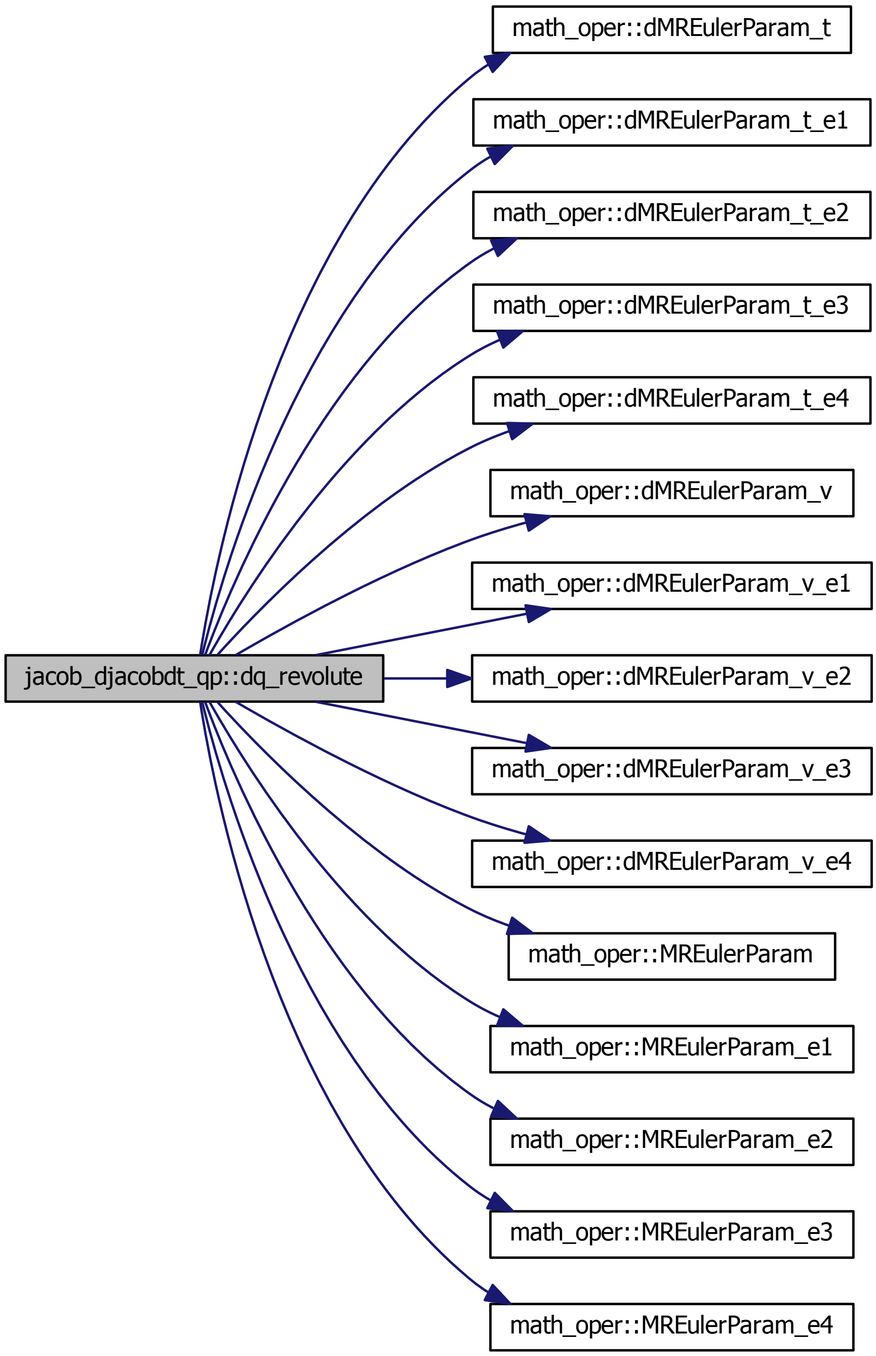
#### 4.14.2.5 subroutine jacob_djacobdt_qp::dq_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in),optional *irg1*, integer,dimension(3),intent(in),optional *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, real(8),dimension(3),intent(in) *u1*, real(8),dimension(3),intent(in) *v1*, real(8),dimension(3),intent(in) *vec2* )

$(\blacksquare \mathbf{q}\dot{\mathbf{q}})\mathbf{q}$ of a revolute joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



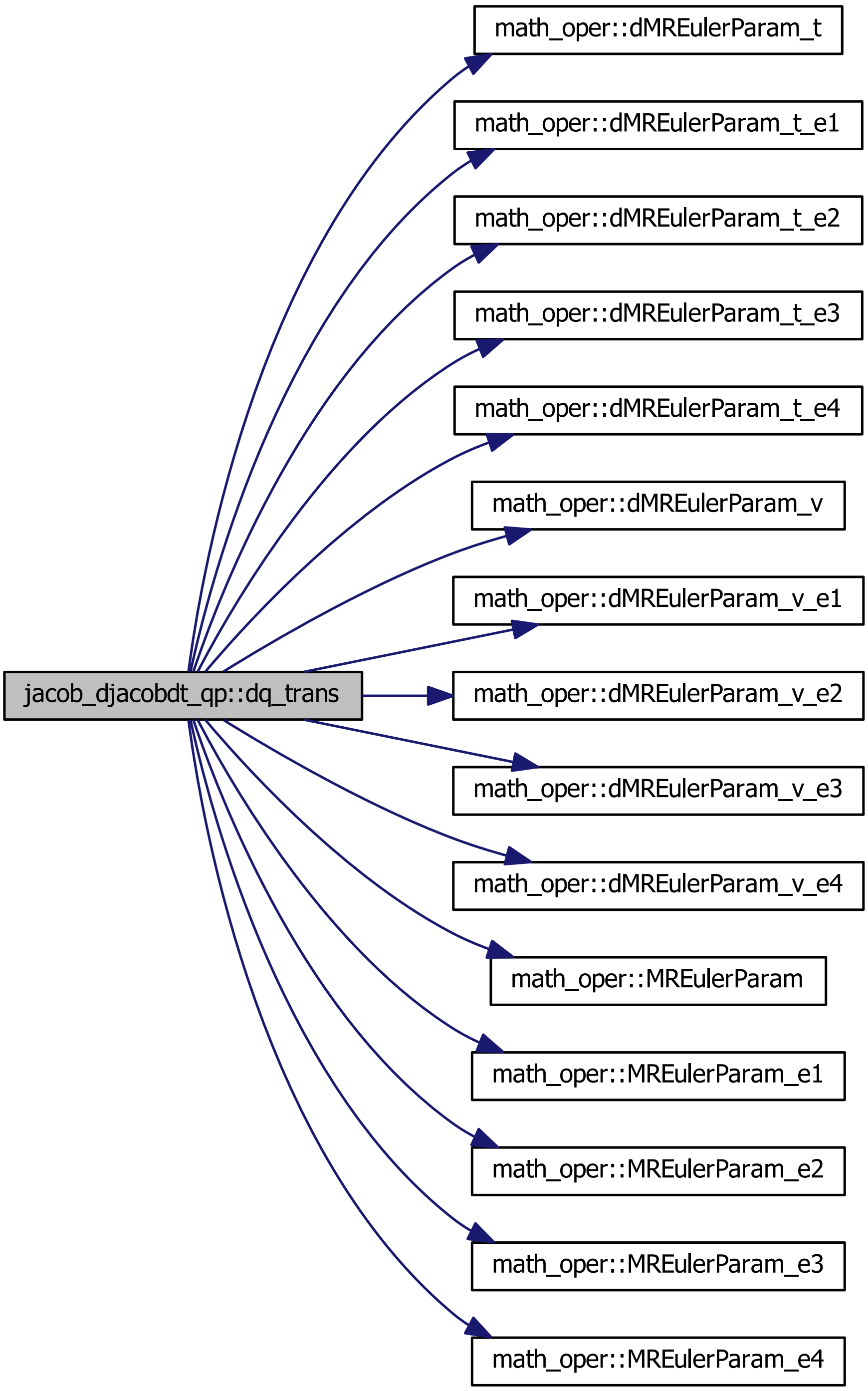
#### 4.14.2.6 subroutine jacob_djacobdt_qp::dq_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

$(\blacksquare_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}$ of a translational joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



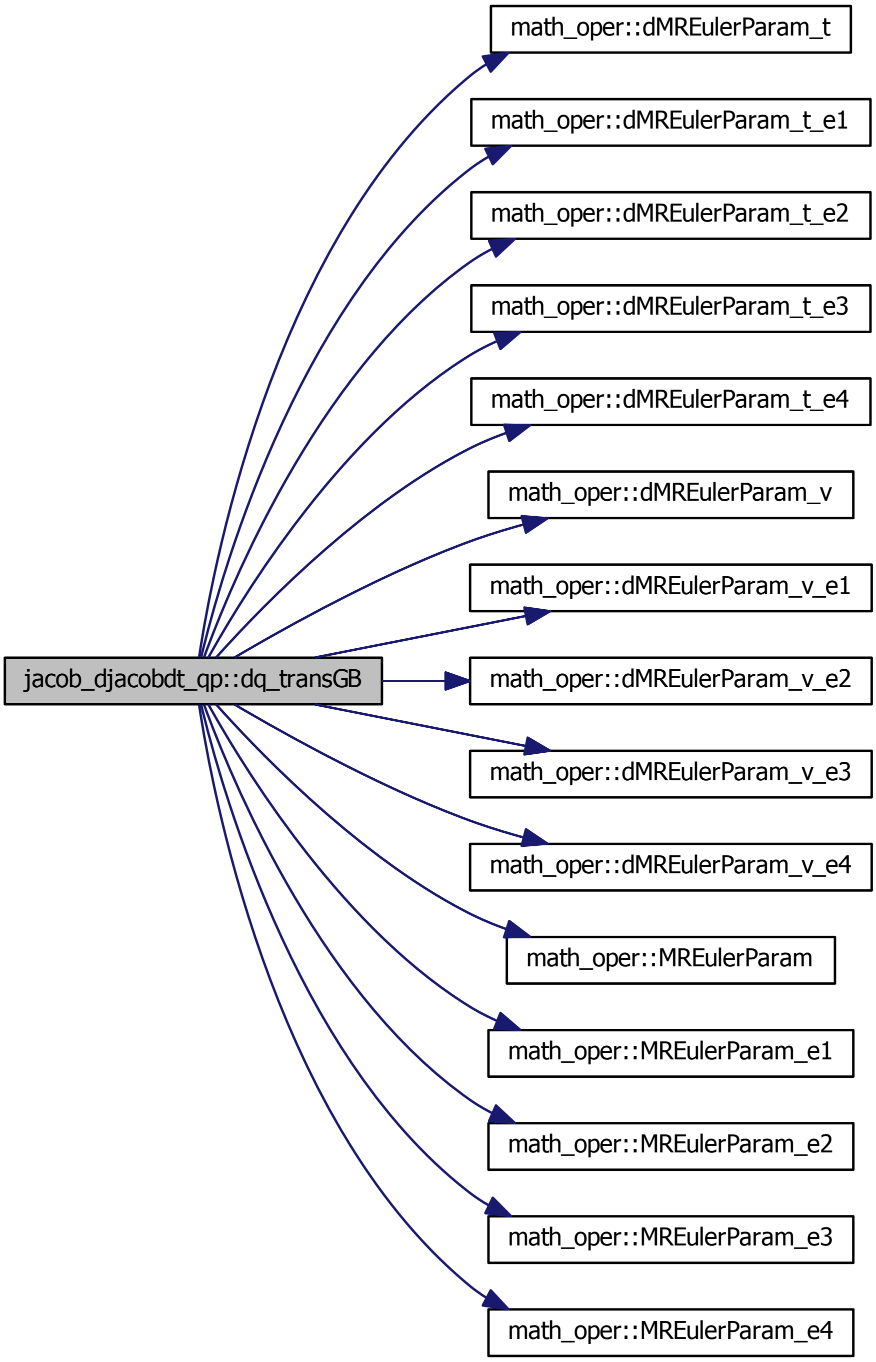
#### 4.14.2.7 subroutine jacob_djacobdt_qp::dq_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

$(\blacksquare \mathbf{q}\dot{\mathbf{q}})\mathbf{q}$ of a translational joint of a body attached to the ground

**Parameters**

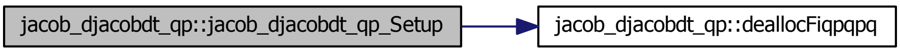
| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:



#### 4.14.2.8 subroutine jacob_djacobdt_qp::jacob_djacobdt_qp_Setup ( )

Here is the call graph for this function:




### 4.14.3 Variable Documentation

#### 4.14.3.1 REAL(8),dimension(:,:),allocatable jacob_djacobdt_qp::Fiqpqpq

#### 4.14.3.2 REAL(8),dimension(:,:),allocatable jacob_djacobdt_qp::PROTECTED

## 4.15 jacob_jacob Module Reference

Module of $\mathbf{\Phi_{qq}V}$, which is the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine jacob_jacob_Setup
- subroutine deallocFiqqlb
- subroutine jjlb_UnitEulParam (ir, iEul, lb)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*
  $\mathbf{\Phi_{qq}V}$ *of unitary Euler parameters.*
- subroutine jjlb_dot1GB (ir, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}V}$ *of a dot-1 jacobian of a body attached on the ground*
- subroutine jjlb_dot1 (ir, iEul1, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}V}$*of a dot-1 jacobian*
- subroutine jjlb_sphericalGB (ir, irg2, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi_{qq}V}$ *of a spherical joint of a body attached to the ground*
- subroutine jjlb_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi_{qq}V}$ *of a spherical joint between two bodies*
- subroutine jjlb_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi_{qq}V}$ *of a revolute joint of a body attached to the ground*
- subroutine jjlb_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi_{qq}V}$ *of a revolute joint between two bodies*
- subroutine jjlb_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi_{qq}V}$ *of a translational joint of a body attached to the ground*

- subroutine jjlb_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

    $\mathbf{\Phi_{qq}V}$ *of a translational joint between two bodies*
- subroutine jjlb_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR, lb)

    $\mathbf{\Phi_{qq}V}$ *of a distance driving jacobians between a point in the ground and a point of one body.*
- subroutine jjlb_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR, lb)

    *Primitive driving constraints for a distance between two points of two bodies.*

## Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiqqlb

### 4.15.1 Detailed Description

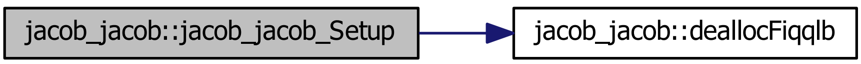
Module of $\mathbf{\Phi_{qq}V}$, which is the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.

### 4.15.2 Function/Subroutine Documentation

#### 4.15.2.1 subroutine jacob_jacob::deallocFiqqlb ( )

#### 4.15.2.2 subroutine jacob_jacob::jacob_jacob_Setup ( )

Here is the call graph for this function:



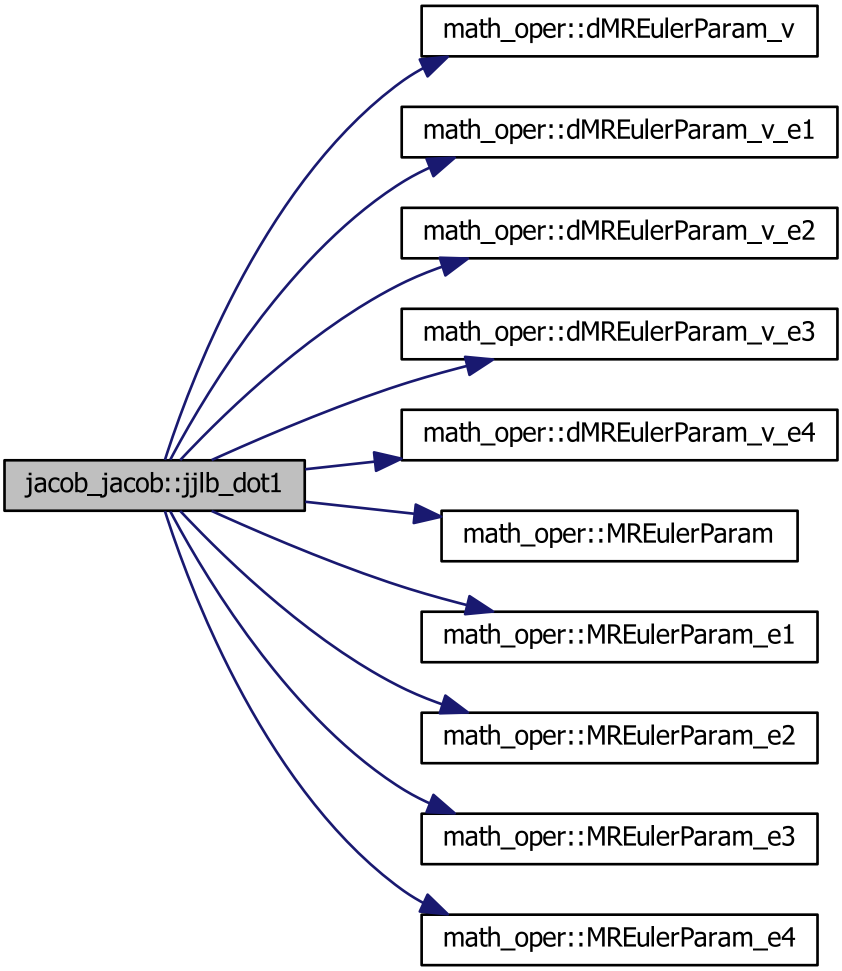

#### 4.15.2.3 subroutine jacob_jacob::jjlb_dot1 ( integer,intent(in) *ir,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* real(8),dimension(3),intent(in) *u,* real(8),dimension(3),intent(in) *v,* REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$of a dot-1 jacobian

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



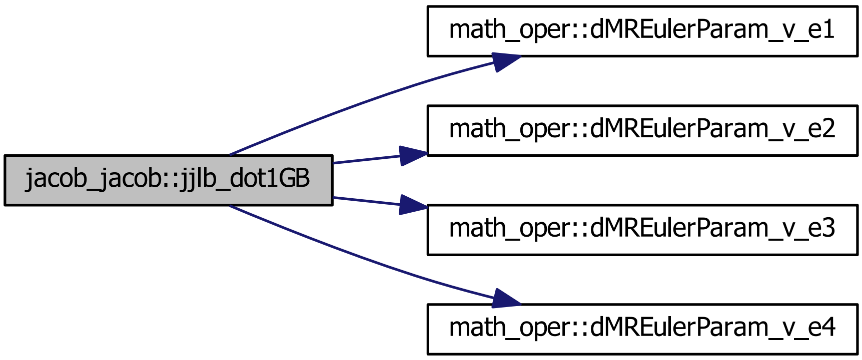
#### 4.15.2.4 subroutine jacob_jacob::jjlb_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v*, REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$ of a dot-1 jacobian of a body attached on the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:




#### 4.15.2.5 subroutine jacob_jacob::jjlb_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR*, REAL(8),dimension(dim),intent(in) *lb* )

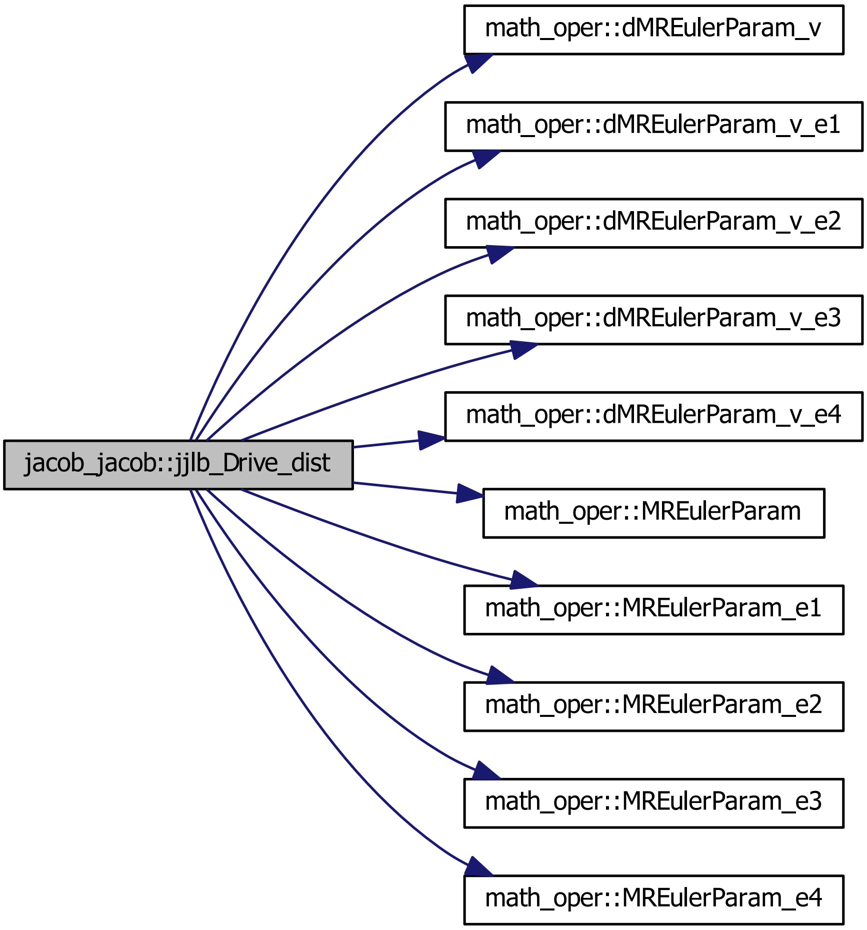
Primitive driving constraints for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |

| | |
|---|---|
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



#### 4.15.2.6 subroutine jacob_jacob::jjlb_Drive_distGB ( INTEGER,intent(in) *ir,* INTEGER,dimension(3),intent(in) *irg2,* INTEGER,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2_loc,* INTEGER,intent(in) *i_MOTOR,* REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq} V}$ of a distance driving jacobians between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:

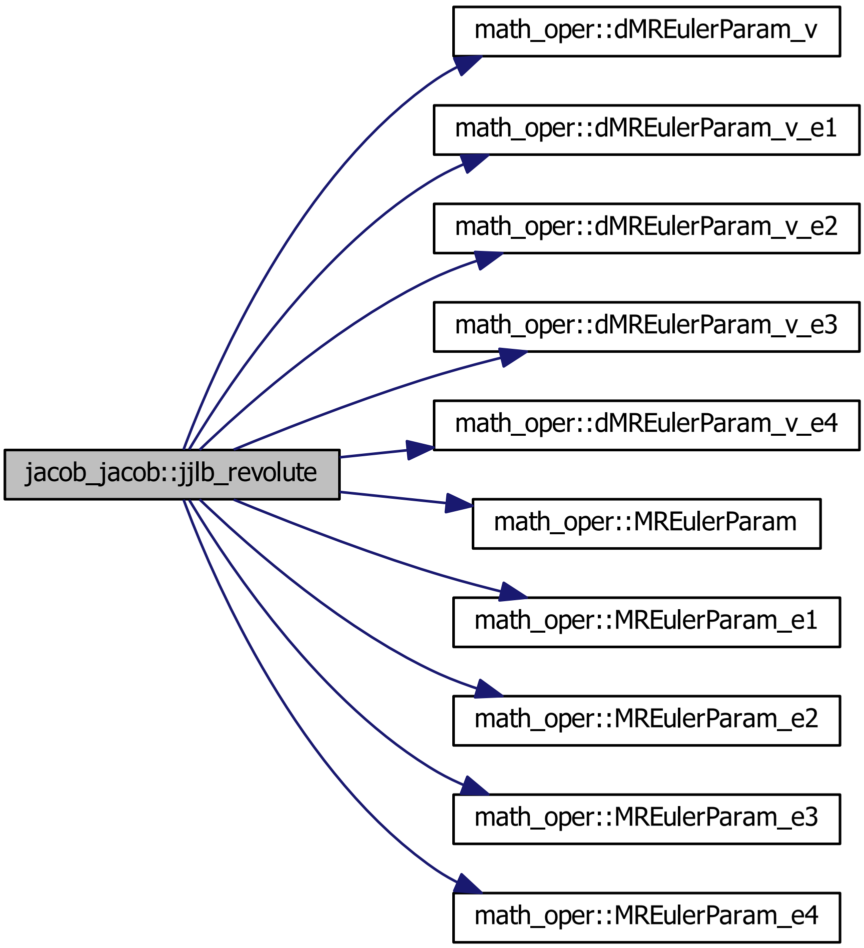

#### 4.15.2.7 subroutine jacob_jacob::jjlb_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2*, REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$ of a revolute joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



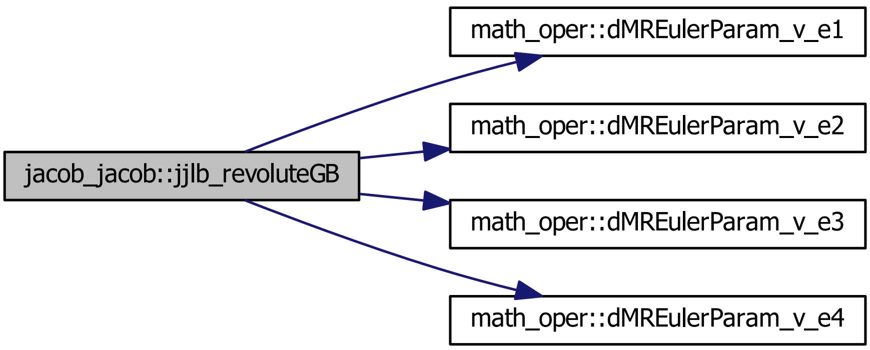
#### 4.15.2.8 subroutine jacob_jacob::jjlb_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2*, REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$ of a revolute joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,v1* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



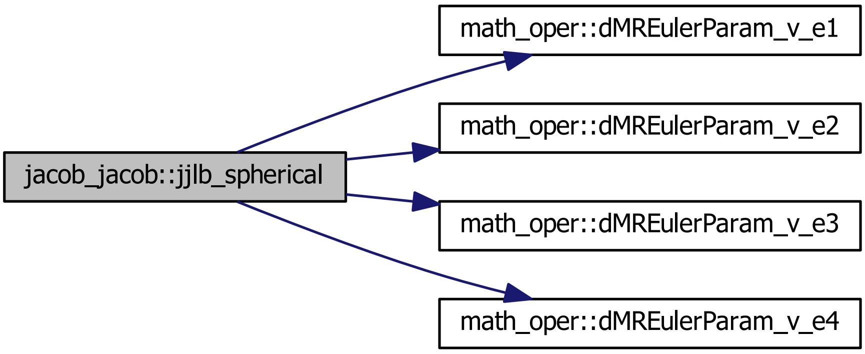

#### 4.15.2.9 subroutine jacob_jacob::jjlb_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$ of a spherical joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



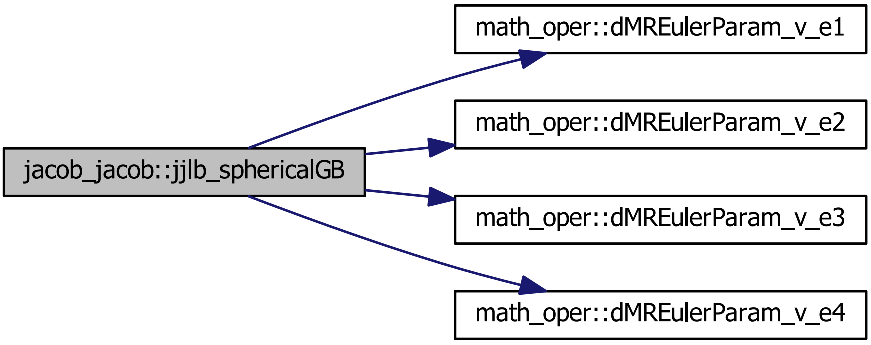

#### 4.15.2.10 subroutine jacob_jacob::jjlb_sphericalGB ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul2,* real(8),dimension(3),intent(in) *pt1,* real(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq}V}$ of a spherical joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



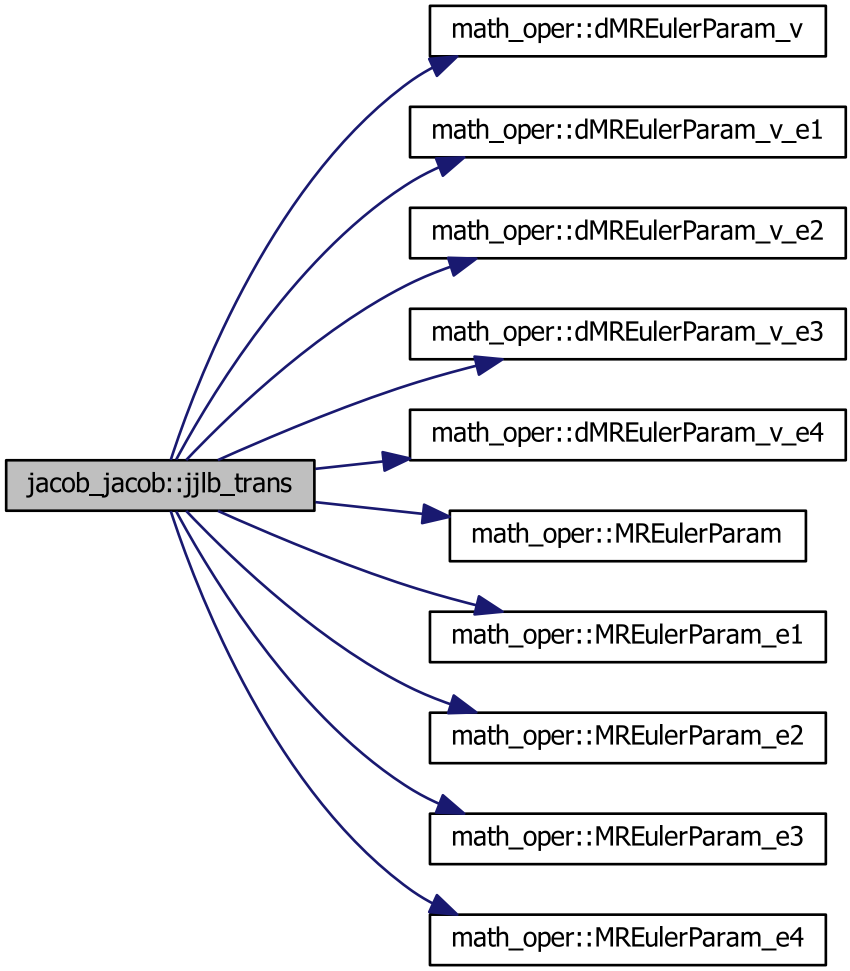

#### 4.15.2.11 subroutine jacob_jacob::jjlb_trans ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg1,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *vec1y,* REAL(8),dimension(3),intent(in) *vec1x,* REAL(8),dimension(3),intent(in) *vec2x,* REAL(8),dimension(3),intent(in) *vec2z,* REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq} V}$ of a translational joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:



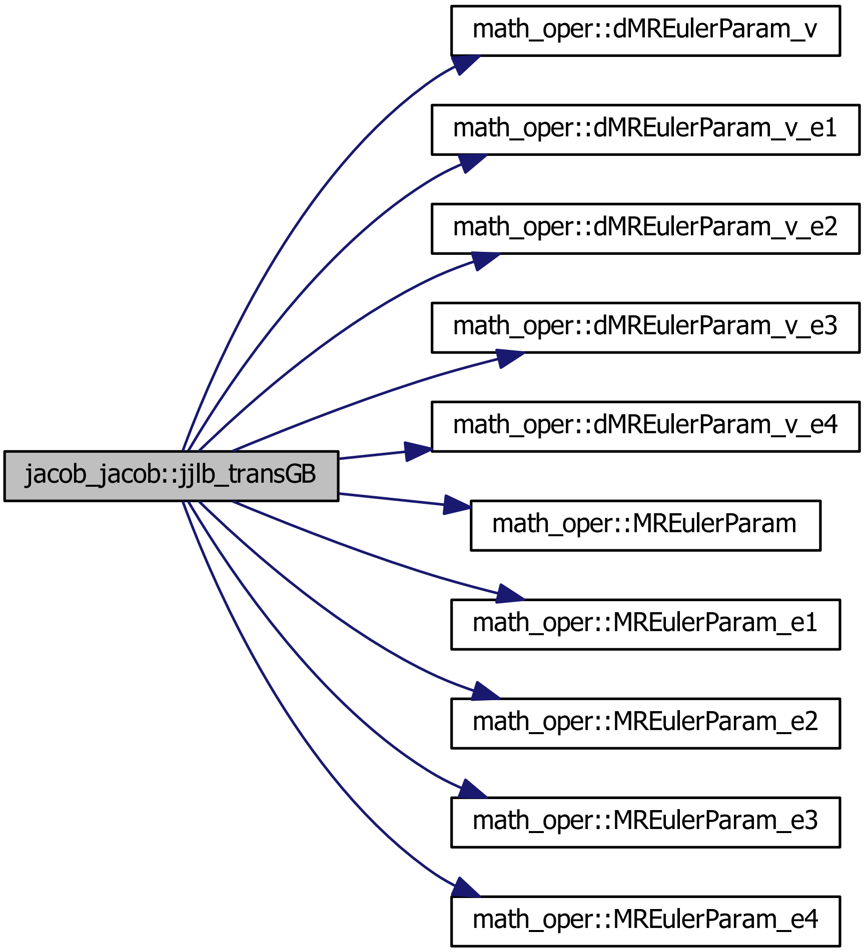
#### 4.15.2.12 subroutine jacob_jacob::jjlb_transGB ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *vec1y,* REAL(8),dimension(3),intent(in) *vec1x,* REAL(8),dimension(3),intent(in) *vec2x,* REAL(8),dimension(3),intent(in) *vec2z,* REAL(8),dimension(dim),intent(in) *lb* )

$\mathbf{\Phi_{qq} V}$ of a translational joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

Here is the call graph for this function:




#### 4.15.2.13 subroutine jacob_jacob::jjlb_UnitEulParam ( integer,intent(in) *ir,* integer,dimension(4),intent(in) *iEul,* REAL(8),dimension(dim),intent(in) *lb* )

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
$\mathbf{\Phi_{qq}V}$ of unitary Euler parameters.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}}$ |

### 4.15.3 Variable Documentation

#### 4.15.3.1 REAL(8),dimension(:,:),allocatable jacob_jacob::Fiqqlb

#### 4.15.3.2 REAL(8),dimension(:,:),allocatable jacob_jacob::PROTECTED

## 4.16 jacobT_jacob Module Reference

Module of $\mathbf{\Phi_{qq}^T V}$, which is the transpose of the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine jacobT_jacob_Setup
- subroutine deallocFiqqtlb
- subroutine resetFiqqtlb
- subroutine jjtlb_UnitEulParam (ir, iEul, lb)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC* $\mathbf{\Phi_{qq}^T V}$ *of unitary Euler parameters.*
- subroutine jjtlb_dot1GB (ir, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of dot-1 jacobian of a body attached on the ground*
- subroutine jjtlb_dot1 (ir, iEul1, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of dot-1 jacobian*
- subroutine jjtlb_sphericalGB (ir, irg2, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a spherical joint of a body attached to the ground*
- subroutine jjtlb_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a spherical joint between two bodies*
- subroutine jjtlb_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a revolute joint of a body attached to the ground*
- subroutine jjtlb_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a revolute joint between two bodies*
- subroutine jjtlb_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a translational joint of a body attached to the ground*
- subroutine jjtlb_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a translational joint between two bodies*
- subroutine jjtlb_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of driving jacobians for a distance between a point in the ground and a point of one body.*

- subroutine jjtlb_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of driving constraints for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Fiqqtlb

## 4.16.1 Detailed Description

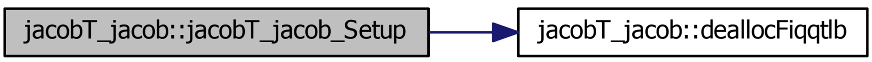
Module of $\mathbf{\Phi_{qq}^T V}$, which is the transpose of the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.

## 4.16.2 Function/Subroutine Documentation

#### 4.16.2.1 subroutine jacobT_jacob::deallocFiqqtlb ( )

#### 4.16.2.2 subroutine jacobT_jacob::jacobT_jacob_Setup ( )

Here is the call graph for this function:



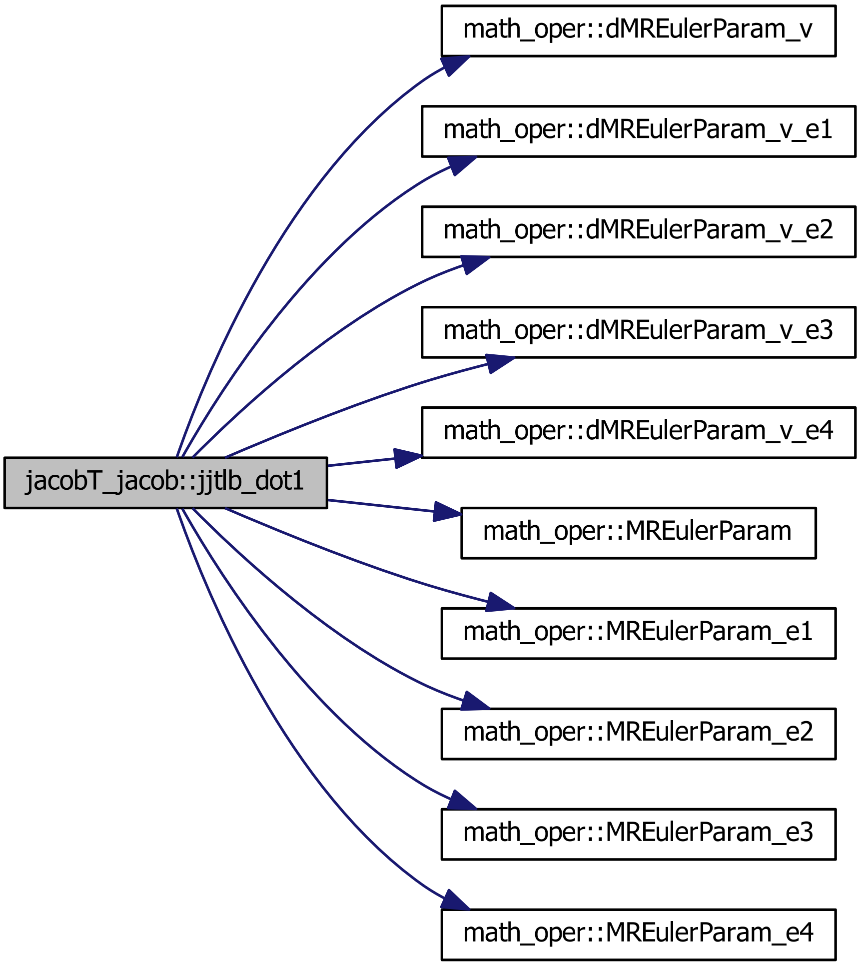

#### 4.16.2.3 subroutine jacobT_jacob::jjtlb_dot1 ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi_{qq}^T V}$ of dot-1 jacobian

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}^T}$ |

Here is the call graph for this function:



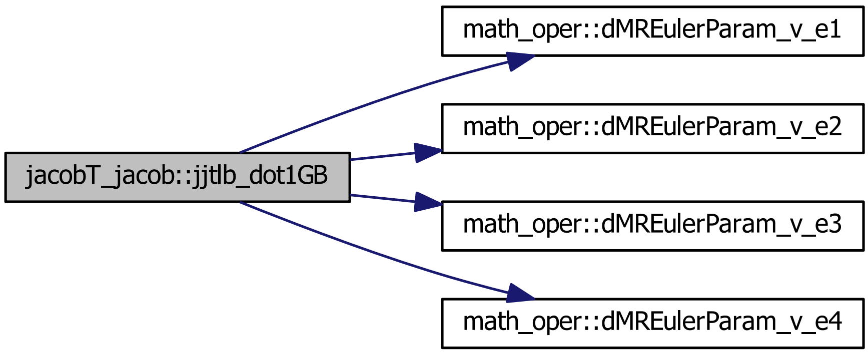

#### 4.16.2.4 subroutine jacobT_jacob::jjtlb_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of dot-1 jacobian of a body attached on the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



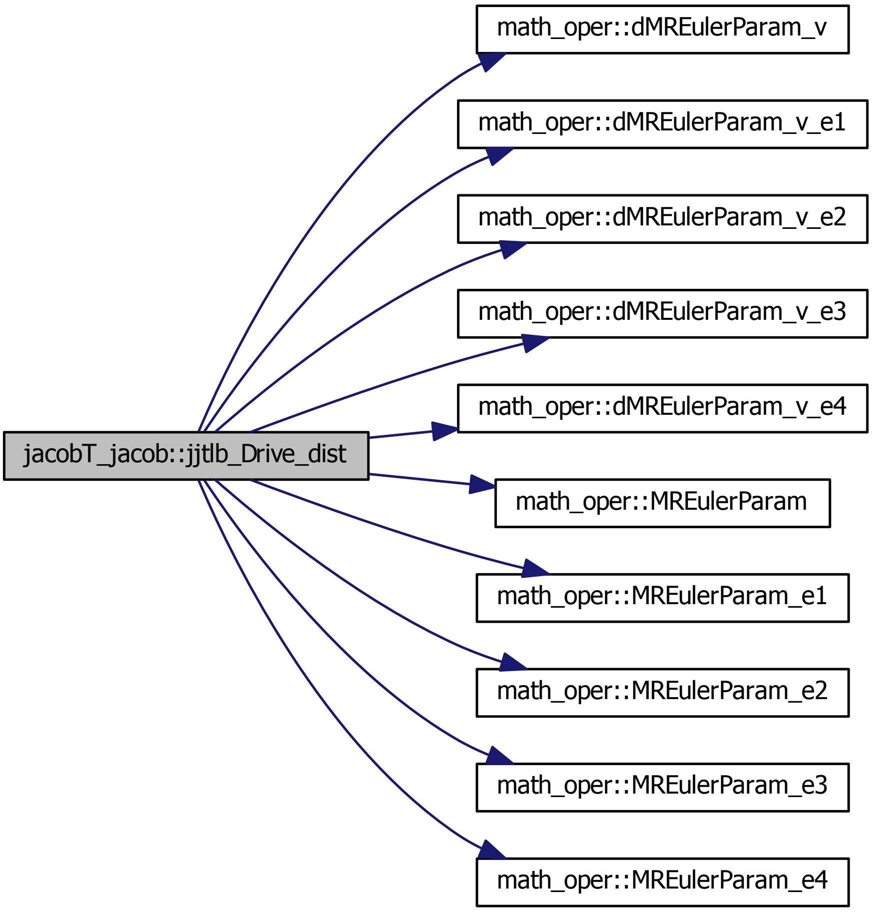

#### 4.16.2.5 subroutine jacobT_jacob::jjtlb_Drive_dist ( INTEGER,intent(in) *ir,* INTEGER,dimension(3),intent(in) *irg1,* INTEGER,dimension(3),intent(in) *irg2,* INTEGER,dimension(4),intent(in) *iEul1,* INTEGER,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1_loc,* REAL(8),dimension(3),intent(in) *pt2_loc,* INTEGER,intent(in) *i_MOTOR,* REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of driving constraints for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1_loc* | point in the first body given in the body reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



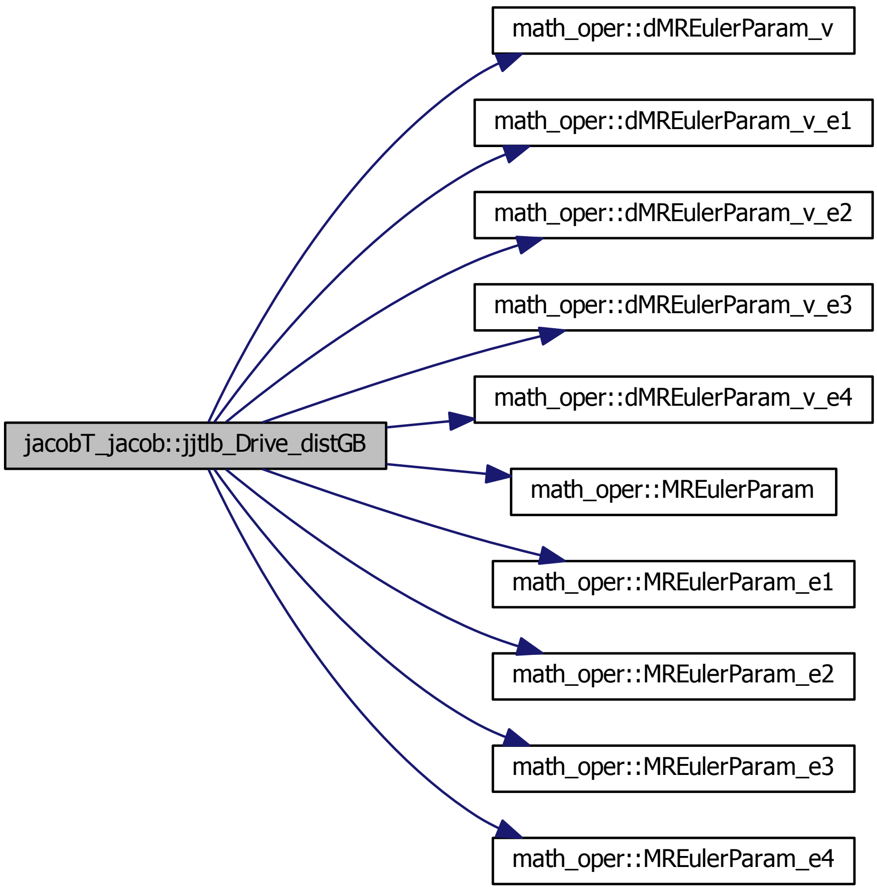

#### 4.16.2.6 subroutine jacobT_jacob::jjtlb_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of driving jacobians for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



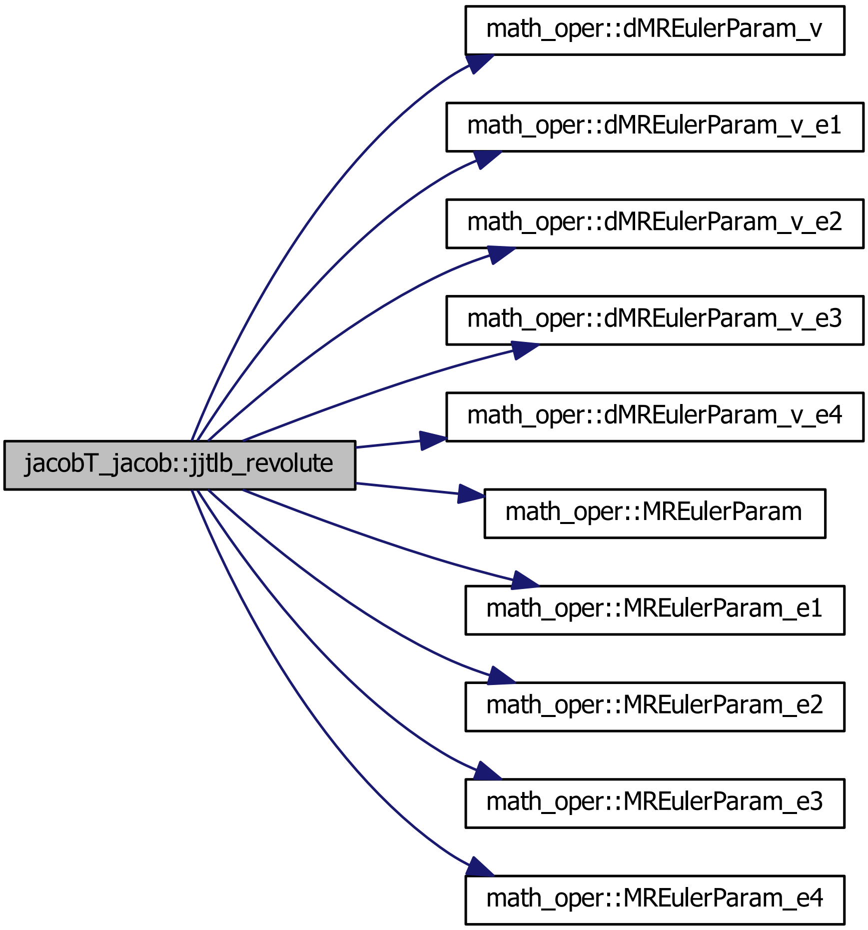
#### 4.16.2.7 subroutine jacobT_jacob::jjtlb_revolute ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of a revolute joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *u1,v1* | perpendicular vectors in the first body |
| *vec2* | vector in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



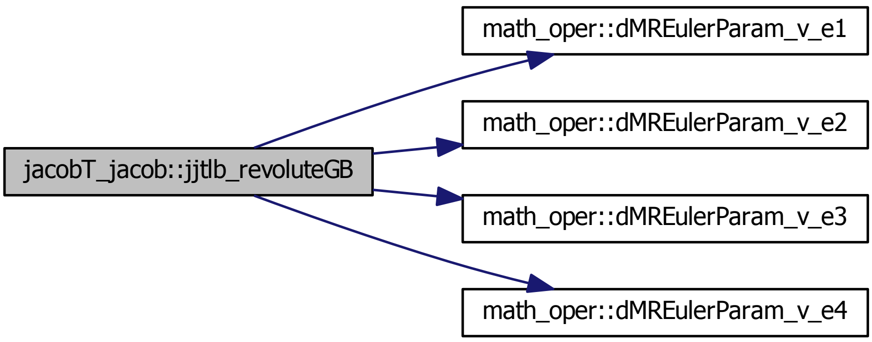

#### 4.16.2.8 subroutine jacobT_jacob::jjtlb_revoluteGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *u1*, REAL(8),dimension(3),intent(in) *v1*, REAL(8),dimension(3),intent(in) *vec2*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of a revolute joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *u1,u2* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



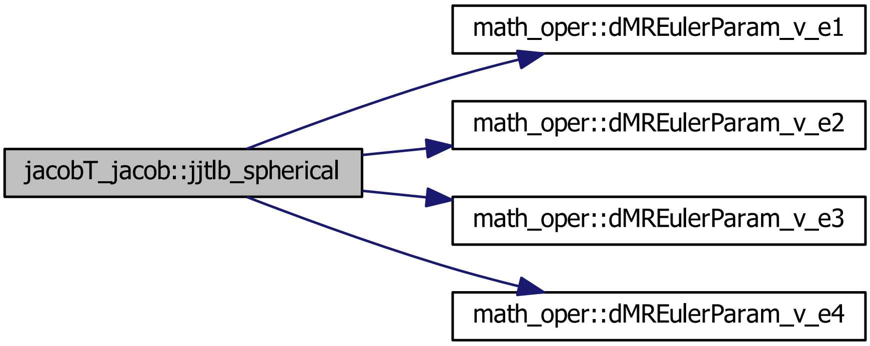

#### 4.16.2.9 subroutine jacobT_jacob::jjtlb_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of a spherical joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1,pt2* | points given in the bodies reference frames |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



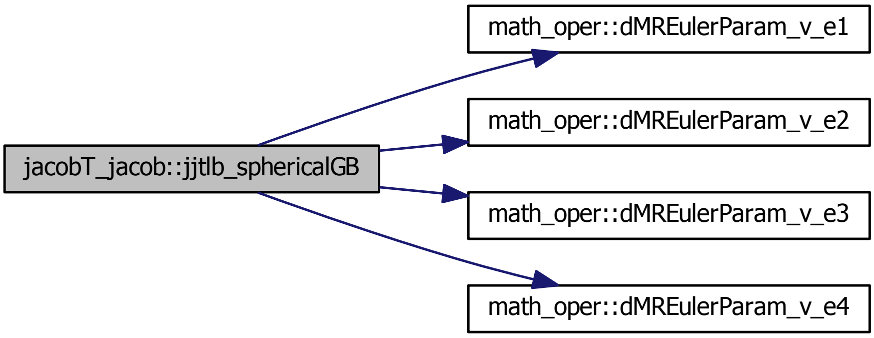

#### 4.16.2.10 subroutine jacobT_jacob::jjtlb_sphericalGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi_{qq}^{T}V}$ of a spherical joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}^{T}}$ |

Here is the call graph for this function:



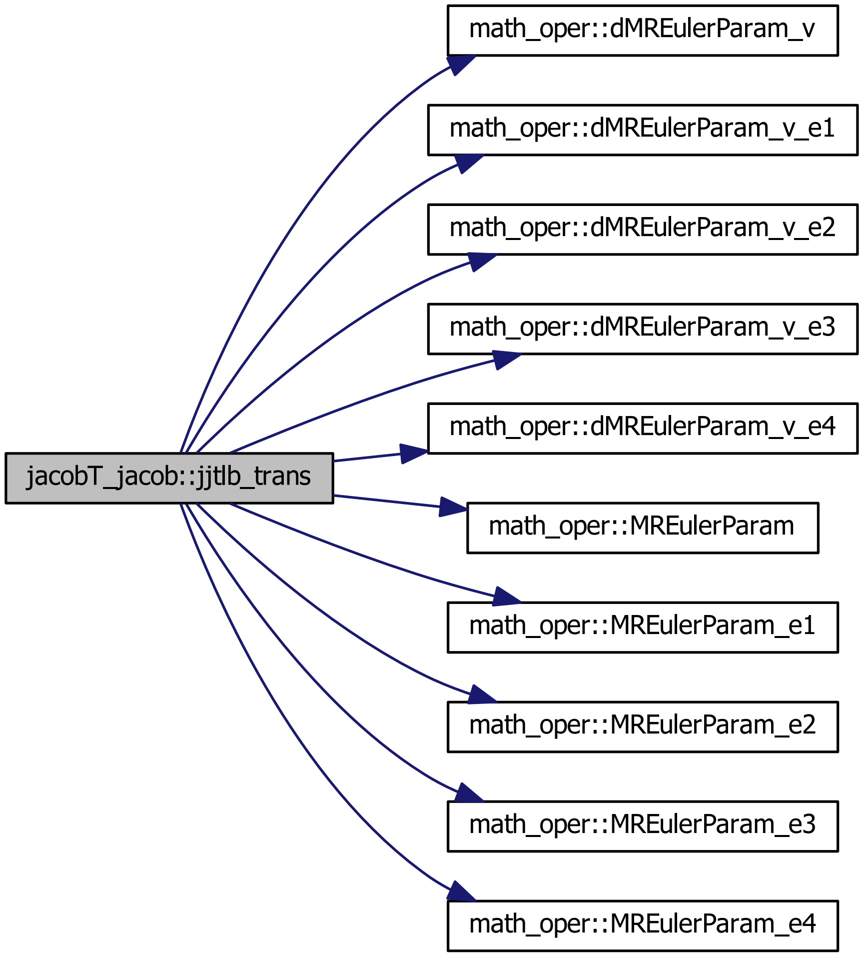

#### 4.16.2.11 subroutine jacobT_jacob::jjtlb_trans ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi_{qq}^{T}V}$ of a translational joint between two bodies

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}^{T}}$ |

Here is the call graph for this function:



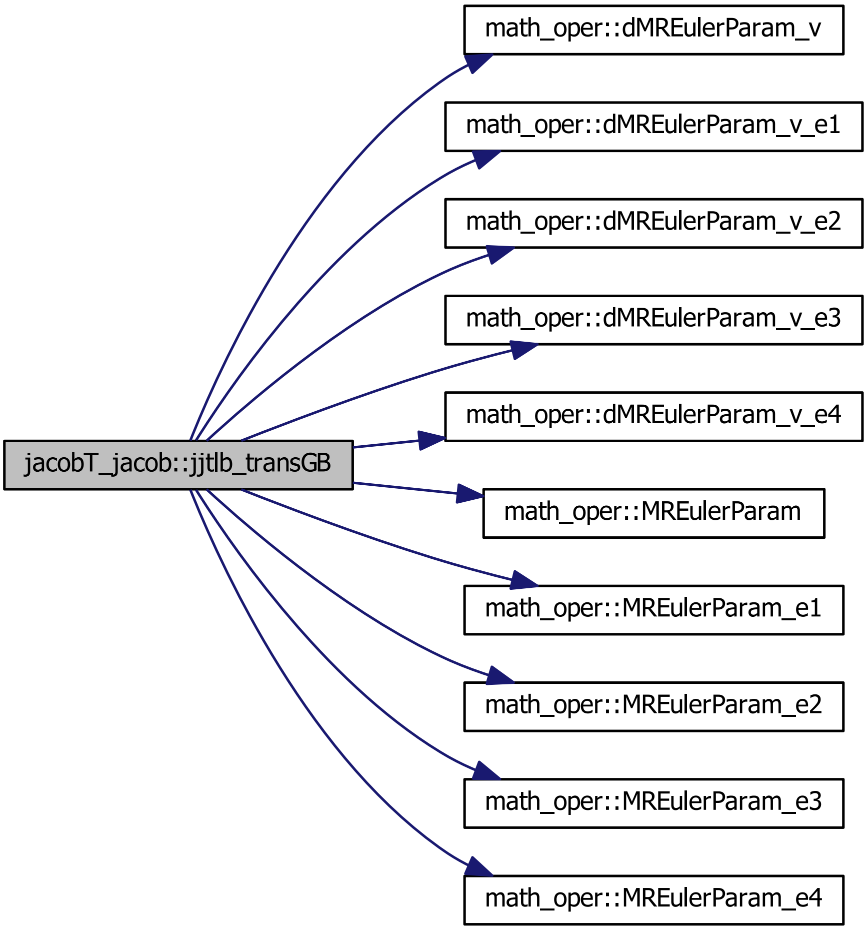

#### 4.16.2.12 subroutine jacobT_jacob::jjtlb_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z*, REAL(8),dimension(nrt),intent(in) *lb* )

$\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ of a translational joint of a body attached to the ground

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}$ |

Here is the call graph for this function:



#### 4.16.2.13 subroutine jacobT_jacob::jjtlb_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul*, REAL(8),dimension(nrt),intent(in) *lb* )

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
$\mathbf{\Phi_{qq}^T V}$ of unitary Euler parameters.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |
| *lb* | the vector $V$ multiplied by $\mathbf{\Phi_{qq}^T}$ |

#### 4.16.2.14 subroutine jacobT_jacob::resetFiqqtlb ( )

### 4.16.3 Variable Documentation

#### 4.16.3.1 REAL(8),dimension(:,:),allocatable jacobT_jacob::Fiqqtlb

#### 4.16.3.2 REAL(8),dimension(:,:),allocatable jacobT_jacob::PROTECTED

## 4.17 mass_massq Module Reference

Module of $\mathbf{M_q V}$, which is the jacobian of the mass matrix multiplied by a vector. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine dMdqV_Setup
- subroutine deallocdMdqV
- subroutine Eval_Mass_Matrix

  *Subroutine to assemble the mass matrix of the whole system.*
- subroutine Mq (body, lb)

  *Subroutine to evaluate $\mathbf{M}_q\mathbf{V}$ of one body.*
- subroutine Eval_Mq (lb)

  *Subroutine to assemble $\mathbf{M}_q\mathbf{V}$ of the whole system.*

### Variables

- REAL(8), dimension(:,:), allocatable PROTECTED
- REAL(8), dimension(:,:), allocatable Mqlb

### 4.17.1 Detailed Description

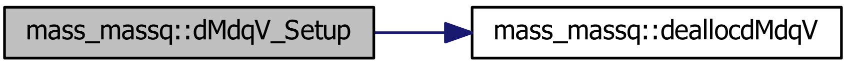
Module of $\mathbf{M_q V}$, which is the jacobian of the mass matrix multiplied by a vector. It's NOT a user module, it's used by the solver.

### 4.17.2 Function/Subroutine Documentation

#### 4.17.2.1 subroutine mass_massq::deallocdMdqV ( )

#### 4.17.2.2 subroutine mass_massq::dMdqV_Setup ( )

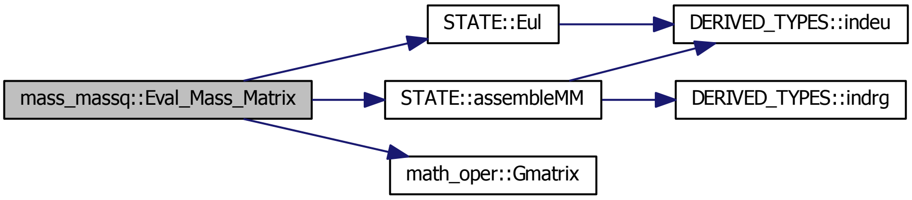
Here is the call graph for this function:




#### 4.17.2.3 subroutine mass_massq::Eval_Mass_Matrix ( )

Subroutine to assemble the mass matrix of the whole system.

Here is the call graph for this function:



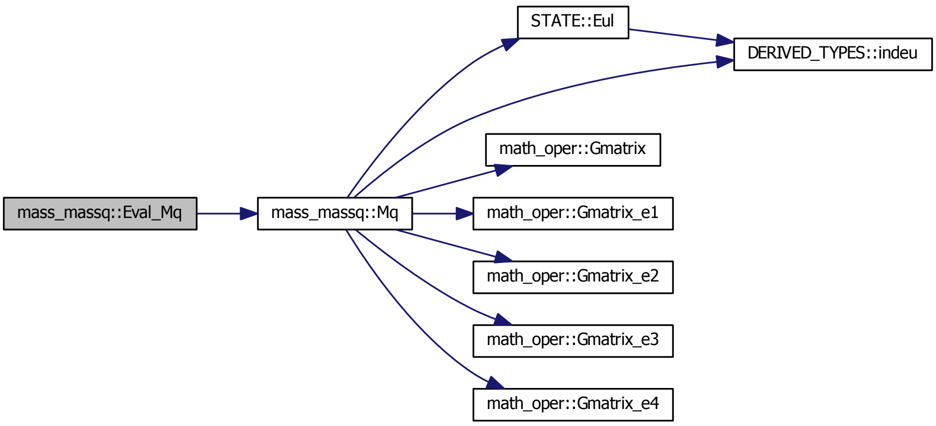

#### 4.17.2.4 subroutine mass_massq::Eval_Mq ( REAL(8),dimension(dim),intent(in) *lb* )

Subroutine to assemble $\mathbf{M}_q\mathbf{V}$ of the whole system.

**Parameters**

| | |
|---|---|
| *lb* | the vector $V$ multiplied by the derivatives of the mass matrix |

Here is the call graph for this function:



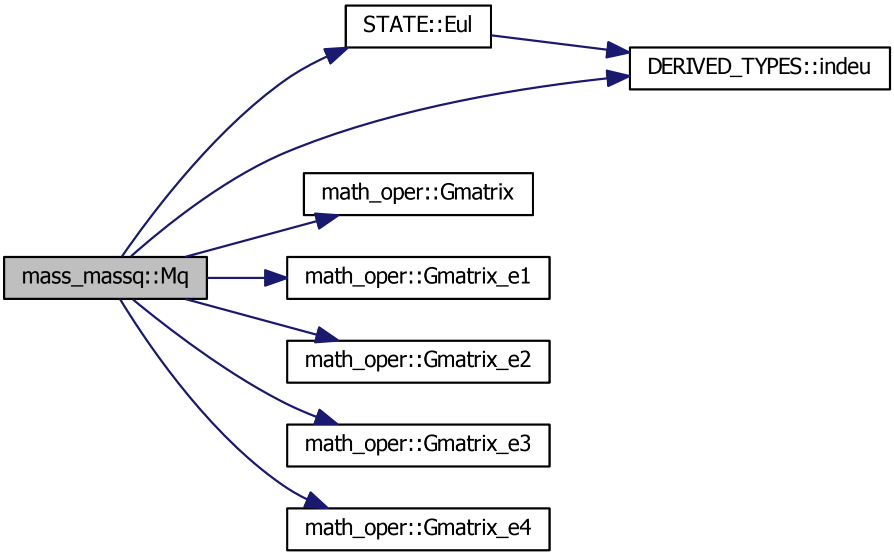

#### 4.17.2.5 subroutine mass_massq::Mq ( integer *body*, REAL(8),dimension(dim),intent(in) *lb* )

Subroutine to evaluate $\mathbf{M}_q\mathbf{V}$ of one body.

**Parameters**

| | |
|---|---|
| *body* | body involved |
| *lb* | the vector $V$ multiplied by the derivatives of the mass matrix |

Here is the call graph for this function:




### 4.17.3 Variable Documentation

#### 4.17.3.1 REAL(8),dimension(:,:),allocatable mass_massq::Mqlb

#### 4.17.3.2 REAL(8),dimension(:,:),allocatable mass_massq::PROTECTED

## 4.18 math_oper Module Reference

Module of non-intrinsic mathematical operations. Contains all the operations necessary for multi body dynamics computations not supported by the Fortran 2003 standard.

### Functions/Subroutines

- REAL(8) norm (r)

  *Calculate the standard* $|\mathbf{r}|$ *of a vector* $\mathbf{r}$.
- REAL(8), dimension(size(r)) normalize (r)

  *Normalize an vector* $\mathbf{normalize}(\mathbf{r}) = \frac{\mathbf{r}}{|\mathbf{r}|}$.
- REAL(8), dimension(3) pvect (u, v)

  *Calculates the cross product of two vectors.*

- subroutine PerpVectors (nfl, u, v)

  *Given a vector, calculates two perpendicular vectors to the given one.*
- real(8) MREulerParam (e)

  *Given the vector of Euler parameters $e$, it returns the rotation transformation matrix $A$.*
- real(8) Gmatrix (e)

  *Given the vector of Euler parameters $e$, it returns the G matrix $G$.*
- real(8) Gmatrix_e1 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_0}.*
- real(8) Gmatrix_e2 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_1}.*
- real(8) Gmatrix_e3 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_2}.*
- real(8) Gmatrix_e4 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_3}.*
- real(8) dMREulerParam_v (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\frac{\partial Av}{\partial e}$.*
- real(8) dMREulerParam_t (p, pd, t)

  *Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}Av}{\mathrm{d}t}$.*
- real(8) d2MREulerParam_t2 (p, pd, ps, t)

  *Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}^2Av}{\mathrm{d}t^2}$.*
- real(8) dMREulerParam_t_e1 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_0$.*
- real(8) dMREulerParam_t_e2 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_1$.*
- real(8) dMREulerParam_t_e3 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_2$.*
- real(8) dMREulerParam_t_e4 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_3$.*
- real(8) dMREulerParam_t_e1d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_0$.*
- real(8) dMREulerParam_t_e2d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_1$.*
- real(8) dMREulerParam_t_e3d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_2$.*
- real(8) dMREulerParam_t_e4d (p, pd, t)

*Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{dAv}{dt} \partial \dot{e}_3$.*

- real(8) dMREulerParam_v_e1 (e, v)

    *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_0$.*

- real(8) dMREulerParam_v_e2 (e, v)

    *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_1$.*

- real(8) dMREulerParam_v_e3 (e, v)

    *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_2$.*

- real(8) dMREulerParam_v_e4 (e, v)

    *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_3$.*

- real(8) MREulerParam_e1 (e)

    *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_0$.*

- real(8) MREulerParam_e2 (e)

    *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_1$.*

- real(8) MREulerParam_e3 (e)

    *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_2$.*

- real(8) MREulerParam_e4 (e)

    *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_3$.*

- real(8), dimension(size(u), size(v)) tensor_product (u, v)

    *Given the two vectors, find the tensor product between these two vectors.*

- real(8) GAV (p, v)

    *Given the euler parameters $p$ and a vector $v$ of one body , find $2 * G^T \tilde{A}v$.*

- real(8) GAV_e1 (p, v)

    *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_0}$.*

- real(8) GAV_e2 (p, v)

    *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_1}$.*

- real(8) GAV_e3 (p, v)

    *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_2}$.*

- real(8) GAV_e4 (p, v)

    *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_3}$.*

## Variables

- REAL(8), dimension(3, 3), parameter EYE3 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0, 0.d0,1.d0,0.d0, 0.d0,0.d0,1.d0/), SHAPE=(/3,3/))
- REAL(8), dimension(3, 3), parameter ZEROS3 = 0.d0
- REAL(8), dimension(4, 4), parameter EYE4 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0,0.d0, 0.d0,1.d0,0.d0,0.d0, 0.d0,0.d0,1.d0,0.d0, 0.d0,0.d0,0.d0,1.d0/), SHAPE=(/4,4/))

### 4.18.1 Detailed Description

Module of non-intrinsic mathematical operations. Contains all the operations necessary for multi body dynamics computations not supported by the Fortran 2003 standard.

### 4.18.2 Function/Subroutine Documentation

#### 4.18.2.1 real(8) math_oper::d2MREulerParam_t2 ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(4),intent(in) *ps*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}^2 Av}{\mathrm{d}t^2}$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the first derivative of Euler parameters. |
| *ps* | vector of the second derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.2 real(8) math_oper::dMREulerParam_t ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}Av}{\mathrm{d}t}$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.3 real(8) math_oper::dMREulerParam_t_e1 ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_0$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.4 real(8) math_oper::dMREulerParam_t_e1d ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_0$.

**Parameters**

| | |
|---:|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.5 real(8) math_oper::dMREulerParam_t_e2 ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_1$.

**Parameters**

| | |
|---:|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.6 real(8) math_oper::dMREulerParam_t_e2d ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_1$.

**Parameters**

| | |
|---:|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.7 real(8) math_oper::dMREulerParam_t_e3 ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_2$.

**Parameters**

| | |
|---:|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.8 real(8) math_oper::dMREulerParam_t_e3d ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_2$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.9 real(8) math_oper::dMREulerParam_t_e4 ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_3$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.10 real(8) math_oper::dMREulerParam_t_e4d ( real(8),dimension(4),intent(in) *p*, real(8),dimension(4),intent(in) *pd*, real(8),dimension(3),intent(in) *t* )

Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_3$.

**Parameters**

| | |
|---|---|
| *p* | vector of Euler parameters. |
| *pd* | vector of the derivative of Euler parameters. |
| *t* | vector given in the body reference frame. |

#### 4.18.2.11 real(8) math_oper::dMREulerParam_v ( real(8),dimension(4),intent(in) *e*, real(8),dimension(3),intent(in) *v* )

Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\frac{\partial Av}{\partial e}$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |
| *v* | vector given in the body reference frame. |

#### 4.18.2.12 real(8) math_oper::dMREulerParam_v_e1 ( real(8),dimension(4),intent(in) *e*, real(8),dimension(3),intent(in) *v* )

Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_0$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |
| *v* | vector given in the body reference frame. |

#### 4.18.2.13 real(8) math_oper::dMREulerParam_v_e2 ( real(8),dimension(4),intent(in) *e*, real(8),dimension(3),intent(in) *v* )

Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_1$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |
| *v* | vector given in the body reference frame. |

#### 4.18.2.14 real(8) math_oper::dMREulerParam_v_e3 ( real(8),dimension(4),intent(in) *e*, real(8),dimension(3),intent(in) *v* )

Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_2$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |
| *v* | vector given in the body reference frame. |

#### 4.18.2.15 real(8) math_oper::dMREulerParam_v_e4 ( real(8),dimension(4),intent(in) *e*, real(8),dimension(3),intent(in) *v* )

Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_3$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |
| *v* | vector given in the body reference frame. |

#### 4.18.2.16 real(8) math_oper::GAV ( real(8),dimension(4) *p*, real(8),dimension(3) *v* )

Given the euler parameters $p$ and a vector $v$ of one body , find $2 * G^T \tilde{A} v$.

**Parameters**

| | |
|---|---|
| *p* | euler parameters |
| *v* | the vector |

#### 4.18.2.17 real(8) math_oper::GAV_e1 ( real(8),dimension(4) *p*, real(8),dimension(3) *v* )

Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{\mathrm{d}2*G^T\tilde{A}v}{\mathrm{d}e_0}$.

**Parameters**

| | |
|---|---|
| *p* | euler parameters |
| *v* | the vector |

#### 4.18.2.18 real(8) math_oper::GAV_e2 ( real(8),dimension(4) *p*, real(8),dimension(3) *v* )

Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{\mathrm{d}2*G^T\tilde{A}v}{\mathrm{d}e_1}$.

**Parameters**

| | |
|---|---|
| *p* | euler parameters |
| *v* | the vector |

#### 4.18.2.19 real(8) math_oper::GAV_e3 ( real(8),dimension(4) *p*, real(8),dimension(3) *v* )

Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{\mathrm{d}2*G^T\tilde{A}v}{\mathrm{d}e_2}$.

**Parameters**

| | |
|---|---|
| *p* | euler parameters |
| *v* | the vector |

#### 4.18.2.20 real(8) math_oper::GAV_e4 ( real(8),dimension(4) *p*, real(8),dimension(3) *v* )

Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{\mathrm{d}2*G^T\tilde{A}v}{\mathrm{d}e_3}$.

**Parameters**

| | |
|---|---|
| *p* | euler parameters |
| *v* | the vector |

#### 4.18.2.21 real(8) math_oper::Gmatrix ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the G matrix $G$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.22 real(8) math_oper::Gmatrix_e1 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns ={ A}{ e_0}.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.23 real(8) math_oper::Gmatrix_e2 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns ={ A}{ e_1}.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.24 real(8) math_oper::Gmatrix_e3 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns ={ A}{ e_2}.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.25 real(8) math_oper::Gmatrix_e4 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns ={ A}{ e_3}.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.26 real(8) math_oper::MREulerParam ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the rotation transformation matrix $A$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.27 real(8) math_oper::MREulerParam_e1 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_0$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.28 real(8) math_oper::MREulerParam_e2 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_1$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.29 real(8) math_oper::MREulerParam_e3 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_2$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.30 real(8) math_oper::MREulerParam_e4 ( real(8),dimension(4),intent(in) *e* )

Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_3$.

**Parameters**

| | |
|---|---|
| *e* | vector of Euler parameters. |

#### 4.18.2.31 REAL(8) math_oper::norm ( REAL(8),dimension(:) *r* )

Calculate the standard $|\mathbf{r}|$ of a vector $\mathbf{r}$.

**Parameters**

| | |
|---|---|
| *r* | vector. |

#### 4.18.2.32 REAL(8),dimension(size(r)) math_oper::normalize ( REAL(8),dimension(:) *r* )

Normalize an vector $\mathbf{normalize(r)} = \frac{\mathbf{r}}{|\mathbf{r}|}$.

**Parameters**

| | |
|---|---|
| *r* | vector |

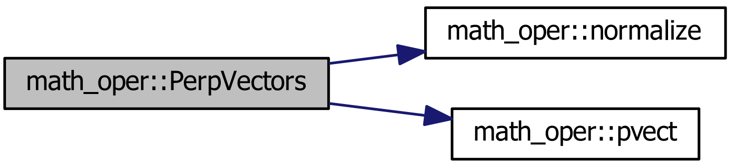
#### 4.18.2.33 subroutine math_oper::PerpVectors ( REAL(8),dimension(3),intent(in) *nfl*, REAL(8),dimension(3),intent(out) *u*, REAL(8),dimension(3),intent(out) *v* )

Given a vector, calculates two perpendicular vectors to the given one.

**Parameters**

| | |
|---|---|
| *nfl* | given vector. |

Here is the call graph for this function:




#### 4.18.2.34 REAL(8),dimension(3) math_oper::pvect ( REAL(8),dimension(:),intent(in) *u*, REAL(8),dimension(:),intent(in) *v* )

Calculates the cross product of two vectors.

**Parameters**

| | |
|---|---|
| *u,v* | vectors involved. |

#### 4.18.2.35 real(8),dimension(size(u),size(v)) math_oper::tensor_product ( real(8),dimension(:) *u*, real(8),dimension(:) *v* )

Given the two vectors, find the tensor product between these two vectors.

**Parameters**

| | |
|---|---|
| *u* | the first vector |
| *v* | the second vector |

### 4.18.3 Variable Documentation

#### 4.18.3.1 REAL(8),dimension(3,3),parameter math_oper::EYE3 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0, 0.d0,1.d0,0.d0, 0.d0,0.d0,1.d0/), SHAPE=(/3,3/))

#### 4.18.3.2 REAL(8),dimension(4,4),parameter math_oper::EYE4 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0,0.d0, 0.d0,1.d0,0.d0,0.d0, 0.d0,0.d0,1.d0,0.d0, 0.d0,0.d0,0.d0,1.d0/), SHAPE=(/4,4/))

#### 4.18.3.3 REAL(8),dimension(3,3),parameter math_oper::ZEROS3 = 0.d0

## 4.19 matlab_caller Module Reference

Managment of sessions of MATLAB engine: This is a part of the matlab_caller module of MBSLIM.

### Functions/Subroutines

- subroutine MATLAB_OPENSES

  *OPENS MATLAB SESSION.*
- subroutine MATLAB_CLOSESES

  *CLOSES MATLAB SESSION.*
- subroutine MATLAB_CHECKSES

  *CHECKS IF THERE IS A MATLAB SESSION (for internal use of the module only)*
- subroutine MATLAB_EVALSTRING (STRING)

  *It evaluates Matlab expression.*
- subroutine MATLAB_PUTREALVECTOR (b, NOMBRE)

  *It passes a real vector b.*
- subroutine MATLAB_GETREALVECTOR (b, NOMBRE)

  *It gets/reads the vector 'nombre' from matlab and it places it in b!>*
- subroutine MATLAB_GETREAL (c, NOMBRE)

  *It reads the scalar NOMBRE from Matlab and it places it on variable c.*
- subroutine MATLAB_PUTINTEGER (i, NOMBRE)

  *PASSES AN INTEGER i.*
- subroutine MATLAB_PLOT (t_graph, y_graph, figur, linecolor, linewidth)

  *PLOTS 2 VECTORS OF REAL DATA Y.VS.X.*

### 4.19.1 Detailed Description

Managment of sessions of MATLAB engine: This is a part of the matlab_caller module of MBSLIM.

### 4.19.2 Function/Subroutine Documentation

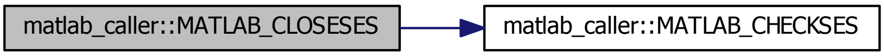
#### 4.19.2.1 subroutine matlab_caller::MATLAB_CHECKSES ( )

CHECKS IF THERE IS A MATLAB SESSION (for internal use of the module only)

#### 4.19.2.2 subroutine matlab_caller::MATLAB_CLOSESES ( )

CLOSES MATLAB SESSION.

Here is the call graph for this function:




#### 4.19.2.3 subroutine matlab_caller::MATLAB_EVALSTRING ( CHARACTER(LEN=*) *STRING* )

It evaluates Matlab expression.

**Parameters**

| | |
|---|---|
| *string* | text chain to evaluate |

Here is the call graph for this function:

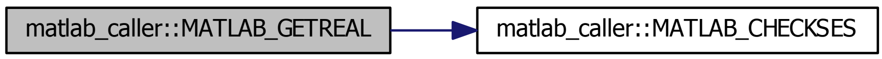
#### 4.19.2.4 subroutine matlab_caller::MATLAB_GETREAL ( REAL(8),intent(out) *c*, CHARACTER(LEN=∗) *NOMBRE* )

It reads the scalar NOMBRE from Matlab and it places it on variable c.

**Parameters**

| | |
|---|---|
| *c* | Fortran scalar |
| *NOMBRE* | matlab name of the variable |

Here is the call graph for this function:



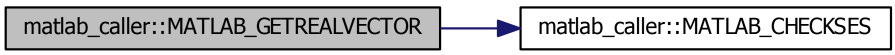

#### 4.19.2.5 subroutine matlab_caller::MATLAB_GETREALVECTOR ( REAL(8),dimension(:),intent(out) *b*, CHARACTER(LEN=∗) *NOMBRE* )

It gets/reads the vector 'nombre' from matlab and it places it in b!>

**Parameters**

| | |
|---|---|
| *b* | vector |
| *NOMBRE* | matlab name |

Here is the call graph for this function:




#### 4.19.2.6 subroutine matlab_caller::MATLAB_OPENSES ( )

OPENS MATLAB SESSION.

Here is the call graph for this function:

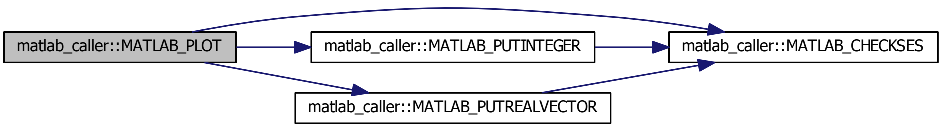

#### 4.19.2.7 subroutine matlab_caller::MATLAB_PLOT ( REAL(8),dimension(:),intent(in) *t_graph*, REAL(8),dimension(:),intent(in) *y_graph*, *figur*, CHARACTER(LEN=1),optional *linecolor*, REAL(8),optional *linewidth* )

PLOTS 2 VECTORS OF REAL DATA Y.VS.X.

**Parameters**

| | |
|---|---|
| *t_graph* | abscisa vector |
| *y_graph* | ordinate vector |
| *figur* | number of the matlab figure |
| *linecolor* | line color (see matlab codes) |
| *linewidth* | line width |

Here is the call graph for this function:




#### 4.19.2.8 subroutine matlab_caller::MATLAB_PUTINTEGER ( ,intent(in) *i*, CHARACTER(LEN=∗) *NOMBRE* )

PASSES AN INTEGER i.

**Parameters**

| | |
|---|---|
| *i* | scalar |
| *NOMBRE* | matlab name of the integer |

Here is the call graph for this function:

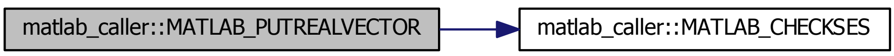

#### 4.19.2.9 subroutine matlab_caller::MATLAB_PUTREALVECTOR ( REAL(8),dimension(:),intent(in) *b*, CHARACTER(LEN=∗) *NOMBRE* )

It passes a real vector b.

**Parameters**

| | |
|---|---|
| *b* | vector |
| *NOMBRE* | matlab name of the vector |

Here is the call graph for this function:




## 4.20 primitive_forces Module Reference

Primitive forces module.

### Functions/Subroutines

- subroutine TSDA (r1, r2, r1p, r2p, s0, k, c, F1, F2)

  *Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA_q (r1, r2, r1p, r2p, s0, k, c, df1dr1, df1dr2, df2dr1, df2dr2)

  *Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine TSDA_qp (r1, r2, c, df1dr1p, df1dr2p, df2dr1p, df2dr2p)

*Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.*

### 4.20.1 Detailed Description

Primitive forces module. This module:

1)Contains computational routines for primitive forces.

### 4.20.2 Function/Subroutine Documentation

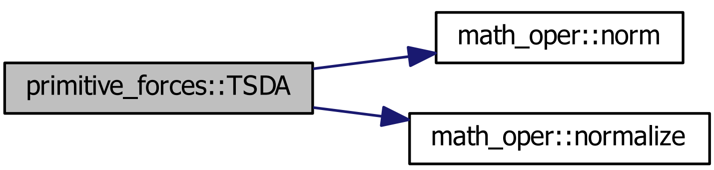
#### 4.20.2.1 subroutine primitive_forces::TSDA ( REAL(8),dimension(3),intent(in) *r1*, REAL(8),dimension(3),intent(in) *r2*, REAL(8),dimension(3),intent(in) *r1p*, REAL(8),dimension(3),intent(in) *r2p*, REAL(8),intent(in) *s0*, REAL(8),intent(in) *k*, REAL(8),intent(in) *c*, REAL(8),dimension(3),intent(out) *F1*, REAL(8),dimension(3),intent(out) *F2* )

Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *F1* | return the primitive force acting on the first body |
| *F2* | return the primitive force acting on the second body |

Here is the call graph for this function:



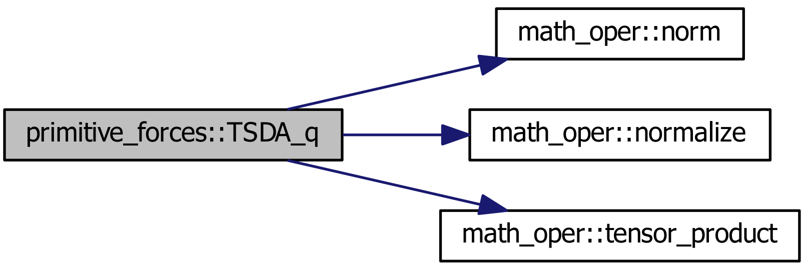
#### 4.20.2.2 subroutine primitive_forces::TSDA_q ( real(8),dimension(3),intent(in) *r1*, real(8),dimension(3),intent(in) *r2*, real(8),dimension(3),intent(in) *r1p*, real(8),dimension(3),intent(in) *r2p*, real(8),intent(in) *s0*, real(8),intent(in) *k*, real(8),intent(in) *c*, real(8),dimension(3,3),intent(out) *df1dr1*, real(8),dimension(3,3),intent(out) *df1dr2*, real(8),dimension(3,3),intent(out) *df2dr1*, real(8),dimension(3,3),intent(out) *df2dr2* )

Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *df1dr1,df1dr2* | return the primitive stiffness acting on the first body |
| *df2dr1,df2dr2* | return the primitive stiffness acting on the second body |

Here is the call graph for this function:



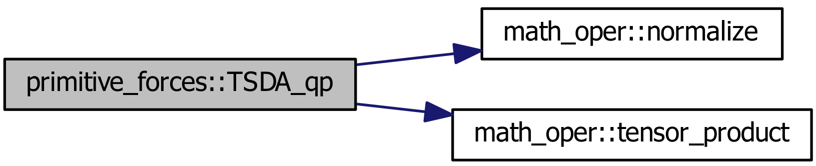
#### 4.20.2.3 subroutine primitive_forces::TSDA_qp ( real(8),dimension(3),intent(in) *r1*, real(8),dimension(3),intent(in) *r2*, real(8),intent(in) *c*, real(8),dimension(3,3),intent(out) *df1dr1p*, real(8),dimension(3,3),intent(out) *df1dr2p*, real(8),dimension(3,3),intent(out) *df2dr1p*, real(8),dimension(3,3),intent(out) *df2dr2p* )

Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.

**Parameters**

| | |
|---|---|
| *t* | time. |
| *body1* | the first body involved. |
| *body2* | the second body involved. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *s0* | the unstreched length of the spring |
| *k* | the stiffness of the spring |
| *c* | the damping ratio of the damper |
| *df1dr1p,df1dr2* | return the primitive damping acting on the first body |
| *df2dr1p,df2dr2* | return the primitive damping acting on the second body |

Here is the call graph for this function:




## 4.21 restric Module Reference

Module of primitive constraints. It's NOT a user module, it's used by the solver.

### Functions/Subroutines

- subroutine restric_Setup
- subroutine deallocfi

- subroutine r_UnitEulParam (ir, iEul)

  *Primitive constraint of unitary Euler parameters assume* $\mathbf{p}$ *is the Euler parameter.the constraint equation is :* $\mathbf{p}^T\mathbf{p}-1$.
- subroutine r_dot1GB (ir, iEul2, u, v)

  *Primitive dot-1 constraint of a body attached on the ground.*
- subroutine r_dot1 (ir, iEul1, iEul2, u, v)

  *Primitive dot-1 constraint assume* $\mathbf{p_1}$ *and* $\mathbf{p_2}$ *are the Euler parameter of body 1 and body 2 and* $\mathbf{u}$ *and* $\mathbf{v}$ *are two vectors attached on body 1 and body 2 in the body reference frame, the constraint equation is :* $A(\mathbf{p_1})\mathbf{u}^T A(\mathbf{p_2})\mathbf{v}$.
- subroutine r_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *Primitive constraints of a spherical joint of a body attached to the ground The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{\prime P}-\mathbf{s}_1^P$.
- subroutine r_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *Primitive constraints of a spherical joint between two bodies The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{\prime P}-\mathbf{r}_1-\mathbf{A}_1\mathbf{s}_1^{\prime P}$.
- subroutine r_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *Primitive constraints of a revolute joint of a body attached to the ground The first three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{\prime P}-\mathbf{s}_1^P$ *The fouth constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{h}_2$ *The fifth constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{h}_2$.
- subroutine r_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *Primitive constraints of a revolute joint between two bodies The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{\prime P}-\mathbf{r}_1-\mathbf{A}_1\mathbf{s}_1^{\prime P}$ *The fouth constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The fifth constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$.
- subroutine r_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive constraints of a translational joint of a body attached to the ground The first constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{h}_2$ *The second constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{h}_2$ *The third constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{d}_{12}$ *The forth constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{d}_{12}$ *The fifth constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{f}_2$.
- subroutine r_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive constraints of a translational joint between two bodies. The first constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The second constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The third constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ *The forth constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ *The fifth constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{f}_2$.
- subroutine r_Drive_rgEul (ir, ind, i_MOTOR)

  *Primitive driving constraints for a generalized coordinate of the system.*
- subroutine r_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *Primitive driving constraints for a distance between a point in the ground and a point of one body.*
- subroutine r_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *Primitive driving constraints for a distance between two points of two bodies.*

## Variables

- REAL(8), dimension(:), allocatable PROTECTED
- REAL(8), dimension(:), allocatable fi

### 4.21.1 Detailed Description

Module of primitive constraints. It's NOT a user module, it's used by the solver.

### 4.21.2 Function/Subroutine Documentation

#### 4.21.2.1 subroutine restric::deallocfi ( )

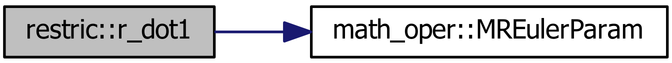
#### 4.21.2.2 subroutine restric::r_dot1 ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

Primitive dot-1 constraint assume $\mathbf{p_1}$ and $\mathbf{p_2}$ are the Euler parameter of body 1 and body 2 and $\mathbf{u}$ and $\mathbf{v}$ are two vectors attached on body 1 and body 2 in the body reference frame, the constraint equation is : $A(\mathbf{p_1})\mathbf{u}^T A(\mathbf{p_2})\mathbf{v}$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *u* | vector in the first body given in the body reference frame |
| *v* | vector in the second body given in the body reference frame |

Here is the call graph for this function:



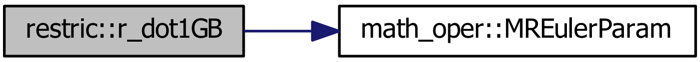

#### 4.21.2.3 subroutine restric::r_dot1GB ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *u*, real(8),dimension(3),intent(in) *v* )

Primitive dot-1 constraint of a body attached on the ground.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul2* | indexes of the Euler parameters of the body. |
| *u* | vector attached on the ground |
| *v* | vector in the second body given in the body reference frame assume are the Euler parameter of the body the constraint equation is : $\mathbf{u}^T A(\mathbf{p})\mathbf{v}$ |

Here is the call graph for this function:



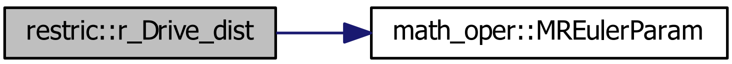

#### 4.21.2.4 subroutine restric::r_Drive_dist ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg1*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul1*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1_loc*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

Primitive driving constraints for a distance between two points of two bodies.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg1,irg2* | index of the center of mass of the body. |
| *iEul1,iEul2* | index of the Euler parameters of the body. |
| *pt1_-loc,pt2_loc* | points in the bodies given in the bodies reference frames. |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:



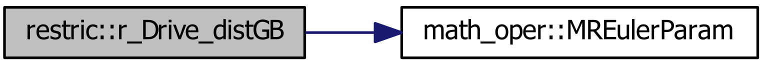
#### 4.21.2.5 subroutine restric::r_Drive_distGB ( INTEGER,intent(in) *ir*, INTEGER,dimension(3),intent(in) *irg2*, INTEGER,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2_loc*, INTEGER,intent(in) *i_MOTOR* )

Primitive driving constraints for a distance between a point in the ground and a point of one body.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint. |
| *irg2* | index of the center of mass of the body. |
| *iEul2* | index of the Euler parameters of the body. |
| *pt1* | point in the ground given in the global reference frame |
| *pt2_loc* | point in the second body given in the body reference frame |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

Here is the call graph for this function:




#### 4.21.2.6 subroutine restric::r_Drive_rgEul ( INTEGER,intent(in) *ir*, INTEGER,intent(in) *ind*, INTEGER,intent(in) *i_MOTOR* )

Primitive driving constraints for a generalized coordinate of the system.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *ind* | index of the driven generalized coordinate. |
| *i_MOTOR* | index in the vector of motors (STATE::pos) to drive the constraint. |

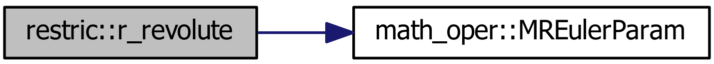
#### 4.21.2.7 subroutine restric::r_revolute ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg1,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *u1,* REAL(8),dimension(3),intent(in) *v1,* REAL(8),dimension(3),intent(in) *vec2* )

Primitive constraints of a revolute joint between two bodies The three constraint equations are : $\mathbf{r}_2 + \mathbf{A}_2\mathbf{s}_2^{'P} - \mathbf{r}_1 - \mathbf{A}_1\mathbf{s}_1^{'P}$ The fouth constraint equation is: $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ The fifth constraint equation is: $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |
| *u1,v1* | perpendicular vectors in the first body given in the body reference frame |
| *vec2* | vector in the second body given in the body reference frame |

Here is the call graph for this function:




#### 4.21.2.8 subroutine restric::r_revoluteGB ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *u1,* REAL(8),dimension(3),intent(in) *v1,* REAL(8),dimension(3),intent(in) *vec2* )

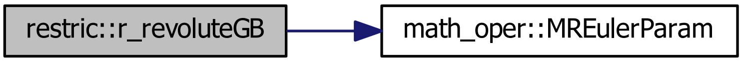
Primitive constraints of a revolute joint of a body attached to the ground The first three constraint equations are : $\mathbf{r}_2 + \mathbf{A}_2\mathbf{s}_2^{'P} - \mathbf{s}_1^P$ The fouth constraint equation is: $\mathbf{f}_1^T\mathbf{A}_2\mathbf{h}_2$ The fifth constraint equation is: $\mathbf{g}_1^T\mathbf{A}_2\mathbf{h}_2$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

| | |
|---|---|
| *u1,v1* | perpendicular vectors in the ground |
| *vec2* | vector in the body given in the body reference frame |

Here is the call graph for this function:



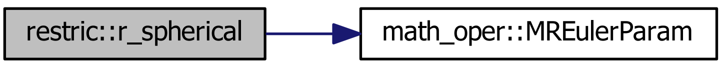

#### 4.21.2.9 subroutine restric::r_spherical ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg1*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul1*, integer,dimension(4),intent(in) *iEul2*, real(8),dimension(3),intent(in) *pt1*, real(8),dimension(3),intent(in) *pt2* )

Primitive constraints of a spherical joint between two bodies The three constraint equations are : $\mathbf{r}_2 + \mathbf{A}_2\mathbf{s}_2^{\prime P} - \mathbf{r}_1 - \mathbf{A}_1\mathbf{s}_1^{\prime P}$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point in the first body given in the body reference frame |
| *pt2* | point in the second body given in the body reference frame |

Here is the call graph for this function:



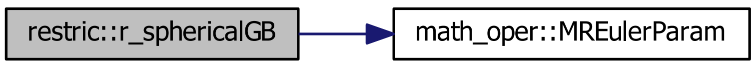
#### 4.21.2.10 subroutine restric::r_sphericalGB ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul2,* real(8),dimension(3),intent(in) *pt1,* real(8),dimension(3),intent(in) *pt2* )

Primitive constraints of a spherical joint of a body attached to the ground The three constraint equations are : $\mathbf{r}_2 + \mathbf{A}_2\mathbf{s}_2^{\prime P} - \mathbf{s}_1^P$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameters of the body |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |

Here is the call graph for this function:



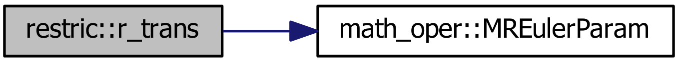

#### 4.21.2.11 subroutine restric::r_trans ( integer,intent(in) *ir,* integer,dimension(3),intent(in) *irg1,* integer,dimension(3),intent(in) *irg2,* integer,dimension(4),intent(in) *iEul1,* integer,dimension(4),intent(in) *iEul2,* REAL(8),dimension(3),intent(in) *pt1,* REAL(8),dimension(3),intent(in) *pt2,* REAL(8),dimension(3),intent(in) *vec1y,* REAL(8),dimension(3),intent(in) *vec1x,* REAL(8),dimension(3),intent(in) *vec2x,* REAL(8),dimension(3),intent(in) *vec2z* )

Primitive constraints of a translational joint between two bodies. The first constraint equation is: $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ The second constraint equation is: $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$ The third constraint equation is: $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ The forth constraint equation is: $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ The fifth constraint equation is: $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{f}_2$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg1,irg2* | indexes of the centers of mass of the bodies. |
| *iEul1,iEul2* | indexes of the Euler parameters of the bodies. |
| *pt1* | point given in the first body given in the body reference frame |
| *pt2* | point given in the second body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the first body given in the body reference frame |
| *vec2x,vec2z* | perpendicular vectors in the second body given in the body reference frame |

Here is the call graph for this function:



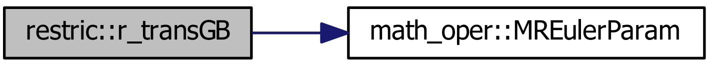

#### 4.21.2.12 subroutine restric::r_transGB ( integer,intent(in) *ir*, integer,dimension(3),intent(in) *irg2*, integer,dimension(4),intent(in) *iEul2*, REAL(8),dimension(3),intent(in) *pt1*, REAL(8),dimension(3),intent(in) *pt2*, REAL(8),dimension(3),intent(in) *vec1y*, REAL(8),dimension(3),intent(in) *vec1x*, REAL(8),dimension(3),intent(in) *vec2x*, REAL(8),dimension(3),intent(in) *vec2z* )

Primitive constraints of a translational joint of a body attached to the ground The first constraint equation is: $\mathbf{f}_1^T \mathbf{A}_2 \mathbf{h}_2$ The second constraint equation is: $\mathbf{g}_1^T \mathbf{A}_2 \mathbf{h}_2$ The third constraint equation is: $\mathbf{f}_1^T \mathbf{A}_2 \mathbf{d}_{12}$ The forth constraint equation is: $\mathbf{g}_1^T \mathbf{A}_2 \mathbf{d}_{12}$ The fifth constraint equation is: $\mathbf{f}_1^T \mathbf{A}_2 \mathbf{f}_2$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *irg2* | index of the center of mass of the body |
| *iEul2* | index of the Euler parameter of the body. |
| *pt1* | point in the ground |
| *pt2* | point in the body given in the body reference frame |
| *vec1y,vec1x* | perpendicular vectors in the ground |
| *vec2x,vec2z* | perpendicular vectors in the body given in the body reference frame |

Here is the call graph for this function:



#### 4.21.2.13 subroutine restric::r_UnitEulParam ( integer,intent(in) *ir*, integer,dimension(4),intent(in) *iEul* )

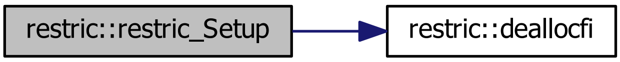
Primitive constraint of unitary Euler parameters assume $\mathbf{p}$ is the Euler parameter.the constraint equation is : $\mathbf{p}^T\mathbf{p}-1$.

**Parameters**

| | |
|---|---|
| *ir* | index of the constraint |
| *iEul* | indexes of the Euler parameters |

#### 4.21.2.14 subroutine restric::restric_Setup ( )

Here is the call graph for this function:




### 4.21.3 Variable Documentation

#### 4.21.3.1 REAL(8),dimension(:),allocatable restric::fi

#### 4.21.3.2 REAL(8),dimension(:),allocatable restric::PROTECTED

## 4.22 SOLIDS Module Reference

Solids module that adds and manages the bodies of the system.

### Functions/Subroutines

- subroutine, public ADDbody (body, rg0, eu0)

  *Adds a body to the model.*
- subroutine, public ADDmassgeom (body, mass, rg_loc, v_lg)

  *Subroutine to add mass, center of masses and inertia tensor with respect to the center of mass to a body.*
- REAL(8), public MatTrans (body)

  *Function to calculate the transformation matrix of a body* $\mathbf{A}^* = \begin{bmatrix} \mathbf{R} & \mathbf{p}_0 \\ \mathbf{0} & 1 \end{bmatrix}$.

- REAL(8), dimension(3), public r (body, pt_loc)

  *Function to evaluate the position of a point belonging to a body.*
- REAL(8), dimension(3), public rp (body, pt_loc)

  *Function to evaluate the velocity of a point belonging to a body.*

### Variables

- TYPE(SOLID), dimension(:), allocatable, public PROTECTED
- TYPE(SOLID), dimension(:), allocatable, public SOLIDlist
- INTEGER, parameter, public ground = 0
- INTEGER, public nSOLID = 0

## 4.22.1 Detailed Description

Solids module that adds and manages the bodies of the system. This module:

1)Adds bodies to the model.

2)Adds mass, center of mass and intertia tensor to the existing bodies.

3)Manages the creation of the vector of variables of the model

## 4.22.2 Function/Subroutine Documentation

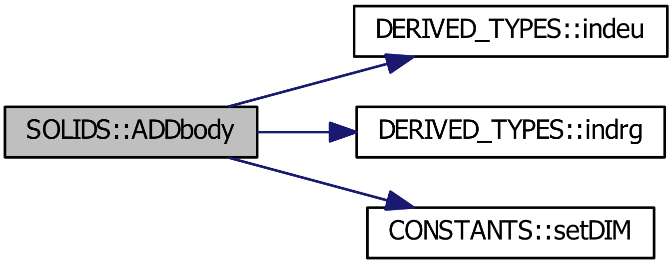
#### 4.22.2.1 subroutine,public SOLIDS::ADDbody ( INTEGER,intent(out) *body*, REAL(8),dimension(3) *rg0*, REAL(8),dimension(4) *eu0* )

Adds a body to the model.

**Parameters**

| | |
|---|---|
| *body* | output variable that stores the id of the body. The user should keep it unchanged for future requests to the library. |
| *rg0* | initial guess for the coordinates of the center of mass of the body |
| *eu0* | initial guess for the Euler parameters of the body |

Here is the call graph for this function:




#### 4.22.2.2 subroutine,public SOLIDS::ADDmassgeom ( INTEGER,intent(in) *body*, REAL(8),intent(in) *mass*, REAL(8),dimension(3),intent(in) *rg_loc*, REAL(8),dimension(6),intent(in) *v_lg* )

Subroutine to add mass, center of masses and inertia tensor with respect to the center of mass to a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *mass* | value of the mass. |
| *rg_loc* | center of mass of the body in the local reference frame of the body. |
| *v_lg* | components of the inertia tensor in the local reference frame of the body, with respect to its center of mass, given in the compact form: $\mathbf{I}_g = \left[ \begin{array}{cccccc} I_x & I_y & I_z & P_{xy} & P_{xz} & P_{yz} \end{array} \right]$ |

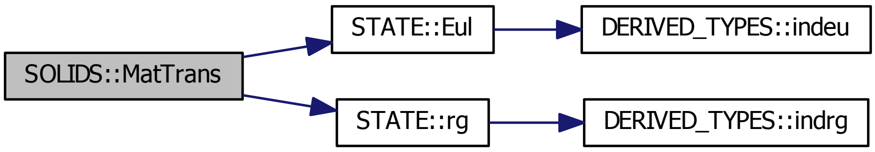
#### 4.22.2.3 REAL(8),public SOLIDS::MatTrans ( INTEGER,intent(in) *body* )

Function to calculate the transformation matrix of a body $\mathbf{A}^* = \left[ \begin{array}{cc} \mathbf{R} & \mathbf{p}_0 \\ \mathbf{0} & 1 \end{array} \right]$.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



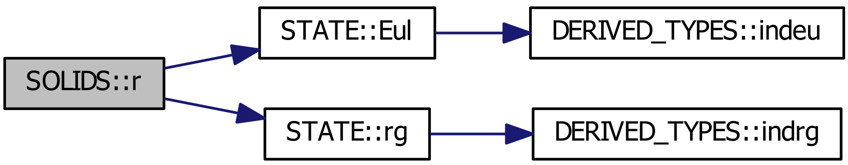

#### 4.22.2.4 REAL(8),dimension(3),public SOLIDS::r ( INTEGER,intent(in) *body*, REAL(8),dimension(3),intent(in) *pt_loc* )

Function to evaluate the position of a point belonging to a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *pt_loc* | local coordinates of the point. |

Here is the call graph for this function:



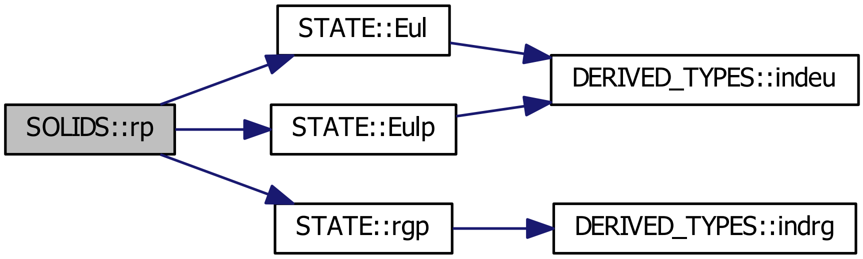

#### 4.22.2.5 REAL(8),dimension(3),public SOLIDS::rp ( INTEGER,intent(in) *body*, REAL(8),dimension(3),intent(in) *pt_loc* )

Function to evaluate the velocity of a point belonging to a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *pt_loc* | local coordinates of the point. |

Here is the call graph for this function:




### 4.22.3 Variable Documentation

#### 4.22.3.1 INTEGER,parameter,public SOLIDS::ground = 0

#### 4.22.3.2 INTEGER,public SOLIDS::nSOLID = 0

#### 4.22.3.3 INTEGER,dimension(:),allocatable,public SOLIDS::PROTECTED

#### 4.22.3.4 TYPE(SOLID),dimension(:),allocatable,public SOLIDS::SOLIDlist

## 4.23 STATE Module Reference

Module of solver state variables, subroutines and functions. It creates, manages and updates the state variables of the model.

### Functions/Subroutines

- subroutine STATE_Setup
- subroutine setDOF (dof_in, zp_in, zs_in)

  *Define the degrees of freedom of the system and the speed of such degree of freedom.*
- REAL(8), dimension(3) rg (body)

  *Function to ask the coordinates of the CG of a body.*
- REAL(8), dimension(3) rgp (body)

*Function to ask the velocity of the CG of a body.*

- REAL(8), dimension(3) rgs (body)

  *Function to ask the acceleration of the CG of a body.*

- REAL(8) rgx (body)

  *Function to ask the x-coordinate of the CG of a body.*

- REAL(8) rgy (body)

  *Function to ask the y-coordinate of the CG of a body.*

- REAL(8) rgz (body)

  *Function to ask the z-coordinate of the CG of a body.*

- REAL(8), dimension(4) Eul (body)

  *Function to ask the Euler parameters of a body.*

- REAL(8), dimension(4) Eulp (body)

  *Function to ask the first derivatives of the Euler parameters of a body.*

- REAL(8), dimension(4) Euls (body)

  *Function to ask the second derivatives of the Euler parameters of a body.*

- subroutine assembleMM (i, m, JJ)

  *Subrotine to assemble elemental mass matrices to the global one. It's NOT a user function, it's intended to be called by the solver.*

### Variables

- REAL(8), dimension(:), allocatable q
- REAL(8), dimension(:), allocatable qp
- REAL(8), dimension(:), allocatable qs
- REAL(8), dimension(:), allocatable qp_g
- REAL(8), dimension(:), allocatable qs_g
- REAL(8), dimension(:), allocatable zp
- REAL(8), dimension(:), allocatable zs
- REAL(8), dimension(:,:), allocatable MM
- REAL(8), dimension(:), allocatable lambda
- REAL(8), dimension(:), allocatable pos
- REAL(8), dimension(:), allocatable vel
- REAL(8), dimension(:), allocatable ace
- INTEGER, dimension(:), allocatable gdl

## 4.23.1 Detailed Description

Module of solver state variables, subroutines and functions. It creates, manages and updates the state variables of the model.

### 4.23.2 Function/Subroutine Documentation

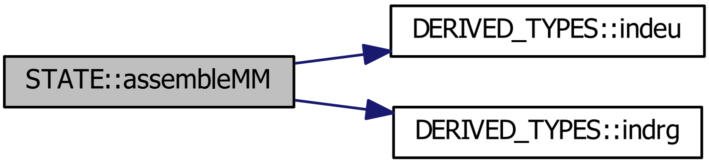
#### 4.23.2.1 subroutine STATE::assembleMM ( INTEGER,intent(in) *i*, REAL(8),intent(in) *m*, REAL(8),dimension(4,4),intent(in) *JJ* )

Subrotine to assemble elemental mass matrices to the global one. It's NOT a user function, it's intended to be called by the solver.

**Parameters**

| | |
|---|---|
| *body* | body involved. |
| *m* | mass. |
| *JJ* | inertia. |

Here is the call graph for this function:



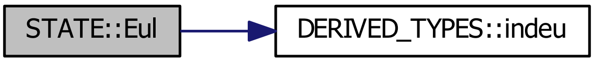

#### 4.23.2.2 REAL(8),dimension(4) STATE::Eul ( INTEGER,intent(in) *body* )

Function to ask the Euler parameters of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



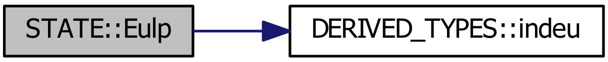

#### 4.23.2.3 REAL(8),dimension(4) STATE::Eulp ( INTEGER,intent(in) *body* )

Function to ask the first derivatives of the Euler parameters of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



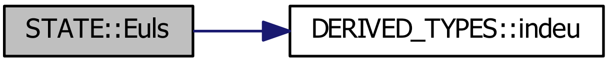

#### 4.23.2.4 REAL(8),dimension(4) STATE::Euls ( INTEGER,intent(in) *body* )

Function to ask the second derivatives of the Euler parameters of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



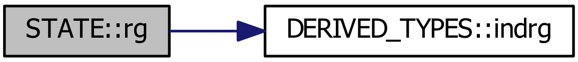

#### 4.23.2.5 REAL(8),dimension(3) STATE::rg ( INTEGER,intent(in) *body* )

Function to ask the coordinates of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



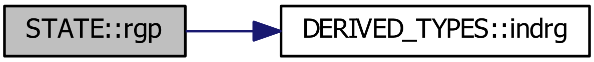

#### 4.23.2.6 REAL(8),dimension(3) STATE::rgp ( INTEGER,intent(in) *body* )

Function to ask the velocity of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



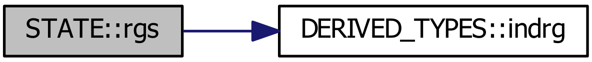

#### 4.23.2.7 REAL(8),dimension(3) STATE::rgs ( INTEGER,intent(in) *body* )

Function to ask the acceleration of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



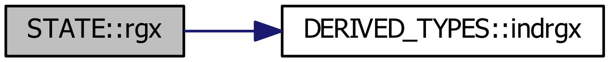

#### 4.23.2.8 REAL(8) STATE::rgx ( INTEGER,intent(in) *body* )

Function to ask the x-coordinate of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



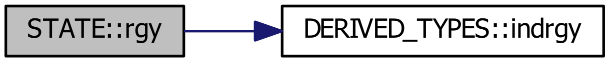

#### 4.23.2.9 REAL(8) STATE::rgy ( INTEGER,intent(in) *body* )

Function to ask the y-coordinate of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



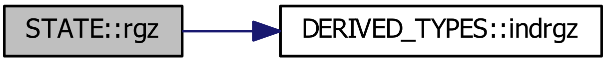

#### 4.23.2.10 REAL(8) STATE::rgz ( INTEGER,intent(in) *body* )

Function to ask the z-coordinate of the CG of a body.

**Parameters**

| | |
|---|---|
| *body* | body involved. |

Here is the call graph for this function:



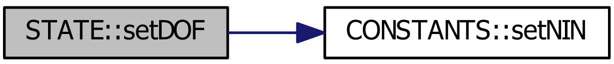

#### 4.23.2.11 subroutine STATE::setDOF ( INTEGER,dimension(:),intent(in) *dof_in*, REAL(8),dimension(:),intent(in) *zp_in*, REAL(8),dimension(:),intent(in),optional *zs_in* )

Define the degrees of freedom of the system and the speed of such degree of freedom.

**Parameters**

| | |
|---|---|
| *dof_in* | positions of the vector of generalized coordinates that you want to define as degree of freedom. |
| *zp_in* | velocities of the vector of generalized coordinates that you want to define as degree of freedom. |
| *zs_in* | accelerations of the vector of generalized coordinates that you want to define as degree of freedom. |

Here is the call graph for this function:




#### 4.23.2.12 subroutine STATE::STATE_Setup ( )

### 4.23.3 Variable Documentation

#### 4.23.3.1 REAL(8),dimension(:),allocatable STATE::ace

#### 4.23.3.2 INTEGER,dimension(:),allocatable STATE::gdl

#### 4.23.3.3 REAL(8),dimension(:),allocatable STATE::lambda

#### 4.23.3.4 REAL(8),dimension(:,:),allocatable STATE::MM

#### 4.23.3.5 REAL(8),dimension(:),allocatable STATE::pos

#### 4.23.3.6 REAL(8),dimension(:),allocatable STATE::q

#### 4.23.3.7 REAL(8),dimension(:),allocatable STATE::qp

#### 4.23.3.8 REAL(8),dimension(:),allocatable STATE::qp_g

#### 4.23.3.9 REAL(8),dimension(:),allocatable STATE::qs

#### 4.23.3.10 REAL(8),dimension(:),allocatable STATE::qs_g

#### 4.23.3.11 REAL(8),dimension(:),allocatable STATE::vel

#### 4.23.3.12 REAL(8),dimension(:),allocatable STATE::zp

#### 4.23.3.13 REAL(8),dimension(:),allocatable STATE::zs

# Chapter 5

# Data Type Documentation

## 5.1 DERIVED_TYPES::callback_AdjInit Interface Reference

### Public Member Functions

- subroutine callback_AdjInit (N, NP, NADJ, T, Y, Lambda, Mu)

### 5.1.1 Constructor & Destructor Documentation

#### 5.1.1.1 subroutine DERIVED_TYPES::callback_AdjInit::callback_AdjInit ( integer *N*, integer *NP*, integer *NADJ*, DOUBLE PRECISION *T*, DOUBLE PRECISION,dimension(n) *Y*, DOUBLE PRECISION,dimension(n,nadj) *Lambda*, DOUBLE PRECISION,dimension(np,nadj) *Mu* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.2 DERIVED_TYPES::callback_damping Interface Reference

### Public Member Functions

- subroutine callback_damping (Cuser, t)

### 5.2.1 Constructor & Destructor Documentation

#### 5.2.1.1 subroutine DERIVED_TYPES::callback_damping::callback_damping ( REAL(8),dimension(∗),intent(out) *Cuser*, REAL(8),intent(in) *t* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.3 DERIVED_TYPES::callback_dgdp Interface Reference

### Public Member Functions

- subroutine callback_dgdp (NADJ, N, NP, T, Y, RP)

### 5.3.1 Constructor & Destructor Documentation

#### 5.3.1.1 subroutine DERIVED_TYPES::callback_dgdp::callback_dgdp ( integer *NADJ*, integer *N*, integer *NP*, REAL(8) *T*, REAL(8),dimension(n) *Y*, REAL(8),dimension(np,nadj) *RP* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.4 DERIVED_TYPES::callback_dgdy Interface Reference

### Public Member Functions

- subroutine callback_dgdy (NADJ, N, NR, T, Y, RY)

### 5.4.1 Constructor & Destructor Documentation

#### 5.4.1.1 subroutine DERIVED_TYPES::callback_dgdy::callback_dgdy ( integer *NADJ*, integer *N*, integer *NR*, REAL(8) *T*, REAL(8),dimension(n) *Y*, REAL(8),dimension(nr,nadj) *RY* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.5 DERIVED_TYPES::callback_forces Interface Reference

### Public Member Functions

- subroutine callback_forces (Quser, t)

### 5.5.1 Constructor & Destructor Documentation

#### 5.5.1.1 subroutine DERIVED_TYPES::callback_forces::callback_forces ( REAL(8),dimension(∗),intent(out) *Quser*, REAL(8),intent(in) *t* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.6 DERIVED_TYPES::callback_gfun Interface Reference

### Public Member Functions

- subroutine callback_gfun (N, NADJ, T, Y, R)

### 5.6.1 Constructor & Destructor Documentation

#### 5.6.1.1 subroutine DERIVED_TYPES::callback_gfun::callback_gfun ( integer *N*, integer *NADJ*, DOUBLE PRECISION *T*, DOUBLE PRECISION,dimension(n) *Y*, DOUBLE PRECISION,dimension(nadj) *R* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.7 DERIVED_TYPES::callback_PMbarPrhoVdot Interface Reference

### Public Member Functions

- subroutine callback_PMbarPrhoVdot (MRVuser, t)

### 5.7.1 Constructor & Destructor Documentation

#### 5.7.1.1 subroutine DERIVED_TYPES::callback_PMbarPrhoVdot::callback_PMbarPrhoVdot ( REAL(8),dimension(∗),intent(out) *MRVuser*, REAL(8),intent(in) *t* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.8 DERIVED_TYPES::callback_PQbarPrho Interface Reference

### Public Member Functions

- subroutine callback_PQbarPrho (QPuser, t)

### 5.8.1 Constructor & Destructor Documentation

#### 5.8.1.1 subroutine DERIVED_TYPES::callback_PQbarPrho::callback_PQbarPrho ( REAL(8),dimension(∗),intent(out) *QPuser,* REAL(8),intent(in) *t* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.9 DERIVED_TYPES::callback_stiffness Interface Reference

### Public Member Functions

- subroutine callback_stiffness (Kuser, t)

### 5.9.1 Constructor & Destructor Documentation

#### 5.9.1.1 subroutine DERIVED_TYPES::callback_stiffness::callback_stiffness ( REAL(8),dimension(∗),intent(out) *Kuser,* REAL(8),intent(in) *t* )

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.10 DERIVED_TYPES::MATRIXTRANSFORM Type Reference

### Public Attributes

- REAL(8), dimension(4, 4) uvwr = EYE4
- LOGICAL iscalc = .false.

### 5.10.1 Member Data Documentation

#### 5.10.1.1 LOGICAL DERIVED_TYPES::MATRIXTRANSFORM::iscalc = .false.

#### 5.10.1.2 REAL(8),dimension(4,4) DERIVED_TYPES::MATRIXTRANSFORM::uvwr = EYE4

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.11 DERIVED_TYPES::POINT Type Reference

### Public Attributes

- REAL(8), dimension(3) pt
- REAL(8), dimension(:), pointer rpt
- REAL(8), dimension(:), pointer rptp
- INTEGER idpt

### 5.11.1 Member Data Documentation

#### 5.11.1.1 INTEGER DERIVED_TYPES::POINT::idpt

#### 5.11.1.2 REAL(8),dimension(3) DERIVED_TYPES::POINT::pt

#### 5.11.1.3 REAL(8),dimension(:),pointer DERIVED_TYPES::POINT::rpt

#### 5.11.1.4 REAL(8),dimension(:),pointer DERIVED_TYPES::POINT::rptp

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.12 DERIVED_TYPES::SOLID Type Reference

### Public Attributes

- TYPE(POINT), dimension(:), allocatable pt_loc
- TYPE(POINT), dimension(:), allocatable vc_loc
- REAL(8) mass = 0.d0
- REAL(8), dimension(3) rg_loc = 0.d0
- REAL(8), dimension(6) v_lg = 0.d0
- TYPE(MATRIXTRANSFORM) Mt
- TYPE(MATRIXTRANSFORM) Mtp
- INTEGER id = 0

- INTEGER npt_loc = 0
- INTEGER nvc_loc = 0
- INTEGER nz = 0

### 5.12.1 Member Data Documentation

#### 5.12.1.1 INTEGER DERIVED_TYPES::SOLID::id = 0

#### 5.12.1.2 REAL(8) DERIVED_TYPES::SOLID::mass = 0.d0

#### 5.12.1.3 TYPE(MATRIXTRANSFORM) DERIVED_TYPES::SOLID::Mt

#### 5.12.1.4 TYPE(MATRIXTRANSFORM) DERIVED_TYPES::SOLID::Mtp

#### 5.12.1.5 INTEGER DERIVED_TYPES::SOLID::npt_loc = 0

#### 5.12.1.6 INTEGER DERIVED_TYPES::SOLID::nvc_loc = 0

#### 5.12.1.7 INTEGER DERIVED_TYPES::SOLID::nz = 0

#### 5.12.1.8 TYPE (POINT),dimension(:),allocatable DERIVED_TYPES::SOLID::pt_loc

#### 5.12.1.9 REAL(8),dimension(3) DERIVED_TYPES::SOLID::rg_loc = 0.d0

#### 5.12.1.10 REAL(8),dimension(6) DERIVED_TYPES::SOLID::v_lg = 0.d0

#### 5.12.1.11 TYPE (POINT),dimension(:),allocatable DERIVED_TYPES::SOLID::vc_loc

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.13 DERIVED_TYPES::typeConstr_dot1 Type Reference

Dot-1 constraints.

### Public Attributes

- INTEGER ir
- INTEGER iEul1
- INTEGER iEul2
- real(8), dimension(3) u
- real(8), dimension(3) v

### 5.13.1 Detailed Description

Dot-1 constraints.

### 5.13.2 Member Data Documentation

#### 5.13.2.1 INTEGER DERIVED_TYPES::typeConstr_dot1::iEul1

#### 5.13.2.2 INTEGER DERIVED_TYPES::typeConstr_dot1::iEul2

#### 5.13.2.3 INTEGER DERIVED_TYPES::typeConstr_dot1::ir

#### 5.13.2.4 real(8),dimension(3) DERIVED_TYPES::typeConstr_dot1::u

#### 5.13.2.5 real(8),dimension(3) DERIVED_TYPES::typeConstr_dot1::v

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.14 DERIVED_TYPES::typeconstr_Drive_Dist Type Reference

Driving distance constraints.

### Public Attributes

- INTEGER ir
- INTEGER, dimension(3) irg1
- INTEGER, dimension(3) irg2
- INTEGER iEul1
- INTEGER, dimension(4) ieul2
- REAL(8), dimension(3) pt1
- REAL(8), dimension(3) pt2
- INTEGER i_MOTOR

### 5.14.1 Detailed Description

Driving distance constraints.

### 5.14.2 Member Data Documentation

#### 5.14.2.1 INTEGER DERIVED_TYPES::typeconstr_Drive_Dist::i_MOTOR

#### 5.14.2.2 INTEGER DERIVED_TYPES::typeconstr_Drive_Dist::iEul1

#### 5.14.2.3 INTEGER,dimension(4) DERIVED_TYPES::typeconstr_Drive_Dist::ieul2

#### 5.14.2.4 INTEGER DERIVED_TYPES::typeconstr_Drive_Dist::ir

#### 5.14.2.5 INTEGER,dimension(3) DERIVED_TYPES::typeconstr_Drive_Dist::irg1

#### 5.14.2.6 INTEGER,dimension(3) DERIVED_TYPES::typeconstr_Drive_Dist::irg2

#### 5.14.2.7 REAL(8),dimension(3) DERIVED_TYPES::typeconstr_Drive_Dist::pt1

#### 5.14.2.8 REAL(8),dimension(3) DERIVED_TYPES::typeconstr_Drive_Dist::pt2

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.15 DERIVED_TYPES::typeconstr_Drive_rgEul Type Reference

Driving constraints coordinates.

### Public Attributes

- INTEGER ir
- INTEGER ind
- INTEGER i_MOTOR

### 5.15.1 Detailed Description

Driving constraints coordinates.

### 5.15.2 Member Data Documentation

#### 5.15.2.1 INTEGER DERIVED_TYPES::typeconstr_Drive_rgEul::i_MOTOR

#### 5.15.2.2 INTEGER DERIVED_TYPES::typeconstr_Drive_rgEul::ind

#### 5.15.2.3 INTEGER DERIVED_TYPES::typeconstr_Drive_rgEul::ir

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.16 DERIVED_TYPES::typeConstr_RevJoint Type Reference

Revolute joint constraints.

### Public Attributes

- INTEGER ir
- INTEGER, dimension(3) irg1
- INTEGER, dimension(3) irg2
- INTEGER iEul1
- INTEGER iEul2
- REAL(8), dimension(3) pt1
- REAL(8), dimension(3) pt2
- REAL(8), dimension(3) vec1
- REAL(8), dimension(3) vec2
- REAL(8), dimension(3) uPerp1
- REAL(8), dimension(3) vPerp1

### 5.16.1 Detailed Description

Revolute joint constraints.

### 5.16.2 Member Data Documentation

#### 5.16.2.1 INTEGER DERIVED_TYPES::typeConstr_RevJoint::iEul1

#### 5.16.2.2 INTEGER DERIVED_TYPES::typeConstr_RevJoint::iEul2

#### 5.16.2.3 INTEGER DERIVED_TYPES::typeConstr_RevJoint::ir

#### 5.16.2.4 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_RevJoint::irg1

#### 5.16.2.5 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_RevJoint::irg2

#### 5.16.2.6 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::pt1

#### 5.16.2.7 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::pt2

#### 5.16.2.8 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::uPerp1

#### 5.16.2.9 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::vec1

#### 5.16.2.10 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::vec2

#### 5.16.2.11 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_RevJoint::vPerp1

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.17 DERIVED_TYPES::typeConstr_SpheJoint Type Reference

Spherical joint constraints.

### Public Attributes

- INTEGER ir
- INTEGER, dimension(3) irg1
- INTEGER, dimension(3) irg2
- INTEGER iEul1
- INTEGER iEul2
- REAL(8), dimension(3) pt1
- REAL(8), dimension(3) pt2

### 5.17.1 Detailed Description

Spherical joint constraints.

### 5.17.2 Member Data Documentation

#### 5.17.2.1 INTEGER DERIVED_TYPES::typeConstr_SpheJoint::iEul1

#### 5.17.2.2 INTEGER DERIVED_TYPES::typeConstr_SpheJoint::iEul2

#### 5.17.2.3 INTEGER DERIVED_TYPES::typeConstr_SpheJoint::ir

#### 5.17.2.4 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_SpheJoint::irg1

#### 5.17.2.5 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_SpheJoint::irg2

#### 5.17.2.6 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_SpheJoint::pt1

#### 5.17.2.7 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_SpheJoint::pt2

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.18 DERIVED_TYPES::typeConstr_TransJoint Type Reference

Translational joint constraints.

### Public Attributes

- INTEGER ir
- INTEGER, dimension(3) irg1
- INTEGER, dimension(3) irg2
- INTEGER iEul1
- INTEGER iEul2
- REAL(8), dimension(3) pt1
- REAL(8), dimension(3) pt2
- REAL(8), dimension(3) vec1x
- REAL(8), dimension(3) vec1y
- REAL(8), dimension(3) vec2x
- REAL(8), dimension(3) vec2y
- REAL(8), dimension(3) vec2z

### 5.18.1 Detailed Description

Translational joint constraints.

### 5.18.2 Member Data Documentation

#### 5.18.2.1 INTEGER DERIVED_TYPES::typeConstr_TransJoint::iEul1

#### 5.18.2.2 INTEGER DERIVED_TYPES::typeConstr_TransJoint::iEul2

#### 5.18.2.3 INTEGER DERIVED_TYPES::typeConstr_TransJoint::ir

#### 5.18.2.4 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_TransJoint::irg1

#### 5.18.2.5 INTEGER,dimension(3) DERIVED_TYPES::typeConstr_TransJoint::irg2

##### 5.18.2.6 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::pt1

##### 5.18.2.7 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::pt2

##### 5.18.2.8 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::vec1x

##### 5.18.2.9 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::vec1y

##### 5.18.2.10 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::vec2x

##### 5.18.2.11 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::vec2y

##### 5.18.2.12 REAL(8),dimension(3) DERIVED_TYPES::typeConstr_TransJoint::vec2z

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.19 DERIVED_TYPES::typeConstr_UnitEulParam Type Reference

Euler parameters constraints.

### Public Attributes

- INTEGER ir
- INTEGER iEul

### 5.19.1 Detailed Description

Euler parameters constraints.

### 5.19.2 Member Data Documentation

##### 5.19.2.1 INTEGER DERIVED_TYPES::typeConstr_UnitEulParam::iEul

##### 5.19.2.2 INTEGER DERIVED_TYPES::typeConstr_UnitEulParam::ir

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

## 5.20 DERIVED_TYPES::typeforce_TSDA Type Reference

TSDA forces.

### Public Attributes

- INTEGER body1
- INTEGER body2
- REAL(8), dimension(3) pt1
- REAL(8), dimension(3) pt2
- REAL(8), dimension(3) F1
- REAL(8), dimension(3) F2
- REAL(8) k
- REAL(8) c
- REAL(8) s0

### 5.20.1 Detailed Description

TSDA forces.

### 5.20.2 Member Data Documentation

#### 5.20.2.1 INTEGER DERIVED_TYPES::typeforce_TSDA::body1

#### 5.20.2.2 INTEGER DERIVED_TYPES::typeforce_TSDA::body2

#### 5.20.2.3 REAL(8) DERIVED_TYPES::typeforce_TSDA::c

#### 5.20.2.4 REAL(8),dimension(3) DERIVED_TYPES::typeforce_TSDA::F1

#### 5.20.2.5 REAL(8),dimension(3) DERIVED_TYPES::typeforce_TSDA::F2

#### 5.20.2.6 REAL(8) DERIVED_TYPES::typeforce_TSDA::k

#### 5.20.2.7 REAL(8),dimension(3) DERIVED_TYPES::typeforce_TSDA::pt1

#### 5.20.2.8 REAL(8),dimension(3) DERIVED_TYPES::typeforce_TSDA::pt2

#### 5.20.2.9 REAL(8) DERIVED_TYPES::typeforce_TSDA::s0

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

- D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_-types.f90

# Chapter 6

# File Documentation

## 6.1 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/constants.f90 File Reference

### Modules

- module CONSTANTS

  *Module of solver parameters.*

### Functions/Subroutines

- subroutine CONSTANTS::initialize_CONSTANTS (formulation, integrator, time_-step, penaltycoef, psicoef, omegacoef, gravity)

  *Inicialization of solver constants and parameters.*
- subroutine CONSTANTS::initialize_CALLBACKS (forces, stiffness, damping, PQbarPrho, PMbarPrhoVdot, dgdy, dgdp, AdjInit, gfun)

  *Initialization of solver callbacks: the user provides the subtoutines that the solver calls if necessary. It needs a previous call to constants::initialize_CONSTANTS.*
- subroutine CONSTANTS::setDIM (newDIM)
- subroutine CONSTANTS::setNRT (newNRT)
- subroutine CONSTANTS::setNMT (newNMT)
- subroutine CONSTANTS::setNIN (newNIN)

### Variables

- REAL(8), dimension(3), pointer CONSTANTS::PROTECTED
- REAL(8), dimension(3) CONSTANTS::g = (/0.d0,0.d0,-9.81d0/)
- REAL(8) CONSTANTS::dt = 1.d-2
- REAL(8) CONSTANTS::alfa = 1.d9
- REAL(8) CONSTANTS::psi = 1.d0

- REAL(8) CONSTANTS::omega = 10.d0
- REAL(8) CONSTANTS::tolNRppos = 1.d-10
- REAL(8) CONSTANTS::pivgdlval = 1.d15
- INTEGER CONSTANTS::maxiteppos = 1000
- INTEGER, parameter CONSTANTS::Dynamics = 1
- INTEGER, parameter CONSTANTS::Kinematics = 2
- INTEGER, parameter CONSTANTS::Sensitivity_ADJ = 3
- INTEGER, parameter CONSTANTS::Sensitivity_TLM = 4
- INTEGER, parameter CONSTANTS::E_RK = 1
- INTEGER, parameter CONSTANTS::E_RK2 = 2
- INTEGER, parameter CONSTANTS::I_RK = 3
- INTEGER, parameter CONSTANTS::I_RK_ADJ = 4
- INTEGER, parameter CONSTANTS::I_RK_TLM = 5
- INTEGER CONSTANTS::SWFORM = Dynamics
- INTEGER CONSTANTS::SWINT = E_RK
- INTEGER CONSTANTS::DIM = 0
- INTEGER CONSTANTS::NRT = 0
- INTEGER CONSTANTS::NIN = 0
- INTEGER CONSTANTS::NMT = 0
- PROCEDURE(callback_forces), pointer CONSTANTS::pforces_user
- PROCEDURE(callback_stiffness), pointer CONSTANTS::pstiffness_user
- PROCEDURE(callback_damping), pointer CONSTANTS::pdamping_user
- PROCEDURE(callback_PQbarPrho), pointer CONSTANTS::pqro_user
- PROCEDURE(callback_PMbarPrhoVdot), pointer CONSTANTS::pmpv_user
- PROCEDURE(callback_dgdy), pointer CONSTANTS::pdgdy_user
- PROCEDURE(callback_dgdp), pointer CONSTANTS::pdgdp_user
- PROCEDURE(callback_AdjInit), pointer CONSTANTS::padjinit_user
- PROCEDURE(callback_gfun), pointer CONSTANTS::pgfun_user

## 6.2 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/constraints.f90 File Reference

### Modules

- module CONSTRAINTS

  *Module that manages the constraints.*

### Functions/Subroutines

- subroutine, public CONSTRAINTS::ADDConstr_UnitEulParam (body)

  *Adds an unitary vector constraint to the Euler parameters of one body. It's NOT a user function, it's intended to be called by the solver.*
- subroutine, public CONSTRAINTS::ADDConstr_dot1 (body1, body2, vector1, vector2)

*Adds a dot-1 joint between two bodies or between the ground and one body.*

- subroutine, public CONSTRAINTS::ADDConstr_SpheJoint (body1, body2, point1, point2)

  *Adds a spherical joint between two bodies or between the ground and one body.*

- subroutine, public CONSTRAINTS::ADDConstr_RevJoint (body1, body2, point1, point2, vect1, vect2)

  *Adds a revolute joint between two bodies or between the ground and one body.*

- subroutine, public CONSTRAINTS::ADDConstr_TransJoint (body1, body2, point1, point2, vect1x, vect1y, vect2x, vect2y)

  *Adds a translational joint between two bodies or between the ground and one body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_rgx (body, i_MOTOR)

  *Adds a driving constraint to the x-coordinate of the CDM of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_rgy (body, i_MOTOR)

  *Adds a driving constraint to the y-coordinate of the CDM of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_rgz (body, i_MOTOR)

  *Adds a driving constraint to the z-coordinate of the CDM of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_eu0 (body, i_MOTOR)

  *Adds a driving constraint to the 1st Euler parameter of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_eu1 (body, i_MOTOR)

  *Adds a driving constraint to the 2nd Euler parameter of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_eu2 (body, i_MOTOR)

  *Adds a driving constraint to the 3rd Euler parameter of a body.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_eu3 (body, i_MOTOR)

  *Adds a driving constraint to the 4rd Euler parameter of a body.*

- subroutine CONSTRAINTS::ADDConstr_Drive_rgEul (ind, i_MOTOR)

  *Adds a driving constraint to any body coordinate. It's NOT a user function, it's intended to be called by other user constraints.*

- subroutine, public CONSTRAINTS::ADDConstr_Drive_dist (body1, body2, point1, point2, i_MOTOR)

  *Adds a driving constraint to a distance between two points in two different bodies.*

- subroutine, public CONSTRAINTS::evalconstraints

  *Subroutine that evaluates the constraints vector ( $\mathbf{\Phi}$). It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public CONSTRAINTS::evaljacobian

  *Subroutine that evaluates the jacobian of the constraints vector ( $\mathbf{\Phi_q}$). It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public CONSTRAINTS::evalderjacobianqp

  *Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}}$. It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public CONSTRAINTS::evalderjacobian

  *Subroutine that evaluates the term $\dot{\mathbf{\Phi}}_\mathbf{q}$. It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public CONSTRAINTS::evalderderjacobian

  *Subroutine that evaluates the term $\ddot{\mathbf{\Phi}}_\mathbf{q}$. It's NOT a user function, it's intended to be called by the solver.*

- subroutine, public CONSTRAINTS::evaljacobderjacobianqp
- subroutine, public CONSTRAINTS::evalfit
- subroutine, public CONSTRAINTS::evalfitp
- subroutine, public CONSTRAINTS::evaljacob_jacob (lb)
- subroutine, public CONSTRAINTS::evaljacobT_jacob (lb)
- subroutine, public CONSTRAINTS::CONSTRAINTS_Setup

## Variables

- INTEGER CONSTRAINTS::nConstr_UnitEulParam = 0
- INTEGER CONSTRAINTS::nConstr_dot1GB = 0
- INTEGER CONSTRAINTS::nConstr_dot1 = 0
- INTEGER CONSTRAINTS::nConstr_SpheJointGB = 0
- INTEGER CONSTRAINTS::nConstr_SpheJoint = 0
- INTEGER CONSTRAINTS::nConstr_RevJointGB = 0
- INTEGER CONSTRAINTS::nConstr_RevJoint = 0
- INTEGER CONSTRAINTS::nConstr_TransJointGB = 0
- INTEGER CONSTRAINTS::nConstr_TransJoint = 0
- INTEGER CONSTRAINTS::nConstr_Drive_rgEul = 0
- INTEGER CONSTRAINTS::nConstr_Drive_distGB = 0
- INTEGER CONSTRAINTS::nConstr_Drive_dist = 0
- TYPE(typeConstr_UnitEulParam), dimension(:), allocatable CONSTRAINTS::Constr_-UnitEulParam
- TYPE(typeConstr_dot1), dimension(:), allocatable CONSTRAINTS::Constr_dot1GB
- TYPE(typeConstr_dot1), dimension(:), allocatable CONSTRAINTS::Constr_dot1
- TYPE(typeConstr_SpheJoint), dimension(:), allocatable CONSTRAINTS::Constr_-SpheJointGB
- TYPE(typeConstr_SpheJoint), dimension(:), allocatable CONSTRAINTS::Constr_-SpheJoint
- TYPE(typeConstr_RevJoint), dimension(:), allocatable CONSTRAINTS::Constr_-RevJointGB
- TYPE(typeConstr_RevJoint), dimension(:), allocatable CONSTRAINTS::Constr_-RevJoint
- TYPE(typeConstr_TransJoint), dimension(:), allocatable CONSTRAINTS::Constr_-TransJointGB
- TYPE(typeConstr_TransJoint), dimension(:), allocatable CONSTRAINTS::Constr_-TransJoint
- TYPE(typeconstr_Drive_rgEul), dimension(:), allocatable CONSTRAINTS::Constr_-Drive_rgEul
- TYPE(typeconstr_Drive_Dist), dimension(:), allocatable CONSTRAINTS::Constr_-Drive_DistGB
- TYPE(typeconstr_Drive_Dist), dimension(:), allocatable CONSTRAINTS::Constr_-Drive_Dist

## 6.3 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/d2jacobdt2.f90 File Reference

### Modules

- module d2Jacobdt2

  *Module of second derivatives of the Jacobian. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine d2Jacobdt2::d2Jacobdt2_Setup
- subroutine d2Jacobdt2::deallocFiqpp
- subroutine d2Jacobdt2::ddj_UnitEulParam (ir, iEul)

  *The second derivatives of the jacobians of unitary Euler parameters.*
- subroutine d2Jacobdt2::ddj_dot1GB (ir, iEul2, u, v)

  *The second derivative of the jacobians of a dot-1 constraint attached on the ground.*
- subroutine d2Jacobdt2::ddj_dot1 (ir, iEul1, iEul2, u, v)

  *The second derivatives of the jacobians of the jacobians of a dot-1 constraint.*
- subroutine d2Jacobdt2::ddj_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *The second derivatives of the jacobians of of a spherical joint of a body attached to the ground.*
- subroutine d2Jacobdt2::ddj_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *The second derivatives of the jacobians of of a spherical joint between two bodies.*
- subroutine d2Jacobdt2::ddj_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *The second derivatives of the jacobians of of a revolute joint of a body attached to the ground.*
- subroutine d2Jacobdt2::ddj_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *The second derivatives of the jacobians of of a revolute joint between two bodies.*
- subroutine d2Jacobdt2::ddj_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The second derivatives of the jacobians of of a translational joint of a body attached to the ground.*
- subroutine d2Jacobdt2::ddj_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The second derivatives of the jacobians of a translational joint between two bodies.*
- subroutine d2Jacobdt2::ddj_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *The second derivatives of the jacobians for a distance between a point in the ground and a point of one body.*
- subroutine d2Jacobdt2::ddj_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *The second derivatives of the jacobians for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable d2Jacobdt2::PROTECTED
- REAL(8), dimension(:,:), allocatable d2Jacobdt2::Fiqpp

## 6.4 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/derived_types.f90 File Reference

### Data Types

- type DERIVED_TYPES::MATRIXTRANSFORM
- type DERIVED_TYPES::POINT
- type DERIVED_TYPES::SOLID
- type DERIVED_TYPES::typeConstr_UnitEulParam

  *Euler parameters constraints.*
- type DERIVED_TYPES::typeConstr_dot1

  *Dot-1 constraints.*
- type DERIVED_TYPES::typeConstr_SpheJoint

  *Spherical joint constraints.*
- type DERIVED_TYPES::typeConstr_RevJoint

  *Revolute joint constraints.*
- type DERIVED_TYPES::typeConstr_TransJoint

  *Translational joint constraints.*
- type DERIVED_TYPES::typeconstr_Drive_rgEul

  *Driving constraints coordinates.*
- type DERIVED_TYPES::typeconstr_Drive_Dist

  *Driving distance constraints.*
- type DERIVED_TYPES::typeforce_TSDA

  *TSDA forces.*
- interface DERIVED_TYPES::callback_forces
- interface DERIVED_TYPES::callback_stiffness
- interface DERIVED_TYPES::callback_damping
- interface DERIVED_TYPES::callback_PQbarPrho
- interface DERIVED_TYPES::callback_PMbarPrhoVdot
- interface DERIVED_TYPES::callback_dgdy
- interface DERIVED_TYPES::callback_dgdp
- interface DERIVED_TYPES::callback_AdjInit
- interface DERIVED_TYPES::callback_gfun

### Modules

- module DERIVED_TYPES

  *Module of solver derived types definition and subroutines/functions to manage the derived types.*

### Functions/Subroutines

- INTEGER, dimension(7) DERIVED_TYPES::indre (nSOLID)

  *Function that returns the index for all the states of the body.*
- INTEGER, dimension(3) DERIVED_TYPES::indrg (nSOLID)

  *Function that returns the index for the CDM of the body.*
- INTEGER DERIVED_TYPES::indrgx (nSOLID)
- INTEGER DERIVED_TYPES::indrgy (nSOLID)
- INTEGER DERIVED_TYPES::indrgz (nSOLID)
- INTEGER, dimension(4) DERIVED_TYPES::indeu (nSOLID)

  *Function that returns the index for the Euler parameters of the body.*
- INTEGER DERIVED_TYPES::indeu0 (nSOLID)
- INTEGER DERIVED_TYPES::indeu1 (nSOLID)
- INTEGER DERIVED_TYPES::indeu2 (nSOLID)
- INTEGER DERIVED_TYPES::indeu3 (nSOLID)

## 6.5 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/djacobdt.f90 File Reference

### Modules

- module dJacobdt

  *Module of total derivatives of the Jacobian. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine dJacobdt::dJacobdt_Setup
- subroutine dJacobdt::deallocFiqp
- subroutine dJacobdt::dj_UnitEulParam (ir, iEul)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC.*
- subroutine dJacobdt::dj_dot1GB (ir, iEul2, u, v)

  *The first derivative of the jacobians of a dot-1 constraint attached on the ground.*
- subroutine dJacobdt::dj_dot1 (ir, iEul1, iEul2, u, v)

  *The first derivative of the jacobians of a dot-1 constraint.*
- subroutine dJacobdt::dj_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *The first derivative of the jacobians of a spherical joint of a body attached to the ground.*
- subroutine dJacobdt::dj_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *The first derivative of the jacobians of a spherical joint between two bodies.*
- subroutine dJacobdt::dj_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  *The first derivative of the jacobians of a revolute joint of a body attached to the ground.*
- subroutine dJacobdt::dj_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

*The first derivative of the jacobians of a revolute joint between two bodies.*

- subroutine dJacobdt::dj_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The first derivative of the jacobians of a translational joint of a body attached to the ground.*
- subroutine dJacobdt::dj_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *The first derivative of the jacobians of a translational joint between two bodies.*
- subroutine dJacobdt::dj_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *The first derivative of the jacobians for a distance between a point in the ground and a point of one body.*
- subroutine dJacobdt::dj_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  *The first derivative of the jacobians for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable dJacobdt::PROTECTED
- REAL(8), dimension(:,:), allocatable dJacobdt::Fiqp

## 6.6 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/djacobdt_qp.f90 File Reference

### Modules

- module djacobdt_qp

  *Module of derivatives of the Jacobian mutiplied by the velocity vector. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine djacobdt_qp::djacobdt_qp_Setup
- subroutine djacobdt_qp::deallocfiqpqp
- subroutine djacobdt_qp::deallocfit
- subroutine djacobdt_qp::d_UnitEulParam (ir, iEul)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$*, which is the derevative of jacobian with respect to time multiplies the velocity vector* $\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of unitary Euler parameters*
- subroutine djacobdt_qp::d_dot1GB (ir, iEul2, u, v)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a dot-1 constraint attached on the ground*
- subroutine djacobdt_qp::d_dot1 (ir, iEul1, iEul2, u, v)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a dot-1 constraint.*
- subroutine djacobdt_qp::d_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  $\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a spherical joint of a body attached to the ground*

- subroutine djacobdt_qp::d_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a spherical joint between two bodies*
- subroutine djacobdt_qp::d_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a revolute joint of a body attached to the ground*
- subroutine djacobdt_qp::d_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a revolute joint between two bodies*
- subroutine djacobdt_qp::d_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a translational joint of a body attached to the ground*
- subroutine djacobdt_qp::d_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *of a translational joint between two bodies*
- subroutine djacobdt_qp::d_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance between a point in the ground and a point of one body.*
- subroutine djacobdt_qp::d_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance between two points of two bodies.*
- subroutine djacobdt_qp::dt_Drive_rgEul (ir, ind, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a generalized coordinate of the system.*
- subroutine djacobdt_qp::dtp_Drive_rgEul (ir, ind, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a generalized coordinate of the system.*
- subroutine djacobdt_qp::dt_Drive_dist (ir, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance.*
- subroutine djacobdt_qp::dtp_Drive_dist (ir, i_MOTOR)

  $\dot{\Phi}_{\mathbf{q}}\dot{\mathbf{q}}$ *for a distance.*

### Variables

- REAL(8), dimension(:), allocatable djacobdt_qp::PROTECTED
- REAL(8), dimension(:), allocatable djacobdt_qp::fiqpqp
- REAL(8), dimension(:), allocatable djacobdt_qp::fit
- REAL(8), dimension(:), allocatable djacobdt_qp::fitp

## 6.7 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/forceold.f90 File Reference

### Modules

- module forces

### Functions/Subroutines

- subroutine forces::force (t, n, F, p, Q)

  *Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.*
- subroutine forces::TSDA (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized forces of a translational spring-damper-actuator between acting on two bodies.*
- subroutine forces::TSDA_q (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine forces::TSDA_qp (t, body1, body2, pt1, pt2, s0, k, c, Q1, Q2)

  *Function to get the generalized damping of a translational spring-damper-actuator between acting on two bodies.*

## 6.8 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/forces.f90 File Reference

### Modules

- module forces

### Functions/Subroutines

- subroutine forces::TSDA (r1, r2, r1p, r2p, s0, k, c, F1, F2)

  *Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.*
- subroutine forces::TSDA_q (r1, r2, r1p, r2p, s0, k, c, df1dr1, df1dr2, df2dr1, df2dr2)

  *Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine forces::TSDA_qp (r1, r2, c, df1dr1p, df1dr2p, df2dr1p, df2dr2p)

  *Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.*

## 6.9 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_dynamics.f90 File Reference

### Modules

- module formulation_Dynamics

  *Dynamic simulation module.*

### Functions/Subroutines

- subroutine formulation_Dynamics::Acceleration_penalty (t)

  *Subrutine that solves the equations of motion for the acceleration using penalty (Partially taken from MBSLIM)*
- subroutine formulation_Dynamics::Penalty_fun (NVAR, t, y, yp)
- subroutine formulation_Dynamics::Penalty_Tang (N, T, Y, Fy)

## 6.10 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_kinematics.f90 File Reference

### Modules

- module formulation_Kinematics

  *Kinematic simulation module.*

### Functions/Subroutines

- subroutine formulation_Kinematics::position_kinematics (C, name)

  *Solves the position problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*
- subroutine formulation_Kinematics::velocity_kinematics (C, name)

  *Solves the velocity problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*
- subroutine formulation_Kinematics::acceleration_kinematics (C, name)

  *Solves the acceleration problem in terms of the degrees of freedom of the system (Partially taken from MBSLIM)*

## 6.11 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulation_sensitivity.f90 File Reference

### Modules

- module formulation_Sensitivity

  *Sensitivity analyis module.*

### Functions/Subroutines

- subroutine formulation_Sensitivity::Penalty_Jacp (N, NP, T, Y, FPJAC)

## 6.12 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/formulations.f90 File Reference

### Modules

- module formulations

  *Module of generic formulations. Contains the generic functions that manage the use of different formulations.*

### Functions/Subroutines

- subroutine formulations::acceleration_dynamics (t)

  *Generic subroutine for the acceleration calculation.*
- subroutine formulations::integration_dynamics (TIN, TOUT, RTOL, ATOL, POSTSTEP)

  *Generic subroutine for the integration of the equations of motion.*
- subroutine formulations::integration_sensitivity (NP, NADJ, NNZERO, VAR, Lambda, TIN, TOUT, ATOL_adj, RTOL_adj, ATOL, RTOL, Mu, objval)

  *Generic subroutine for the integration of the equations of motion.*
- subroutine formulations::Model_Setup

  *Generic subroutine to set up the models.*

## 6.13 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/generalized_forces.f90 File Reference

### Modules

- module generalized_forces

  *Generalized forces module.*

### Functions/Subroutines

- subroutine, public generalized_forces::ADDforce_TSDA (body1, body2, pt1, pt2, k, c, s0)
- subroutine, public generalized_forces::evalgenforces (t)

  *Subroutine to evaluate the generalized forces of the system.*
- subroutine, public generalized_forces::evalgenstiffdamp (t)

  *Subroutine to evaluate the generalized stiffness and damping of the system.*
- subroutine generalized_forces::evalprimitiveforces (t, Q_Prim)

  *Subroutine to evaluate the primitive forces of the system.*
- subroutine generalized_forces::evalprimstiffdamp (t, K_Prim, C_Prim)

  *Subroutine to evaluate the primitive stiffness and damping of the system.*

- subroutine generalized_forces::genForce1body (n, F, p, Q)

  *Subroutine to form the local generalized force over one body Function to get the generalized force of one body when torque, force and Euler parameters of this body are given.*
- subroutine, public generalized_forces::generalized_forces_Setup

  *Generalized forces module setup.*

### Variables

- REAL(8), dimension(:,:), allocatable, public generalized_forces::PROTECTED
- REAL(8), dimension(:), allocatable, public generalized_forces::Qgen
- REAL(8), dimension(:,:), allocatable, public generalized_forces::Kgen
- REAL(8), dimension(:,:), allocatable, public generalized_forces::Cgen
- REAL(8), dimension(:), allocatable generalized_forces::Qgrav
- REAL(8), dimension(:,:), allocatable generalized_forces::Kgrav
- REAL(8), dimension(:,:), allocatable generalized_forces::Cgrav
- INTEGER generalized_forces::nforce_TSDA = 0
- TYPE(typeforce_TSDA), dimension(:), allocatable generalized_forces::force_TSDA

## 6.14 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob.f90 File Reference

### Modules

- module Jacob

  *Module of primitive jacobians. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine Jacob::Jacob_Setup
- subroutine Jacob::deallocFiq
- subroutine Jacob::j_UnitEulParam (ir, iEul)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*
  *Primitive jacobian of unitary Euler parameters.*
- subroutine Jacob::j_dot1GB (ir, iEul2, u, v)

  *Primitive dot-1 jacobian of a body attached on the ground.*
- subroutine Jacob::j_dot1 (ir, iEul1, iEul2, u, v)

  *Primitive dot-1 jacobian.*
- subroutine Jacob::j_sphericalGB (ir, irg2, iEul2, pt1, pt2)

  *Primitive jacobians of a spherical joint of a body attached to the ground.*
- subroutine Jacob::j_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

  *Primitive jacobians of a spherical joint between two bodies.*
- subroutine Jacob::j_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

*Primitive jacobians of a revolute joint of a body attached to the ground.*
- subroutine Jacob::j_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  *Primitive jacobians of a revolute joint between two bodies.*
- subroutine Jacob::j_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive jacobians of a translational joint of a body attached to the ground.*
- subroutine Jacob::j_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  *Primitive jacobians of a translational joint between two bodies.*
- subroutine Jacob::j_Drive_rgEul (ir, ind, i_MOTOR)

  *Primitive driving jacobians for a generalized coordinate of the system.*
- subroutine Jacob::j_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

  *Primitive driving jacobians for a distance between a point in the ground and a point of one body.*
- subroutine Jacob::j_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_-MOTOR)

  *Primitive driving constraints for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable Jacob::PROTECTED
- REAL(8), dimension(:,:), allocatable Jacob::Fiq

## 6.15 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_djacobdt_qp.f90 File Reference

### Modules

- module jacob_djacobdt_qp

  *Module of* $(\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}$*. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine jacob_djacobdt_qp::jacob_djacobdt_qp_Setup
- subroutine jacob_djacobdt_qp::deallocFiqpqpq
- subroutine jacob_djacobdt_qp::dq_dot1 (ir, iEul1, iEul2, u, v)

  $\dot{\mathbf{\Phi}}\mathbf{q}\dot{\mathbf{q}}$ *of a dot-1 constraint.*
- subroutine jacob_djacobdt_qp::dq_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

  $(\blacksquare\mathbf{q}\dot{\mathbf{q}})\mathbf{q}$ *of a revolute joint between two bodies*
- subroutine jacob_djacobdt_qp::dq_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $(\blacksquare\mathbf{q}\dot{\mathbf{q}})\mathbf{q}$ *of a translational joint of a body attached to the ground*

- subroutine jacob_djacobdt_qp::dq_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

  $(\blacksquare \mathbf{q}\dot{\mathbf{q}})\mathbf{q}$ *of a translational joint between two bodies*
- subroutine jacob_djacobdt_qp::dq_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_-MOTOR)

  $\dot{\mathbf{\Phi}}\mathbf{q}\dot{\mathbf{q}}$ *for a distance between a point in the ground and a point of one body.*
- subroutine jacob_djacobdt_qp::dq_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR)

  $\dot{\mathbf{\Phi}}\mathbf{q}\dot{\mathbf{q}}$ *for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable jacob_djacobdt_qp::PROTECTED
- REAL(8), dimension(:,:), allocatable jacob_djacobdt_qp::Fiqpqpq

## 6.16 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacob_jacob.f90 File Reference

### Modules

- module jacob_jacob

  *Module of* $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$*, which is the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine jacob_jacob::jacob_jacob_Setup
- subroutine jacob_jacob::deallocFiqqlb
- subroutine jacob_jacob::jjlb_UnitEulParam (ir, iEul, lb)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*
  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$ *of unitary Euler parameters.*
- subroutine jacob_jacob::jjlb_dot1GB (ir, iEul2, u, v, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$ *of a dot-1 jacobian of a body attached on the ground*
- subroutine jacob_jacob::jjlb_dot1 (ir, iEul1, iEul2, u, v, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$*of a dot-1 jacobian*
- subroutine jacob_jacob::jjlb_sphericalGB (ir, irg2, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$ *of a spherical joint of a body attached to the ground*
- subroutine jacob_jacob::jjlb_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$ *of a spherical joint between two bodies*
- subroutine jacob_jacob::jjlb_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}\mathbf{V}$ *of a revolute joint of a body attached to the ground*
- subroutine jacob_jacob::jjlb_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2, lb)

$\mathbf{\Phi_{qq}V}$ *of a revolute joint between two bodies*

- subroutine jacob_jacob::jjlb_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi_{qq}V}$ *of a translational joint of a body attached to the ground*
- subroutine jacob_jacob::jjlb_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi_{qq}V}$ *of a translational joint between two bodies*
- subroutine jacob_jacob::jjlb_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR, lb)

  $\mathbf{\Phi_{qq}V}$ *of a distance driving jacobians between a point in the ground and a point of one body.*
- subroutine jacob_jacob::jjlb_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_MOTOR, lb)

  *Primitive driving constraints for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable jacob_jacob::PROTECTED
- REAL(8), dimension(:,:), allocatable jacob_jacob::Fiqqlb

## 6.17 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/jacobT_jacob.f90 File Reference

### Modules

- module jacobT_jacob

  *Module of* $\mathbf{\Phi_{qq}^T V}$*, which is the transpose of the jacobian of the primitive jacobian multiplied by a vector. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine jacobT_jacob::jacobT_jacob_Setup
- subroutine jacobT_jacob::deallocFiqqtlb
- subroutine jacobT_jacob::resetFiqqtlb
- subroutine jacobT_jacob::jjtlb_UnitEulParam (ir, iEul, lb)

  *CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*
  $\mathbf{\Phi_{qq}^T V}$ *of unitary Euler parameters.*
- subroutine jacobT_jacob::jjtlb_dot1GB (ir, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of dot-1 jacobian of a body attached on the ground*
- subroutine jacobT_jacob::jjtlb_dot1 (ir, iEul1, iEul2, u, v, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of dot-1 jacobian*
- subroutine jacobT_jacob::jjtlb_sphericalGB (ir, irg2, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi_{qq}^T V}$ *of a spherical joint of a body attached to the ground*

- subroutine jacobT_jacob::jjtlb_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of a spherical joint between two bodies*
- subroutine jacobT_jacob::jjtlb_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of a revolute joint of a body attached to the ground*
- subroutine jacobT_jacob::jjtlb_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of a revolute joint between two bodies*
- subroutine jacobT_jacob::jjtlb_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of a translational joint of a body attached to the ground*
- subroutine jacobT_jacob::jjtlb_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of a translational joint between two bodies*
- subroutine jacobT_jacob::jjtlb_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of driving jacobians for a distance between a point in the ground and a point of one body.*
- subroutine jacobT_jacob::jjtlb_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_-loc, i_MOTOR, lb)

  $\mathbf{\Phi}_{\mathbf{qq}}^{\mathrm{T}}\mathbf{V}$ *of driving constraints for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:,:), allocatable jacobT_jacob::PROTECTED
- REAL(8), dimension(:,:), allocatable jacobT_jacob::Fiqqtlb

## 6.18 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/lbfgsb/blas.f File Reference

### Functions/Subroutines

- double precision dnrm2 (n, x, incx)
- subroutine daxpy (n, da, dx, incx, dy, incy)
- subroutine dcopy (n, dx, incx, dy, incy)
- double precision ddot (n, dx, incx, dy, incy)
- subroutine dscal (n, da, dx, incx)

### 6.18.1 Function Documentation

#### 6.18.1.1 subroutine daxpy ( integer *n*, double precision *da*, double precision,dimension($*$) *dx*, integer *incx*, double precision,dimension($*$) *dy*, integer *incy* )

#### 6.18.1.2 subroutine dcopy ( integer *n*, double precision,dimension(∗) *dx*, integer *incx*, double precision,dimension(∗) *dy*, integer *incy* )

#### 6.18.1.3 double precision ddot ( integer *n*, double precision,dimension(∗) *dx*, integer *incx*, double precision,dimension(∗) *dy*, integer *incy* )

#### 6.18.1.4 double precision dnrm2 ( integer *n*, double precision,dimension(n) *x*, integer *incx* )

#### 6.18.1.5 subroutine dscal ( integer *n*, double precision *da*, double precision,dimension(∗) *dx*, integer *incx* )

## 6.19 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/lbfgsb/lbfgsb.f File Reference

### Functions/Subroutines

- subroutine setulb (n, m, x, l, u, nbd, f, g, factr, pgtol, wa, iwa, task, iprint, csave, lsave, isave, dsave)
- subroutine mainlb (n, m, x, l, u, nbd, f, g, factr, pgtol, ws, wy, sy, ss, wt, wn, snd, z, r, d, t, xp, wa, index, iwhere, indx2, task, iprint, csave, lsave, isave, dsave)
- subroutine active (n, l, u, nbd, x, iwhere, iprint, prjctd, cnstnd, boxed)
- subroutine bmv (m, sy, wt, col, v, p, info)
- subroutine cauchy (n, x, l, u, nbd, g, iorder, iwhere, t, d, xcp, m, wy, ws, sy, wt, theta, col, head, p, c, wbp, v, nseg, iprint, sbgnrm, info, epsmch)
- subroutine cmprlb (n, m, x, g, ws, wy, sy, wt, z, r, wa, index, theta, col, head, nfree, cnstnd, info)
- subroutine errclb (n, m, factr, l, u, nbd, task, info, k)
- subroutine formk (n, nsub, ind, nenter, ileave, indx2, iupdat, updatd, wn, wn1, m, ws, wy, sy, theta, col, head, info)
- subroutine formt (m, wt, sy, ss, col, theta, info)
- subroutine freev (n, nfree, index, nenter, ileave, indx2, iwhere, wrk, updatd, cnstnd, iprint, iter)
- subroutine hpsolb (n, t, iorder, iheap)
- subroutine lnsrlb (n, l, u, nbd, x, f, fold, gd, gdold, g, d, r, t, z, stp, dnorm, dtd, xstep, stpmx, iter, ifun, iback, nfgv, info, task, boxed, cnstnd, csave, isave, dsave)
- subroutine matupd (n, m, ws, wy, sy, ss, d, r, itail, iupdat, col, head, theta, rr, dr, stp, dtd)
- subroutine prn1lb (n, m, l, u, x, iprint, itfile, epsmch)
- subroutine prn2lb (n, x, f, g, iprint, itfile, iter, nfgv, nact, sbgnrm, nseg, word, iword, iback, stp, xstep)
- subroutine prn3lb (n, x, f, task, iprint, info, itfile, iter, nfgv, nintol, nskip, nact, sbgnrm, time, nseg, word, iback, stp, xstep, k, cachyt, sbtime, lnscht)
- subroutine projgr (n, l, u, nbd, x, g, sbgnrm)
- subroutine subsm (n, m, nsub, ind, l, u, nbd, x, d, xp, ws, wy, theta, xx, gg, col, head, iword, wv, wn, iprint, info)
- subroutine dcsrch (f, g, stp, ftol, gtol, xtol, stpmin, stpmax, task, isave, dsave)
- subroutine dcstep (stx, fx, dx, sty, fy, dy, stp, fp, dp, brackt, stpmin, stpmax)

### 6.19.1 Function Documentation

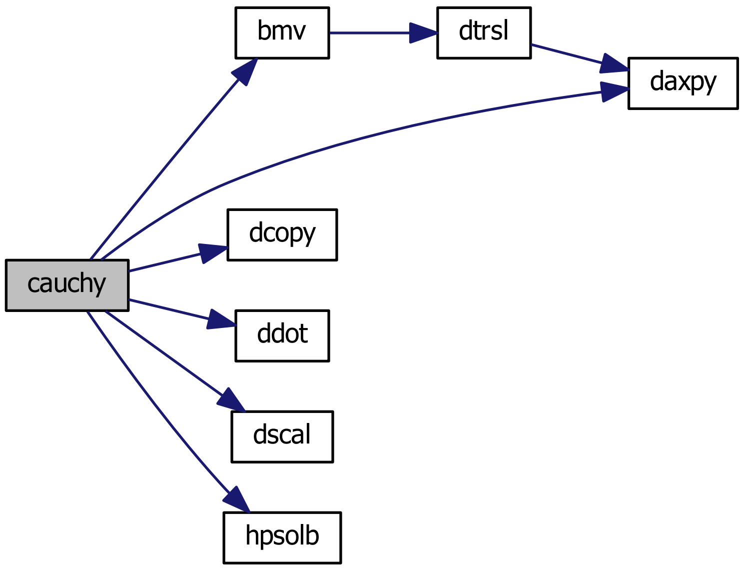
#### 6.19.1.1 subroutine active ( integer *n*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision,dimension(n) *x*, integer,dimension(n) *iwhere*, integer *iprint*, logical *prjctd*, logical *cnstnd*, logical *boxed* )

#### 6.19.1.2 subroutine bmv ( integer *m*, double precision,dimension(m, m) *sy*, double precision,dimension(m, m) *wt*, integer *col*, double precision,dimension(2∗col) *v*, double precision,dimension(2∗col) *p*, integer *info* )

Here is the call graph for this function:

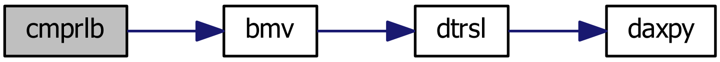
#### 6.19.1.3 subroutine cauchy ( integer *n*, double precision,dimension(n) *x*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision,dimension(n) *g*, integer,dimension(n) *iorder*, integer,dimension(n) *iwhere*, double precision,dimension(n) *t*, double precision,dimension(n) *d*, double precision,dimension(n) *xcp*, integer *m*, double precision,dimension(n, col) *wy*, double precision,dimension(n, col) *ws*, double precision,dimension(m, m) *sy*, double precision,dimension(m, m) *wt*, double precision *theta*, integer *col*, integer *head*, double precision,dimension(2∗m) *p*, double precision,dimension(2∗m) *c*, double precision,dimension(2∗m) *wbp*, double precision,dimension(2∗m) *v*, integer *nseg*, integer *iprint*, double precision *sbgnrm*, integer *info*, double precision *epsmch* )

Here is the call graph for this function:



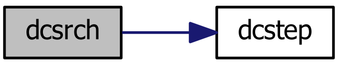
#### 6.19.1.4 subroutine cmprlb ( integer *n*, integer *m*, double precision,dimension(n) *x*, double precision,dimension(n) *g*, double precision,dimension(n, m) *ws*, double precision,dimension(n, m) *wy*, double precision,dimension(m, m) *sy*, double precision,dimension(m, m) *wt*, double precision,dimension(n) *z*, double precision,dimension(n) *r*, double precision,dimension(4∗m) *wa*, integer,dimension(n) *index*, double precision *theta*, integer *col*, integer *head*, integer *nfree*, logical *cnstnd*, integer *info* )

Here is the call graph for this function:




#### 6.19.1.5 subroutine dcsrch ( double precision *f*, double precision *g*, double precision *stp*, double precision *ftol*, double precision *gtol*, double precision *xtol*, double precision *stpmin*, double precision *stpmax*, character∗(∗) *task*, integer,dimension(2) *isave*, double precision,dimension(13) *dsave* )

Here is the call graph for this function:

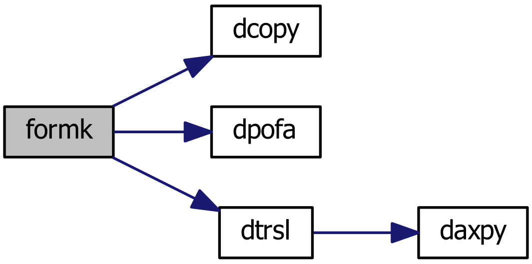



#### 6.19.1.6 subroutine dcstep ( double precision *stx*, double precision *fx*, double precision *dx*, double precision *sty*, double precision *fy*, double precision *dy*, double precision *stp*, double precision *fp*, double precision *dp*, logical *brackt*, double precision *stpmin*, double precision *stpmax* )

#### 6.19.1.7 subroutine errclb ( integer *n*, integer *m*, double precision *factr*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, character∗60 *task*, integer *info*, integer *k* )

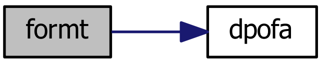
#### 6.19.1.8 subroutine formk ( integer *n*, integer *nsub*, integer,dimension(n) *ind*, integer *nenter*, integer *ileave*, integer,dimension(n) *indx2*, integer *iupdat*, logical *updatd*, double precision,dimension(2∗m, 2∗m) *wn*, double precision,dimension(2∗m, 2∗m) *wn1*, integer *m*, double precision,dimension(n, m) *ws*, double precision,dimension(n, m) *wy*, double precision,dimension(m, m) *sy*, double precision *theta*, integer *col*, integer *head*, integer *info* )

Here is the call graph for this function:




#### 6.19.1.9 subroutine formt ( integer *m*, double precision,dimension(m, m) *wt*, double precision,dimension(m, m) *sy*, double precision,dimension(m, m) *ss*, integer *col*, double precision *theta*, integer *info* )

Here is the call graph for this function:

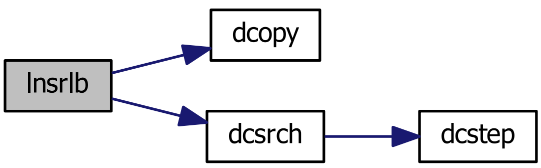



#### 6.19.1.10 subroutine freev ( integer *n*, integer *nfree*, integer,dimension(n) *index*, integer *nenter*, integer *ileave*, integer,dimension(n) *indx2*, integer,dimension(n) *iwhere*, logical *wrk*, logical *updatd*, logical *cnstnd*, integer *iprint*, integer *iter* )

#### 6.19.1.11 subroutine hpsolb ( integer *n*, double precision,dimension(n) *t*, integer,dimension(n) *iorder*, integer *iheap* )

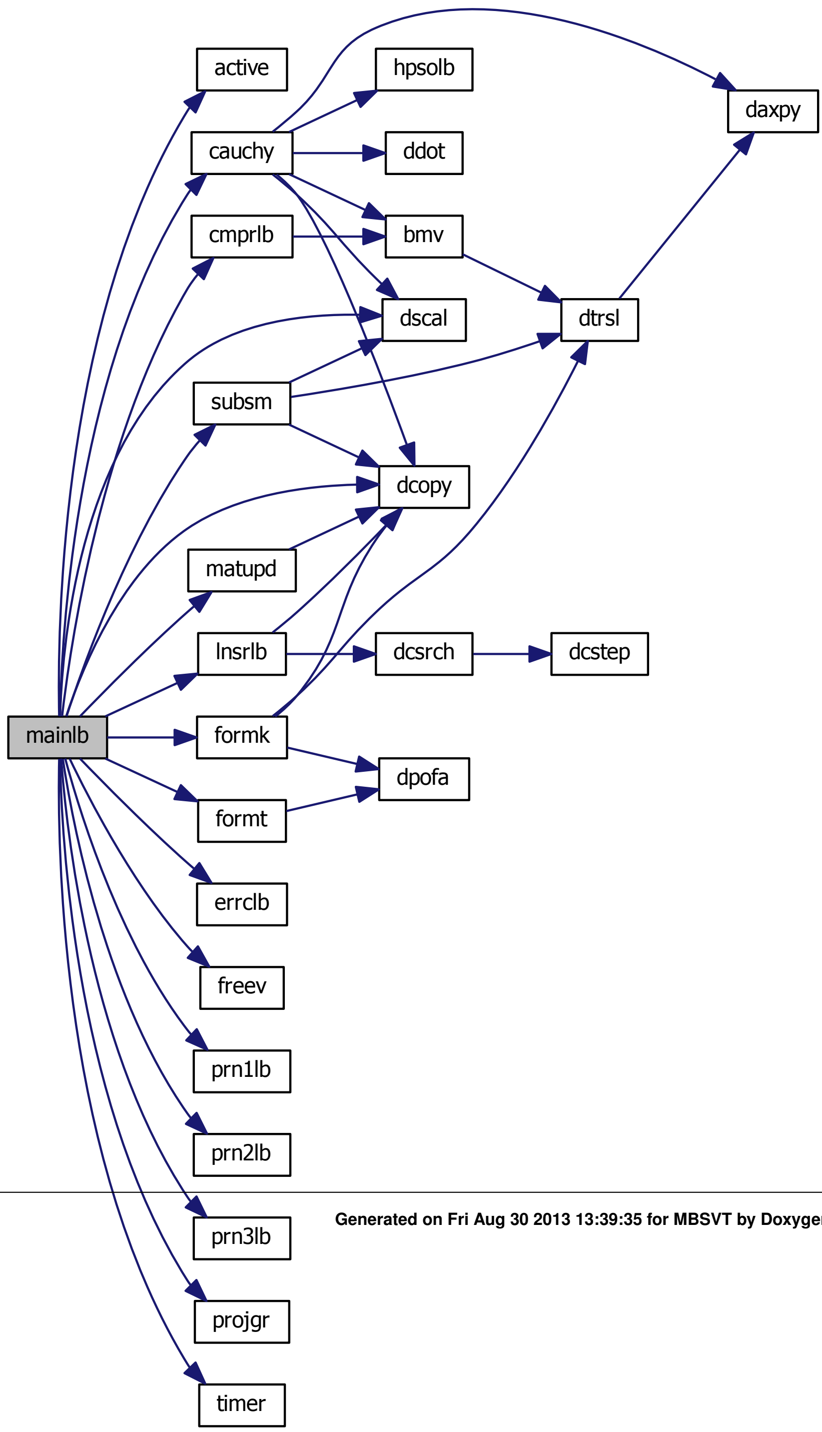
#### 6.19.1.12 subroutine lnsrlb ( integer *n*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision,dimension(n) *x*, double precision *f*, double precision *fold*, double precision *gd*, double precision *gdold*, double precision,dimension(n) *g*, double precision,dimension(n) *d*, double precision,dimension(n) *r*, double precision,dimension(n) *t*, double precision,dimension(n) *z*, double precision *stp*, double precision *dnorm*, double precision *dtd*, double precision *xstep*, double precision *stpmx*, integer *iter*, integer *ifun*, integer *iback*, integer *nfgv*, integer *info*, character∗60 *task*, logical *boxed*, logical *cnstnd*, character∗60 *csave*, integer,dimension(2) *isave*, double precision,dimension(13) *dsave* )

Here is the call graph for this function:



#### 6.19.1.13 subroutine mainlb ( integer *n*, integer *m*, double precision,dimension(n) *x*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision *f*, double precision,dimension(n) *g*, double precision *factr*, double precision *pgtol*, *ws*, *wy*, *sy*, *ss*, *wt*, *wn*, *snd*, double precision,dimension(n) *z*, double precision,dimension(n) *r*, double precision,dimension(n) *d*, double precision,dimension(n) *t*, *xp*, *wa*, integer,dimension(n) *index*, integer,dimension(n) *iwhere*, integer,dimension(n) *indx2*, character∗60 *task*, integer *iprint*, character∗60 *csave*, logical,dimension(4) *lsave*, integer,dimension(23) *isave*, *dsave* )

Here is the call graph for this function:



#### 6.19.1.14 subroutine matupd ( integer *n*, integer *m*, double precision,dimension(n, m) *ws*, double precision,dimension(n, m) *wy*, double precision,dimension(m, m) *sy*, double precision,dimension(m, m) *ss*, double precision,dimension(n) *d*, double precision,dimension(n) *r*, integer *itail*, integer *iupdat*, integer *col*, integer *head*, double precision *theta*, double precision *rr*, double precision *dr*, double precision *stp*, double precision *dtd* )

Here is the call graph for this function:


#### 6.19.1.15 subroutine prn1lb ( integer *n*, integer *m*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, double precision,dimension(n) *x*, integer *iprint*, integer *itfile*, double precision *epsmch* )

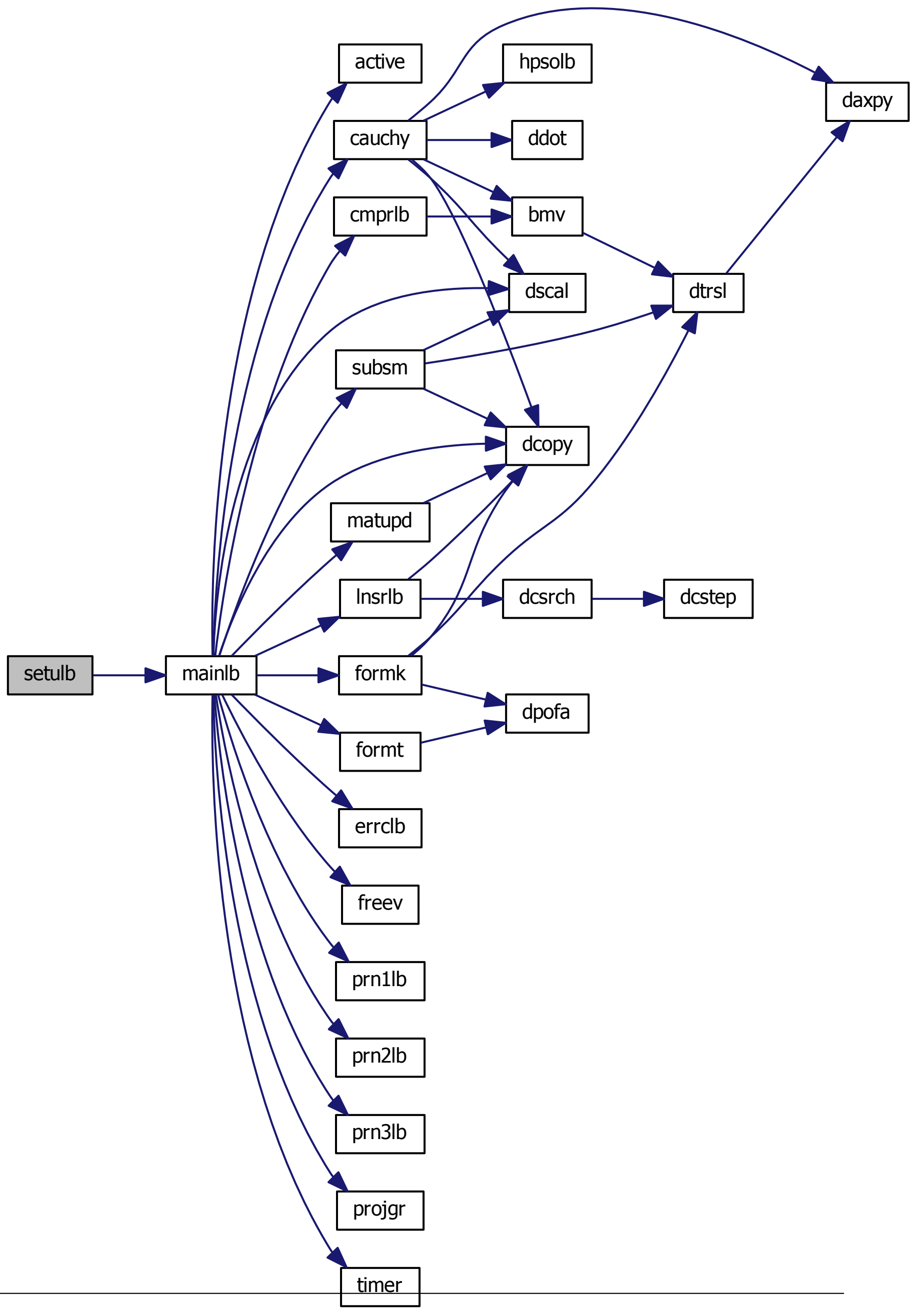
#### 6.19.1.16 subroutine prn2lb ( integer *n*, double precision,dimension(n) *x*, double precision *f*, double precision,dimension(n) *g*, integer *iprint*, integer *itfile*, integer *iter*, integer *nfgv*, integer *nact*, double precision *sbgnrm*, integer *nseg*, character∗3 *word*, integer *iword*, integer *iback*, double precision *stp*, double precision *xstep* )

#### 6.19.1.17 subroutine prn3lb ( integer *n*, double precision,dimension(n) *x*, double precision *f*, character∗60 *task*, integer *iprint*, integer *info*, integer *itfile*, integer *iter*, integer *nfgv*, integer *nintol*, integer *nskip*, integer *nact*, double precision *sbgnrm*, double precision *time*, integer *nseg*, character∗3 *word*, integer *iback*, double precision *stp*, double precision *xstep*, integer *k*, double precision *cachyt*, double precision *sbtime*, double precision *lnscht* )

#### 6.19.1.18 subroutine projgr ( integer *n*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision,dimension(n) *x*, double precision,dimension(n) *g*, double precision *sbgnrm* )

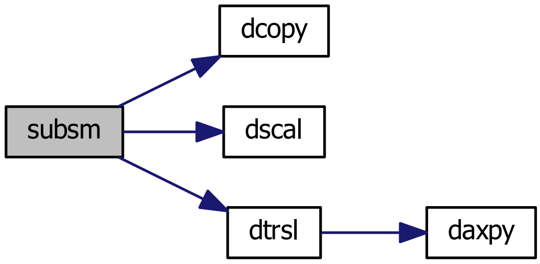
#### 6.19.1.19 subroutine setulb ( integer *n*, integer *m*, double precision,dimension(n) *x*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision *f*, double precision,dimension(n) *g*, double precision *factr*, double precision *pgtol*, *wa*, integer,dimension(3∗n) *iwa*, character∗60 *task*, integer *iprint*, character∗60 *csave*, logical,dimension(4) *lsave*, integer,dimension(44) *isave*, *dsave* )

Here is the call graph for this function:



#### 6.19.1.20 subroutine subsm ( integer *n*, integer *m*, integer *nsub*, integer,dimension(nsub) *ind*, double precision,dimension(n) *l*, double precision,dimension(n) *u*, integer,dimension(n) *nbd*, double precision,dimension(n) *x*, double precision,dimension(n) *d*, double precision,dimension(n) *xp*, double precision,dimension(n, m) *ws*, double precision,dimension(n, m) *wy*, double precision *theta*, double precision,dimension(n) *xx*, double precision,dimension(n) *gg*, integer *col*, integer *head*, integer *iword*, double precision,dimension(2∗m) *wv*, double precision,dimension(2∗m, 2∗m) *wn*, integer *iprint*, integer *info* )

Here is the call graph for this function:




## 6.20 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/lbfgsb/linpack.f File Reference

### Functions/Subroutines

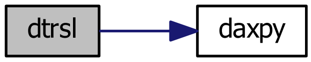
- subroutine dpofa (a, lda, n, info)
- subroutine dtrsl (t, ldt, n, b, job, info)

### 6.20.1 Function Documentation

#### 6.20.1.1 subroutine dpofa ( double precision,dimension(lda,∗) *a*, integer *lda*, integer *n*, integer *info* )

#### 6.20.1.2 subroutine dtrsl ( double precision,dimension(ldt,∗) *t*, integer *ldt*, integer *n*, double precision,dimension(∗) *b*, integer *job*, integer *info* )

Here is the call graph for this function:




## 6.21 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/lbfgsb/timer.f File Reference

### Functions/Subroutines

- subroutine timer (ttime)

### 6.21.1 Function Documentation

#### 6.21.1.1 subroutine timer ( double precision *ttime* )

## 6.22 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/mass_massq.f90 File Reference

### Modules

- module mass_massq

  *Module of $\mathbf{M_q V}$, which is the jacobian of the mass matrix multiplied by a vector. It's NOT a user module, it's used by the solver.*

### Functions/Subroutines

- subroutine mass_massq::dMdqV_Setup
- subroutine mass_massq::deallocdMdqV
- subroutine mass_massq::Eval_Mass_Matrix

  *Subroutine to assemble the mass matrix of the whole system.*
- subroutine mass_massq::Mq (body, lb)

  *Subroutine to evaluate $\mathbf{M}_q\mathbf{V}$ of one body.*

- subroutine mass_massq::Eval_Mq (lb)

  *Subroutine to assemble $\mathbf{M}_q\mathbf{V}$ of the whole system.*

### Variables

- REAL(8), dimension(:,:), allocatable mass_massq::PROTECTED
- REAL(8), dimension(:,:), allocatable mass_massq::Mqlb

## 6.23 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/math_oper.f90 File Reference

### Modules

- module math_oper

  *Module of non-intrinsic mathematical operations. Contains all the operations necessary for multi body dynamics computations not supported by the Fortran 2003 standard.*

### Functions/Subroutines

- REAL(8) math_oper::norm (r)

  *Calculate the standard $|\mathbf{r}|$ of a vector $\mathbf{r}$.*
- REAL(8), dimension(size(r)) math_oper::normalize (r)

  *Normalize an vector $\mathbf{normalize}(\mathbf{r}) = \frac{\mathbf{r}}{|\mathbf{r}|}$.*
- REAL(8), dimension(3) math_oper::pvect (u, v)

  *Calculates the cross product of two vectors.*
- subroutine math_oper::PerpVectors (nfl, u, v)

  *Given a vector, calculates two perpendicular vectors to the given one.*
- real(8) math_oper::MREulerParam (e)

  *Given the vector of Euler parameters $e$, it returns the rotation transformation matrix $A$.*
- real(8) math_oper::Gmatrix (e)

  *Given the vector of Euler parameters $e$, it returns the G matrix $G$.*
- real(8) math_oper::Gmatrix_e1 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_0}.*
- real(8) math_oper::Gmatrix_e2 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_1}.*
- real(8) math_oper::Gmatrix_e3 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_2}.*
- real(8) math_oper::Gmatrix_e4 (e)

  *Given the vector of Euler parameters $e$, it returns ={ A}{ e_3}.*
- real(8) math_oper::dMREulerParam_v (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\frac{\partial Av}{\partial e}$.*

- real(8) math_oper::dMREulerParam_t (p, pd, t)

  *Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}Av}{\mathrm{d}t}$.*
- real(8) math_oper::d2MREulerParam_t2 (p, pd, ps, t)

  *Given the vector of Euler parameters $e$, the velocity vector of Euler parameters $\dot{e}$ and a vector $v$, it returns $\frac{\mathrm{d}^2Av}{\mathrm{d}t^2}$.*
- real(8) math_oper::dMREulerParam_t_e1 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_0$.*
- real(8) math_oper::dMREulerParam_t_e2 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_1$.*
- real(8) math_oper::dMREulerParam_t_e3 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_2$.*
- real(8) math_oper::dMREulerParam_t_e4 (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial e_3$.*
- real(8) math_oper::dMREulerParam_t_e1d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_0$.*
- real(8) math_oper::dMREulerParam_t_e2d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_1$.*
- real(8) math_oper::dMREulerParam_t_e3d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_2$.*
- real(8) math_oper::dMREulerParam_t_e4d (p, pd, t)

  *Given the vector of Euler parameters, the velocity vector of Euler parameters and a vector, it returns $\partial \frac{\mathrm{d}Av}{\mathrm{d}t} \partial \dot{e}_3$.*
- real(8) math_oper::dMREulerParam_v_e1 (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_0$.*
- real(8) math_oper::dMREulerParam_v_e2 (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_1$.*
- real(8) math_oper::dMREulerParam_v_e3 (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_2$.*
- real(8) math_oper::dMREulerParam_v_e4 (e, v)

  *Given the vector of Euler parameters $e$ and a vecter $v$, it returns $\partial(\frac{\partial Av}{\partial e})\partial e_3$.*
- real(8) math_oper::MREulerParam_e1 (e)

  *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_0$.*
- real(8) math_oper::MREulerParam_e2 (e)

  *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_1$.*
- real(8) math_oper::MREulerParam_e3 (e)

*Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_2$.*

- real(8) math_oper::MREulerParam_e4 (e)

  *Given the vector of Euler parameters $e$, it returns the derivative of the rotation matrix with respect to $e_3$.*

- real(8), dimension(size(u), size(v)) math_oper::tensor_product (u, v)

  *Given the two vectors, find the tensor product between these two vectors.*

- real(8) math_oper::GAV (p, v)

  *Given the euler parameters $p$ and a vector $v$ of one body , find $2*G^T\tilde{A}v$.*

- real(8) math_oper::GAV_e1 (p, v)

  *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_0}$.*

- real(8) math_oper::GAV_e2 (p, v)

  *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_1}$.*

- real(8) math_oper::GAV_e3 (p, v)

  *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_2}$.*

- real(8) math_oper::GAV_e4 (p, v)

  *Given the euler parameters $p$ and a vector $v$ of one body , returns $\frac{d2*G^T\tilde{A}v}{de_3}$.*

### Variables

- REAL(8), dimension(3, 3), parameter math_oper::EYE3 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0, 0.d0,1.d0,0.d0, 0.d0,0.d0,1.d0/), SHAPE=(/3,3/))
- REAL(8), dimension(3, 3), parameter math_oper::ZEROS3 = 0.d0
- REAL(8), dimension(4, 4), parameter math_oper::EYE4 = RESHAPE(SOURCE=(/ 1.d0,0.d0,0.d0,0.d0, 0.d0,1.d0,0.d0,0.d0, 0.d0,0.d0,1.d0,0.d0, 0.d0,0.d0,0.d0,1.d0/), SHAPE=(/4,4/))

## 6.24 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/matlab_caller.f90 File Reference

### Modules

- module matlab_caller

  *Managment of sessions of MATLAB engine: This is a part of the matlab_caller module of MBSLIM.*

### Functions/Subroutines

- subroutine matlab_caller::MATLAB_OPENSES

  *OPENS MATLAB SESSION.*

- subroutine matlab_caller::MATLAB_CLOSESES

  *CLOSES MATLAB SESSION.*

- subroutine matlab_caller::MATLAB_CHECKSES

  *CHECKS IF THERE IS A MATLAB SESSION (for internal use of the module only)*
- subroutine matlab_caller::MATLAB_EVALSTRING (STRING)

  *It evaluates Matlab expression.*
- subroutine matlab_caller::MATLAB_PUTREALVECTOR (b, NOMBRE)

  *It passes a real vector b.*
- subroutine matlab_caller::MATLAB_GETREALVECTOR (b, NOMBRE)

  *It gets/reads the vector 'nombre' from matlab and it places it in b!>*
- subroutine matlab_caller::MATLAB_GETREAL (c, NOMBRE)

  *It reads the scalar NOMBRE from Matlab and it places it on variable c.*
- subroutine matlab_caller::MATLAB_PUTINTEGER (i, NOMBRE)

  *PASSES AN INTEGER i.*
- subroutine matlab_caller::MATLAB_PLOT (t_graph, y_graph, figur, linecolor, linewidth)

  *PLOTS 2 VECTORS OF REAL DATA Y.VS.X.*

## 6.25 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/primitive_forces.f90 File Reference

### Modules

- module primitive_forces

  *Primitive forces module.*

### Functions/Subroutines

- subroutine primitive_forces::TSDA (r1, r2, r1p, r2p, s0, k, c, F1, F2)

  *Function to get the primitive forces of a translational spring-damper-actuator between acting on two bodies.*
- subroutine primitive_forces::TSDA_q (r1, r2, r1p, r2p, s0, k, c, df1dr1, df1dr2, df2dr1, df2dr2)

  *Function to get the primitive stiffness of a translational spring-damper-actuator between acting on two bodies.*
- subroutine primitive_forces::TSDA_qp (r1, r2, c, df1dr1p, df1dr2p, df2dr1p, df2dr2p)

  *Function to get the primitive damping of a translational spring-damper-actuator between acting on two bodies.*

## 6.26 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/restric.f90 File Reference

## Modules

- module restric

    *Module of primitive constraints. It's NOT a user module, it's used by the solver.*

## Functions/Subroutines

- subroutine restric::restric_Setup
- subroutine restric::deallocfi
- subroutine restric::r_UnitEulParam (ir, iEul)

    *Primitive constraint of unitary Euler parameters assume* $\mathbf{p}$ *is the Euler parameter.the constraint equation is :* $\mathbf{p}^T\mathbf{p}-1$.
- subroutine restric::r_dot1GB (ir, iEul2, u, v)

    *Primitive dot-1 constraint of a body attached on the ground.*
- subroutine restric::r_dot1 (ir, iEul1, iEul2, u, v)

    *Primitive dot-1 constraint assume* $\mathbf{p_1}$ *and* $\mathbf{p_2}$ *are the Euler parameter of body 1 and body 2 and* $\mathbf{u}$ *and* $\mathbf{v}$ *are two vectors attached on body 1 and body 2 in the body reference frame, the constraint equation is :* $A(\mathbf{p_1})\mathbf{u}^T A(\mathbf{p_2})\mathbf{v}$.
- subroutine restric::r_sphericalGB (ir, irg2, iEul2, pt1, pt2)

    *Primitive constraints of a spherical joint of a body attached to the ground The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{'P}-\mathbf{s}_1^P$.
- subroutine restric::r_spherical (ir, irg1, irg2, iEul1, iEul2, pt1, pt2)

    *Primitive constraints of a spherical joint between two bodies The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{'P}-\mathbf{r}_1-\mathbf{A}_1\mathbf{s}_1^{'P}$.
- subroutine restric::r_revoluteGB (ir, irg2, iEul2, pt1, pt2, u1, v1, vec2)

    *Primitive constraints of a revolute joint of a body attached to the ground The first three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{'P}-\mathbf{s}_1^P$ *The fouth constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{h}_2$ *The fifth constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{h}_2$.
- subroutine restric::r_revolute (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, u1, v1, vec2)

    *Primitive constraints of a revolute joint between two bodies The three constraint equations are :* $\mathbf{r}_2+\mathbf{A}_2\mathbf{s}_2^{'P}-\mathbf{r}_1-\mathbf{A}_1\mathbf{s}_1^{'P}$ *The fouth constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The fifth constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$.
- subroutine restric::r_transGB (ir, irg2, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

    *Primitive constraints of a translational joint of a body attached to the ground The first constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{h}_2$ *The second constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{h}_2$ *The third constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{d}_{12}$ *The forth constraint equation is:* $\mathbf{g}_1^T\mathbf{A}_2\mathbf{d}_{12}$ *The fifth constraint equation is:* $\mathbf{f}_1^T\mathbf{A}_2\mathbf{f}_2$.
- subroutine restric::r_trans (ir, irg1, irg2, iEul1, iEul2, pt1, pt2, vec1y, vec1x, vec2x, vec2z)

    *Primitive constraints of a translational joint between two bodies. The first constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The second constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{h}_2$ *The third constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ *The forth constraint equation is:* $(\mathbf{A}_1\mathbf{g}_1)^T\mathbf{A}_2\mathbf{d}_{12}$ *The fifth constraint equation is:* $(\mathbf{A}_1\mathbf{f}_1)^T\mathbf{A}_2\mathbf{f}_2$.
- subroutine restric::r_Drive_rgEul (ir, ind, i_MOTOR)

    *Primitive driving constraints for a generalized coordinate of the system.*
- subroutine restric::r_Drive_distGB (ir, irg2, iEul2, pt1, pt2_loc, i_MOTOR)

*Primitive driving constraints for a distance between a point in the ground and a point of one body.*

- subroutine restric::r_Drive_dist (ir, irg1, irg2, iEul1, iEul2, pt1_loc, pt2_loc, i_-MOTOR)

  *Primitive driving constraints for a distance between two points of two bodies.*

### Variables

- REAL(8), dimension(:), allocatable restric::PROTECTED
- REAL(8), dimension(:), allocatable restric::fi

## 6.27 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/solids.f90 File Reference

### Modules

- module SOLIDS

  *Solids module that adds and manages the bodies of the system.*

### Functions/Subroutines

- subroutine, public SOLIDS::ADDbody (body, rg0, eu0)

  *Adds a body to the model.*
- subroutine, public SOLIDS::ADDmassgeom (body, mass, rg_loc, v_lg)

  *Subroutine to add mass, center of masses and inertia tensor with respect to the center of mass to a body.*
- REAL(8), public SOLIDS::MatTrans (body)

  *Function to calculate the transformation matrix of a body* $\mathbf{A}^* = \begin{bmatrix} \mathbf{R} & \mathbf{p}_0 \\ \mathbf{0} & 1 \end{bmatrix}$.
- REAL(8), dimension(3), public SOLIDS::r (body, pt_loc)

  *Function to evaluate the position of a point belonging to a body.*
- REAL(8), dimension(3), public SOLIDS::rp (body, pt_loc)

  *Function to evaluate the velocity of a point belonging to a body.*

### Variables

- TYPE(SOLID), dimension(:), allocatable, public SOLIDS::PROTECTED
- TYPE(SOLID), dimension(:), allocatable, public SOLIDS::SOLIDlist
- INTEGER, parameter, public SOLIDS::ground = 0
- INTEGER, public SOLIDS::nSOLID = 0

# 6.28 D:/Mis_Documentos/investigacion/proyectos/VT optimization project/MBSVT/trunk/state.F90 File Reference

## Modules

- module STATE

  *Module of solver state variables, subroutines and functions. It creates, manages and updates the state variables of the model.*

## Functions/Subroutines

- subroutine STATE::STATE_Setup
- subroutine STATE::setDOF (dof_in, zp_in, zs_in)

  *Define the degrees of freedom of the system and the speed of such degree of freedom.*
- REAL(8), dimension(3) STATE::rg (body)

  *Function to ask the coordinates of the CG of a body.*
- REAL(8), dimension(3) STATE::rgp (body)

  *Function to ask the velocity of the CG of a body.*
- REAL(8), dimension(3) STATE::rgs (body)

  *Function to ask the acceleration of the CG of a body.*
- REAL(8) STATE::rgx (body)

  *Function to ask the x-coordinate of the CG of a body.*
- REAL(8) STATE::rgy (body)

  *Function to ask the y-coordinate of the CG of a body.*
- REAL(8) STATE::rgz (body)

  *Function to ask the z-coordinate of the CG of a body.*
- REAL(8), dimension(4) STATE::Eul (body)

  *Function to ask the Euler parameters of a body.*
- REAL(8), dimension(4) STATE::Eulp (body)

  *Function to ask the first derivatives of the Euler parameters of a body.*
- REAL(8), dimension(4) STATE::Euls (body)

  *Function to ask the second derivatives of the Euler parameters of a body.*
- subroutine STATE::assembleMM (i, m, JJ)

  *Subrotine to assemble elemental mass matrices to the global one. It's NOT a user function, it's intended to be called by the solver.*

## Variables

- REAL(8), dimension(:), allocatable STATE::q
- REAL(8), dimension(:), allocatable STATE::qp
- REAL(8), dimension(:), allocatable STATE::qs
- REAL(8), dimension(:), allocatable STATE::qp_g
- REAL(8), dimension(:), allocatable STATE::qs_g

- REAL(8), dimension(:), allocatable STATE::zp
- REAL(8), dimension(:), allocatable STATE::zs
- REAL(8), dimension(:,:), allocatable STATE::MM
- REAL(8), dimension(:), allocatable STATE::lambda
- REAL(8), dimension(:), allocatable STATE::pos
- REAL(8), dimension(:), allocatable STATE::vel
- REAL(8), dimension(:), allocatable STATE::ace
- INTEGER, dimension(:), allocatable STATE::gdl