

SALADYN

Saladyn MultiBody Toolbox

ANR Project Saladyn

ANR-08-COSI-014
Liverable 2b Version 2

Vincent Acary, Olivier Bonnefon

Date : September 2011

Table of Contents

1 SALADYN Multibody toolbox task 2b - Project Overview.....	1
1.1 Release Information.....	1
1.2 Mission and Scope.....	1
1.2.1 What is the purpose and scope of this document ?.....	1
1.2.2 What is the scope of this project?.....	1
1.2.3 Goal of this document.....	1
1.3 Main features.....	1
1.4 Overview of existing open-source multibody toolbox.....	1
1.4.1 About MBDyn.....	1
1.5 Overview of existing collision detection libraries.....	2
1.5.1 Specification around the collision detection.....	2
1.5.1.1 SalaGeom.....	2
1.5.1.2 Builder of interactions.....	3
1.5.1.3 Simulation module.....	3
2 Software requirements and design. Tasks definition.....	5
2.1 Newton-Euler relations and dynamical systems.....	5
2.2 Projection on the constraints(Task Projection1).....	5
2.3 Numerical solver.....	6
2.3.1 The bilateral constraints(Task Numerical1).....	6
2.3.2 Adapt the MLCP solver(Task Numerical2).....	6
2.4 CAD analysis using OpenCascade(OCC).....	6
2.4.1 Initialisation of the data:.....	6
2.4.2 Geometrical analysis(Task OCC3).....	6
2.4.3 update the OCC model(Task OCC5).....	7
2.4.4 OpenCasCade(OCC) Application User Interface.....	7
2.4.5 Data flow.....	7
2.4.6 Visualisation (Task OCC4).....	7
2.5 Global architecture.....	7
2.5.1 The input module (Task INPUT1).....	8
3 Development plan.....	9
4 Future Tasks.....	11
4.1 To add joints in Siconos (JOINT_SICONOS1).....	11
4.2 to connect a joint from siconos to the MBTB (Task JOINT_MBTB).....	11
4.3 Schneider validation plan (Task UC42).....	11
4.4 Schneider integration (Task INTEGRATION1).....	11
4.5 To improve the projection algorithm (Task PROJ_IMPROVE1).....	11
4.6 Summary.....	11
5 Appendices.....	13
5.1 User Guide.....	13
5.2 Application Programming Interface of the 3D modeler (OCC) for the geometrical module of SALADYN.....	13

1 SALADYN Multibody toolbox task 2b - Project Overview

1.1 Release Information

Project:	SALADYN
Internal Release Number:	1.0
Last update:	September 01, 2009

1.2 Mission and Scope

1.2.1 What is the purpose and scope of this document ?

This document describes the features of the SALADYN Multibody toolbox. The major constraints of development and exploitation are delineated. It plans the software design and development. The Appendices of this document contains the user guide of the Multibody toolbox and the Application Programming Interface waiting from the 3D modeler(OCC) for the geometrical module of SALADYN. It concerns the users and the software framework builders.

1.2.2 What is the scope of this project?

This project is a part of the SALADYN ANR project. ANR-08-COSI-014 .

1.2.3 Goal of this document

The goal of the project is to answer to the task 2b described in the project proposal document SALADYN_B

Task 2b: Design of the multibody toolbox.

1.3 Main features

List of the main features of this module.

- Multibody mechanical systems with perfect joints.
- Unilateral constraints, impacts and Coulomb's friction.
- Collision detection.
- CAD files import and analysis.
- Salome integration.
- SALADYN toolbox integration. Model adaptability and interoperability.

1.4 Overview of existing open-source multibody toolbox

1.4.1 About MBDyn

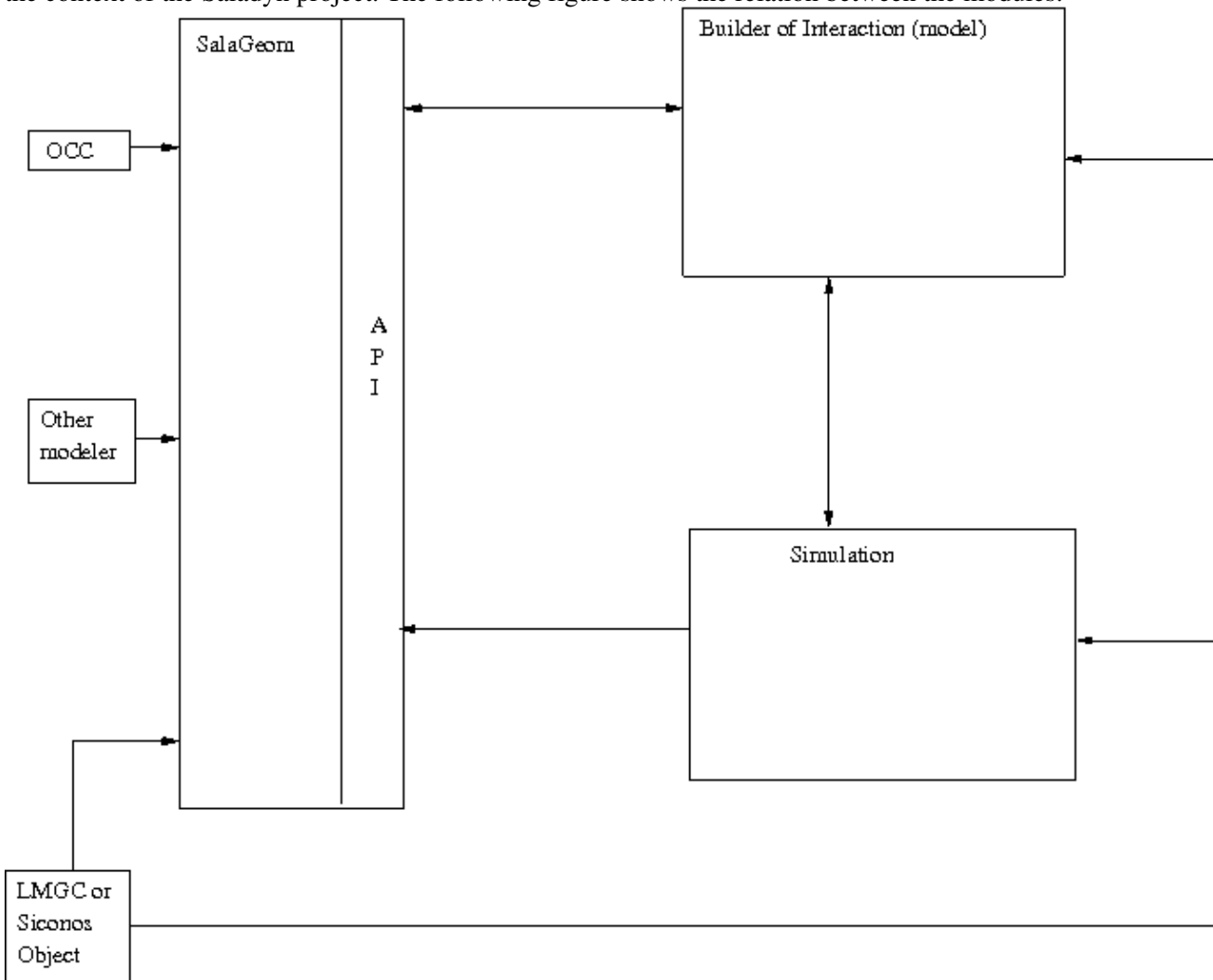
We were interested to couple Siconos to MbDyn, because MbDyn already provides many implemented bilateral constraints (joints). Nevertheless, this coupling task would need an important effort that we are not able to do, because of our own ressources. Moreover our application needs only simple bilateral constraints (Knee, Pivot and Prismatic), so we chose to do it by ourself to obtain a self-contained toolbox in a first step.

1.5 Overview of existing collision detection libraries

For standard multibody applications, the contact surfaces are known in advance. We only need to compute the gap functions between two bodies and gradients of these gap functions. These functionalities are based on the OCC methods. For the collision detection with other type of mechanical systems (FEM and granular materials), the collision detector will appear as a module of the Saladyn platform. It is planned to adapt the LMG algorithm (Task COLL1).

1.5.1 Specification around the collision detection

This paragraph comes from the meeting :[Discussion about collision detector at LMG](#). The goal of this part is to describe the software architecture of the geometrical toolbox. We zoom on the plug of the OCC object in the context of the Saladyn project. The following figure shows the relation between the modules:



1.5.1.1 SalaGeom

It is a library providing only the geometrical features useful to determinate the points of contact. This library hides the modeler libraries. The list of features is(must be completed and detailed): Features about loading:

- Add Object from a CAD file.
- Declare the LMG and/or Siconos objects.

Features useful for the global detector:

Saladyn Multibody toolbox

Parse the tree description of the world.

Get bounding box: computing the BB in the global frame.

Get Oriented bounding box: computing the oriented BB.

Features computing distances:

Compute the distance between simple objects. ((Face-Wire-Point)x(Face-Wire-Point)): It consists in finding the local minimal from a initial values of parameters.

Move an object (after simulation).

Build a graphical model.(optional).

The SalaGeom library assumes all wires and surfaces have a 1d or 2d parametrisation. LMGC and SICONOS have to provide it, except if the object is reduced to a point (case of sphere). It is noteworthy that this module doesn't contain the algorithm building the contacts points.

1.5.1.2 Builder of interactions

Based on the SalaGeom, this module (lib, or python ?) builds the interactions. For example, in the case of planar contact, started from a set of parameters value can lead to a convenient set of point of contacts.

1.5.1.3 Simulation module.

After the building of interactions, it is possible to run the simulation. Do not forget to update the positions in SalaGeom.

2 Software requirements and design. Tasks definition

2.1 Newton-Euler relations and dynamical systems.

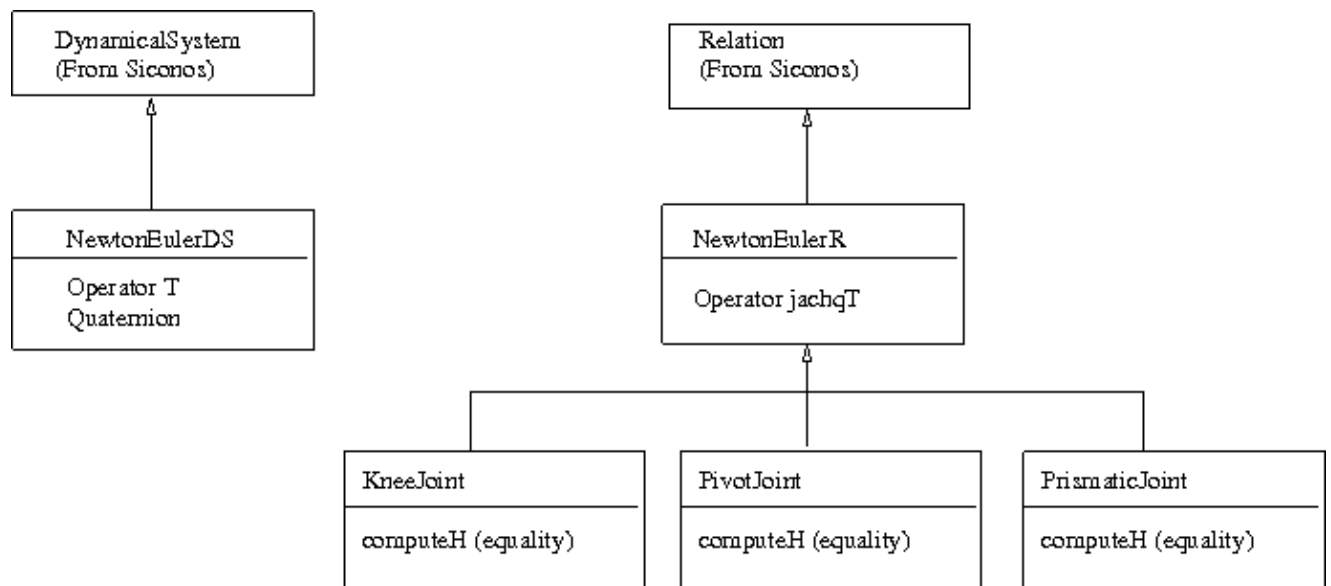
The multibody toolbox is based on Siconos. The holonomic constraints are defined using the standard Siconos relations. The unilateral constraints and the Coulomb friction is represented using specific nonsmooth laws. For more details, see the user documentation of Siconos.

Work to do in terms development: create basic joints in Siconos does not needs to much work, so it was chosen, in a first step, to develop a basic MultiBody toolbox to validate the time integration and the numerical solver.

Define a Newton-Euler representation of body in Siconos. (Task NE1)

Define relation to implement some basic joints. (Task NE2-3-4)

The following diagram shows the class hierarchy. The NewtonEulerDS adds the quaternion and the operator T to the dynamical system master class. The operator is defined by the parameterization of the finite rotations in 3D motion (see livrable L1a). The NewtonEulerR deals with the operator T for the relation. Each type of joints is inherited from the NewtonEulerR class and implements the bilateral constraints and its gradients. This equalities could be generated using a computer algebra system such as Maple or Mathematica.



2.2 Projection on the constraints(Task Projection1)

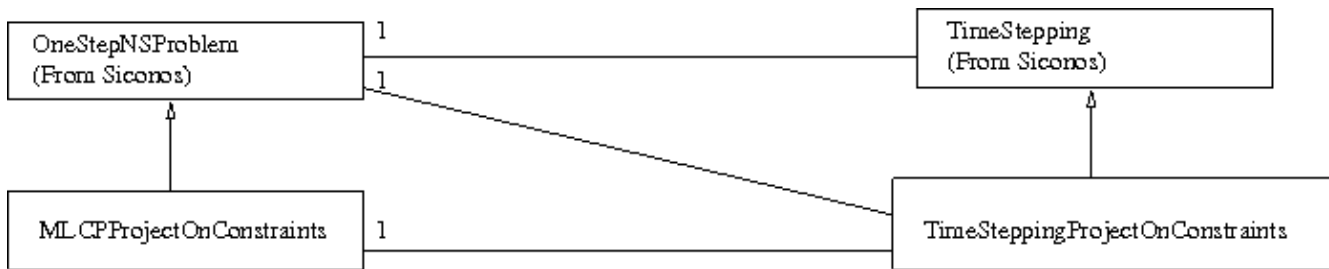
Because of the velocity formulation and its associated numerical drift, the bilateral and unilateral constraints are slight violated in discrete time. A correction is applied by adding a position formulation fo the constraints. The MLCPProjectOnConstraints class contains the formulation:

$$\begin{aligned}
 Y &= h(q) \\
 q_1 &= q_0 + jachq' \lambda \\
 h(q_1) &= h(q_0) + jachq (q_1 - q_0)
 \end{aligned}$$

The obtained MLCP is:

$$[H(q_0) + (jachq * jachq') \lambda] \in \{R, R+\}^*$$

$[\lambda] \in \{R, R+\}$



The class TimeSteppingProjectOnConstraints performs this formulation during the time integration.

2.3 Numerical solver.

2.3.1 The bilateral constraints(Task Numerical1)

The current version of numerical solver only deals with friction contacts and unilateral constraints. The bilateral constraints must be added. Mainly it consists in adding the so-called GeneralMechanicalProblem including also equalities. A dedicated Gauss-Seidel algorithm will be developed in Siconos/Numerics to solve this new problem.

2.3.2 Adapt the MLCP solver(Task Numerical2)

It consists in adapting the current algorithm to manage the case where the equalities are mixed with the inequalities. The current version of Siconos/Numerics assumes that they are sorted and not mixed.

2.4 CAD analysis using OpenCascade(OCC)

This part lists the functionalities of OpenCascade that we need. We do not focus on the collision manager, but on getting the geometrical informations like, frames, axis, inertial matrix and local frame at contact.

2.4.1 Initialisation of the data:

We need to get the OCC Object (TopoDS_shape) that represents the node or the dynamical system simulated (Task OCC2). Then, we get some geometrical informations like the position, the mass, the inertial matrix and many other information to build our system.

2.4.2 Geometrical analysis(Task OCC3)

During the simulation, the SALADYN platform makes some request to OCC to get geometrical informations. From the couple of nearest points, a numerical analysis must be done to build the jacqH operator (Task OCC8).

2.4.3 update the OCC model(Task OCC5)

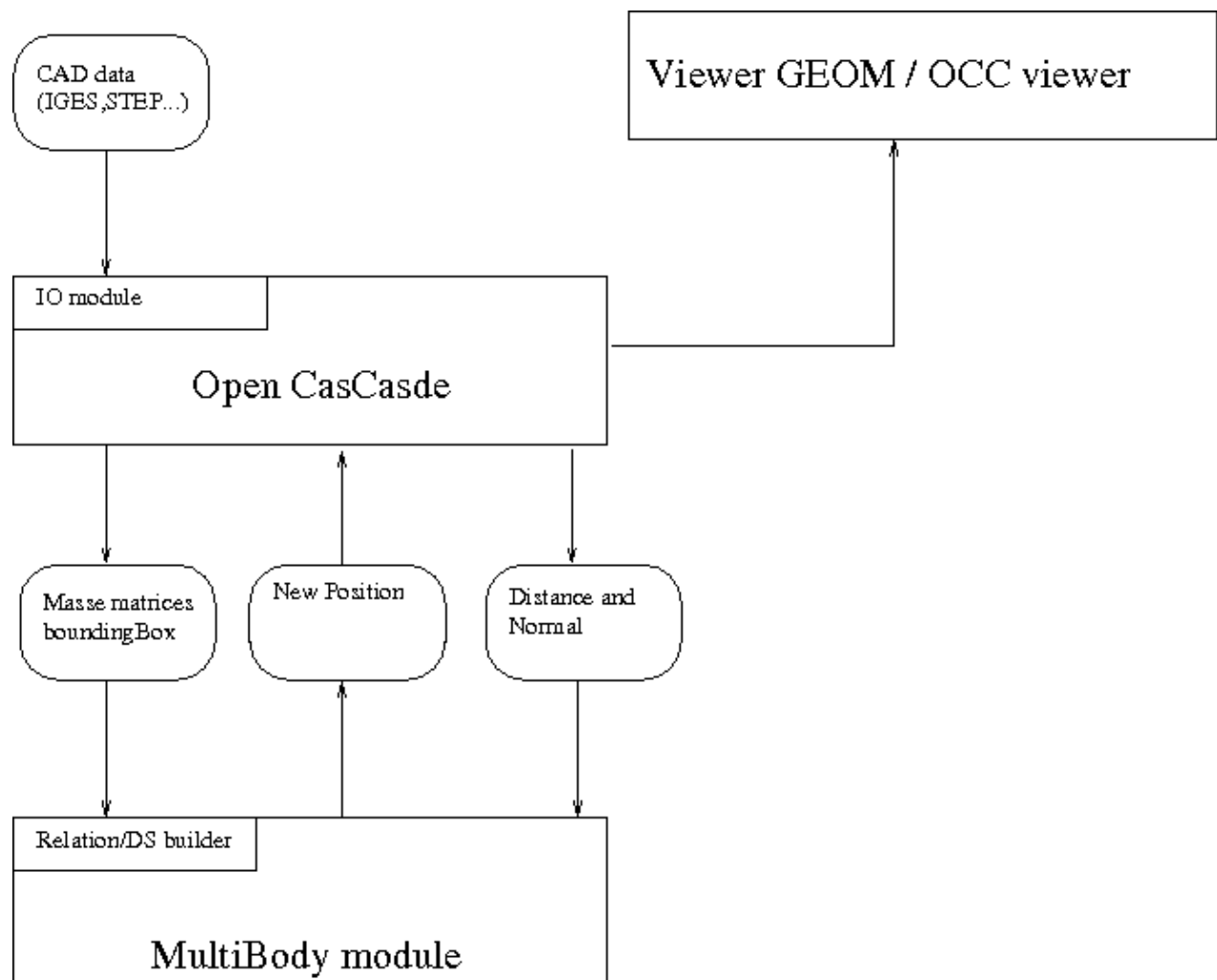
After each time-step of simulations, the positions in the OCC model must be updated. This is the goal of this task to do it efficiently.

2.4.4 OpenCasCade(OCC) Application User Interface

The document [Application user interface of OCC](#) (see also in Appendix) contains the API waited from OCC. Several tests have shown that OCC's algorithms to compute the nearest points between two surfaces is not efficient. It is planned to implement a new one based on the quasi-Newton methods. (Task OCC6-7)

2.4.5 Data flow

The following diagram shows the data exchanged between OpenCasCade and the multibody toolbox.



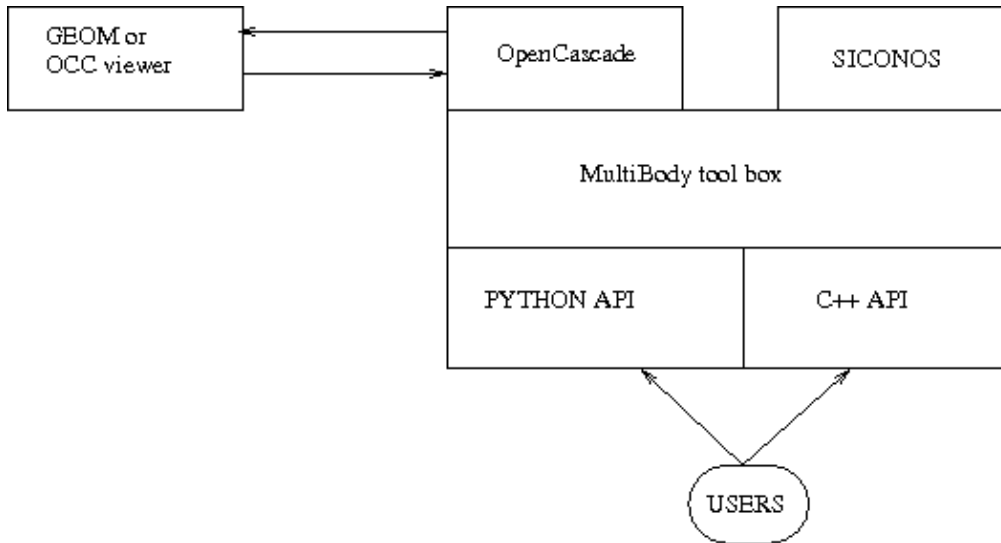
2.4.6 Visualisation (Task OCC4)

Visualisation is ensured either by Geom or an OCC viewer.

2.5 Global architecture

This section shows the relation between the module previously described. The integration to the SALADYN toolbox and then to the Salome platform will be done in Python.

Saladyn Multibody toolbox



2.5.1 The input module (Task INPUT1)

A task consists in writing an input module building the multi-body system from a XML file. This input file will contain the following fields:

Rigid bodies of the system.

- ◆ Id
- ◆ Name
- ◆ CAD model
- ◆ Mass and matrices
- ◆ Initial positions
- ◆ Forces and momentum plugin

Joints

- ◆ Id
- ◆ Name
- ◆ Type (Knee, prismatic ...)
- ◆ Position
- ◆ Id1 of Rigid body
- ◆ Id2 of Rigid body

Unilateral constraints and friction.

- ◆ Id
- ◆ Name
- ◆ Type (Friction or not)
- ◆ en
- ◆ et
- ◆ mu
- ◆ Id1 of Rigid body
- ◆ Id2 of Rigid body
- ◆ CAD1 model
- ◆ CAD1 model

Moreover the corresponding API will be available in C++ and in Python (Task INPUT2).

3 Development plan

The following table displays the time of development needed for each task.

Tasks Id	Tasks Summary	Estim. Days	% done 09/2010	% done 08/2011	Remark
OCC1	Training	5	100	100	Done with OCC staff
OCC2	IO	2	100	100	
OCC3	get Dist. and Normal	2	100	100	OCC algo is wrong
OCC4	Graphical model	2	100	100	Not stable
OCC5	Move an Object	3	80	100	
OCC6	Evaluation	3	100	100	
OCC7	improve dist algo	7	0	100	using optimisation algorithm
OCC8	Build operator jacqh	5	20	100	
Numerical1	add a GenericMechanicalProblem	8	70	100	
Numerical2	Adapt MLCP solver (mixed problem)	4	100	100	
Projection1	projection on constraints	7	80	100	
NE1	add NewtonEuler Classes	5	80	100	
NE2	KneeJoint	2	100	100	
NE3	PivotJoint	2	100	100	
NE4	PrismaticJoint	2	100	100	
INPUT1	Input module	10	0	100	with python
INPUT2	API	5	0	100	MBTB_PYTHON_API
COLL1	Collision	15	0	0	

All this development can not be done without meeting and coordination with the other partners. For example, in view of the use case UC-62, many meetings have been organised. At present time, the multi-body toolbox is able to simulate the circuit-breaker from an initial position allowing for external forces. But some improvements must be done, mainly around the geometrical aspect (Task OCC7) and also in view of a better integration in the SALADYN platform.

4 Future Tasks

4.1 To add joints in Siconos (JOINT_SICONOS1)

The current version of siconos provides the rotate and prismatic joints. To add a new joint in siconos you have to:

- Chose a convenient function $h(q)=0$ and its gradients. (using maple)
- Add the corresponding NewtonEuler relation.
- Add tests.

An approximated development time is 4 days per joint.

4.2 to connect a joint from siconos to the MBTB (Task JOINT_MBTB)

it consists in plugging a joint (pivot, prismatic...) from Siconos in the MBTB.

4.3 Schneider validation plan (Task UC42)

A validation plan has been created with Schneider (X. Herreros). It is the validation of the UC42. The goal is to validation the MBTB simulation on typical cases. This work is in course.

4.4 Schneider integration (Task INTEGRATION1)

This work consists in doing the software adaptation for Schneider. It is an important point, raising some questions about the software hot-line. Who will ensure the future evolution of this software ?

4.5 To improve the projection algorithm (Task PROJ_IMPROVE1)

This work is in course, it consists in improving the projection algorithm. It consist in writting a paper improving the projection algorithm shown in this document. Moreover, it can be useful to implemente the resulting numerical method.

4.6 Summury

The following table lists the future tasks.

Tasks Id	Tasks Summury	Estim. Days
COLL1	Collision	15
JOINT_SICONOS1	To add a new joint in Siconos	4 per joint
JOINT_MBTB	To plug a joint in MBTB	1
UC42	Schneider validation plan	5
INTEGRATION1	Schneider integration	?
PROJ_IMPROVE1	To improve the projection algorithm	?

5 Appendices

The appendices contains the user guide and the API of the 3D modeler.

5.1 User Guide

Please, see the attached document.

5.2 Application Programming Interface of the 3D modeler (OCC) for the geometrical module of SALADYN

Please, see the attached document.

SALADYN MultiBody ToolBox
1.0

User Guide

Wed Aug 24 14:34:25 2011

Contents

1	MultiBody Tool Box Documentation	1
1.1	Introduction	1
1.2	Installation	1
1.2.1	Build and compilation	1
1.2.2	Tools required	2
1.2.3	Tools optional	2
1.2.4	Intallation and running tests	2
2	Module Index	3
2.1	Modules	3
3	Class Index	5
3.1	Class List	5
4	Module Documentation	7
4.1	MBTB_DATA	7
4.1.1	Detailed Description	9
4.2	MBTB_INTERNAL_TOOL	10
4.2.1	Detailed Description	10
4.2.2	Function Documentation	10
4.2.2.1	_MBTB_STEP	10
4.2.2.2	_MBTB_updateContactFromDS	11
4.3	MBTB_PYTHON_API	12
4.3.1	Detailed Description	14
4.3.2	Enumeration Type Documentation	14
4.3.2.1	JOINTS_TYPE	14
4.3.2.2	MBTB_CST	14
4.3.3	Function Documentation	14
4.3.3.1	MBTB_BodyBuild	14

4.3.3.2	MBTB_BodyLoadCADFile	15
4.3.3.3	MBTB_BodySetVelocity	15
4.3.3.4	MBTB_ContactBuild	15
4.3.3.5	MBTB_ContactLoadCADFile	16
4.3.3.6	MBTB_ContactSetDParam	16
4.3.3.7	MBTB_ContactSetIParam	17
4.3.3.8	MBTB_doOnlyProj	17
4.3.3.9	MBTB_doProj	17
4.3.3.10	MBTB_init	18
4.3.3.11	MBTB_initSimu	18
4.3.3.12	MBTB_JointBuild	18
4.3.3.13	MBTB_moveBodyToPosWithSpeed	19
4.3.3.14	MBTB_run	19
4.3.3.15	MBTB_setGraphicFreq	20
4.3.3.16	MBTB_setJointPoints	20
4.3.3.17	MBTB_setSolverDOption	20
4.3.3.18	MBTB_setSolverIOption	20
4.3.3.19	MBTB_step	21
4.3.3.20	MBTB_updateDSFromSiconos	21
4.4	CADMBTB_API	22
4.4.1	Detailed Description	23
4.4.2	Function Documentation	23
4.4.2.1	CADMBTB_buildCylinderArtefactLine	23
4.4.2.2	CADMBTB_buildLineArtefactLine	23
4.4.2.3	CADMBTB_buildOrientedLineArtefactLine	24
4.4.2.4	CADMBTB_computeUVBounds	24
4.4.2.5	CADMBTB_getMinDistance	24
4.4.2.6	CADMBTB_getUVBounds	25
4.4.2.7	CADMBTB_getUVBounds2	25
4.4.2.8	CADMBTB_init	26
4.4.2.9	CADMBTB_initContact	26
4.4.2.10	CADMBTB_moveGraphicalModelFromModel	26
4.4.2.11	CADMBTB_moveModelFromModel	27
4.4.2.12	CADMBTB_reset	27
4.4.2.13	CADMBTB_setLocation	28
4.4.2.14	CADMBTB_setNbOfArtefacts	28

4.4.2.15	CADMBTB_updateGraphic	28
4.5	CADMBTB_DATA	29
4.5.1	Detailed Description	31
4.5.2	Variable Documentation	31
4.5.2.1	sTopoDS	31
4.6	CADMBTB_INTERNALTOOLS	32
4.6.1	Detailed Description	32
4.6.2	Function Documentation	32
4.6.2.1	_CADMBTB_getMinDistanceFaceEdge_using_n2qn1	32
4.6.2.2	_CADMBTB_getMinDistanceFaceFace	33
4.6.2.3	_CADMBTB_getMinDistanceFaceFace	34
4.6.2.4	_CADMBTB_getMinDistanceFaceFace_using_n2qn1	34
4.7	CADMBTB_PYTHON_API	36
4.7.1	Detailed Description	36
4.7.2	Function Documentation	36
4.7.2.1	CADMBTB_loadArtefactCADFile	37
4.7.2.2	CADMBTB_setContactDParam	37
4.7.2.3	CADMBTB_setIParam	37
4.7.2.4	CADMBTB_setShapeDParam	37
4.8	SliderCrankexample	39
4.8.1	Detailed Description	41
4.8.2	Variable Documentation	41
4.8.2.1	afile	41
4.8.2.2	afileContact1	41
4.8.2.3	afileContact2	42
4.8.2.4	BATIE	42
4.8.2.5	contactBody1	42
4.8.2.6	contactBody2	42
4.8.2.7	contacten	42
4.8.2.8	contactmu	43
4.8.2.9	contactName	43
4.8.2.10	contactNormalFromFace1	43
4.8.2.11	contactOffset	43
4.8.2.12	contactOffsetP1	43
4.8.2.13	contactType3D	44
4.8.2.14	fctf	44

4.8.2.15	fctm	44
4.8.2.16	inertialMatrix	44
4.8.2.17	initCenterMass	44
4.8.2.18	initPos	45
4.8.2.19	initVel	45
4.8.2.20	jointBody1	45
4.8.2.21	jointBody2	45
4.8.2.22	jointName	45
4.8.2.23	jointPos	46
4.8.2.24	jointType	46
4.8.2.25	m	46
4.8.2.26	PART1	46
4.8.2.27	PART2	46
4.8.2.28	PISTON	46
4.8.2.29	plugin	46
4.9	MbtbLocalOption	47
4.9.1	Detailed Description	48
4.9.2	Variable Documentation	48
4.9.2.1	Artefactfile	48
4.9.2.2	ArtefactTrans	48
4.9.2.3	bodyDraw	48
4.9.2.4	bodyTrans	49
4.9.2.5	contactDraw1	49
4.9.2.6	contactDraw2	49
4.9.2.7	contactTrans1	49
4.9.2.8	contactTrans2	49
4.9.2.9	freqUpdate	50
4.9.2.10	stepNumber	50
4.9.2.11	stepSize	50
4.9.2.12	with3D	50
4.9.2.13	withProj	50
5	Class Documentation	51
5.1	MBTB_Body Class Reference	51
5.1.1	Detailed Description	51
5.1.2	Constructor & Destructor Documentation	51
5.1.2.1	MBTB_Body	52

5.2	MBTB_Contact Class Reference	53
5.2.1	Detailed Description	54
5.2.2	Constructor & Destructor Documentation	54
5.2.2.1	MBTB_Contact	54
5.2.3	Member Data Documentation	55
5.2.3.1	_curTimeh	55
5.2.3.2	_Relation	55
5.3	MBTB_ContactRelation Class Reference	56
5.3.1	Detailed Description	56
5.4	MBTB_FC3DContactRelation Class Reference	57
5.4.1	Detailed Description	57
5.5	MBTB_JointR Class Reference	58
5.5.1	Detailed Description	58
5.6	MBTB_TimeStepping Class Reference	59
5.6.1	Detailed Description	59
5.6.2	Member Function Documentation	59
5.6.2.1	updateWorldFromDS	59
5.7	MBTB_TimeSteppingProj Class Reference	61
5.7.1	Detailed Description	61
5.7.2	Member Function Documentation	61
5.7.2.1	updateWorldFromDS	61

Chapter 1

MultiBody Tool Box Documentation

1.1 Introduction

This is the MultiBody Tool Box documentation of the Saladyn project. This tool box is composed of the libraries CADMBTB and MBTB.

- CADMBTB provides the geometrical features needed to perform the multi bodies system simulation. In view of a Salome integration, the CADMBTB is based on the OpenCascade Library.
- MBTB provides an Application Programming Interface to define and simulate a multi bodies system. These modules have a python API generated by SWIG from the files [MBTB_PYTHON_API.hpp](#) and [CADMBTB_PYTHON_API.hpp](#) .

1.2 Installation

1.2.1 Build and compilation

To compil and build the python modules, run 'cmake SALADYN_DIR/trunk/Multibody/src'

It will build the following directories:

- SALADYN_BUILD/MBTB
- SALADYN_BUILD/MBTB/CADMBTB : containing libCADMBTB.so
- SALADYN_BUILD/MBTB/CADMBTB : containing libTIMERMBTB.so
- SALADYN_BUILD/MBTB/MBTB : containing libMBTB.so
- SALADYN_BUILD/MBTB/plugin : containing libplugin1.so

- SALADYN_BUILD/MBTB/frontEnd
- SALADYN_BUILD/MBTB/frontEnd/CADMBTB : containing the python module cadmbtb
- SALADYN_BUILD/MBTB/frontEnd/MBTB : containing the python module mbtb
- SALADYN_BUILD/MBTB/TIMERMBTB : An API on a simple timer profiling library.

1.2.2 Tools required

- Siconos
- OpenCascade

1.2.3 Tools optional

- pythonOCC, only for the 3D visualisation.

1.2.4 Intallation and running tests

The directory SALADYN_DIR/trunk/Multibody/Tests contains some tests about the multiBodies tool box. Each example directory contains a bodydef.py describing the multibody.

It assumes that environment variable SALADYN_BUILD and SALADYN_DIR are defined.

SALADYN_BUILD is the directory containing the directory MBTB used for the compilation.

SALADYN_DIR is the main directory of saladyn.

To run an example:

- Go to the directory of the example.
- Copy the file SALADYN_DIR/trunk/Multibody/Tests/E1/env.sh, and make the necessary adaptations.
- source env.sh
- python ./run.py

The python files involved during the simulation are:

- bodydef.py, this file is required, it contains the multi bodies system. It is the input file. See the SliderCrank example module for a complete documentaion.
- mbtbLocalOptions.py, this file is optional, it is located in the example directory. It is used to defined the local options. See the mbtbLocalOptions module for a complete documentaion.
- Tests/mbtbDefaultOptions.py, this file must not be modified by the user, it contains the default options.
- Tests/run.py, this file must not be modified by the user, like said previously, it runs the simulation.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

MBTB_DATA	7
MBTB_INTERNAL_TOOL	10
MBTB_PYTHON_API	12
CADMBTB_API	22
CADMBTB_DATA	29
CADMBTB_INTERNALTOOLS	32
CADMBTB_PYTHON_API	36
SliderCrankexample	39
MbtbLocalOption	47

Chapter 3

Class Index

3.1 Class List

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

MBTB_Body (This class implements a body in a multi-bodies system. It inherits from Siconos::NewtonEulerDS)	51
MBTB_Contact (A Contact class)	53
MBTB_ContactRelation (It is a relation dedicated for the simple unilateral (ie: without Coulomb friction))	56
MBTB_FC3DContactRelation (It is a relation dedicated for the unilateral constraint with Coulomb friction)	57
MBTB_JointR (This class implements a joint in a multi-bodies system. It is an aggregation to the class siconos::NewtonEulerR. Mainly, it consists in adding members needed for the computation of joint forces)	58
MBTB_TimeStepping (This class implements the time stepping of a multi-bodies system. It inherits from Siconos::TimeStepping. It consists in update the CAD word during the simulation)	59
MBTB_TimeSteppingProj (This class implements the time stepping with projection of a multi-bodies system. It inherits from Siconos::TimeSteppingProjectOnConstraints. It consists in update the CAD word during the simulation)	61

Chapter 4

Module Documentation

4.1 MBTB_DATA

This file contains the static memory of the MBTB module.

Defines

- #define `FREQ_UPDATE_GRAPHIC` 1
Must be 1. It is the update frequency setting the transformation of the graphical object.
- #define `MBTB_MAX_BODIES_NUMBER` 30
The maximal number of bodies.
- #define `MBTB_MAX_JOINTS_NUMBER` 30
The maximal number of joints.
- #define `MBTB_MAX_CONTACTS_NUMBER` 30
The maximal number of contacts.

Variables

- `SP::MBTB_Body sDS` []
The dynamical bodies.
- `MBTB_JointR * sJointRelations` []
The joint relations.
- `MBTB_Contact * sContacts` []
The contacts.
- unsigned int `sNbOfBodies`

The number of bodies.

- unsigned int [sNbOfJoints](#)
The number of joints.
- unsigned int [sNbOfContacts](#)
The number of contacts.
- unsigned int [sTimerCmp](#)
The counter of step of simulation.
- unsigned int [sFreqGraphic](#)
The graphical frequency.
- DynamicalSystemsSet [sAllDS](#)
Dynamical set for siconos.
- DynamicalSystemsSet [sAllDSByInter](#) []
Dynamical set for siconos.
- SP::Interaction [sInterJoints](#) []
The siconos joint interactions.
- SP::Interaction [sInterContacts](#) []
The siconos contact interactions.
- InteractionsSet [sAllInteractions](#)
siconos set of interation.
- SP::Model [myModel](#)
siconos model.
- int [sJointIndexDS](#) []
for the graph building.
- int [sJointType](#) []
The type of joint see JOINTS_TYPE.
- SP::TimeStepping [sSimu](#)
The siconos simulation.
- unsigned int [sDrawMode](#)
The draw mode of the artefacts (forces, normals). Used with bit to bit test with MBTB_CST.
- double [sArtefactLength](#)
The nominal length of an artefact.
- double [sArtefactThershold](#)
The minimal length drawing.

- double `sNominalForce`

The nominal forces.

4.1.1 Detailed Description

This file contains the static memory of the MBTB module. The memory allocation is done using `MBTB_MAX_BODIES_NUMBER`, `MBTB_MAX_JOINTS_NUMBER` and `MBTB_MAX_CONTACTS_NUMBER`.

4.2 MBTB_INTERNAL_TOOL

This file contains the internal tools of the MBTB.

Functions

- void [_MBTB_updateContactFromDS](#) ()
It updates the contacts CAD model from the body.
- void [_MBTB_updateContactFromDS](#) (int numDS)
It updates the contacts CAD model from the body.
- void [_MBTB_printHeader](#) (FILE *fp)
It prints the header of the output file.
- void [_MBTB_printStep](#) (FILE *fp)
It prints the current state in the output file.
- void [_MBTB_STEP](#) ()
It performs a step.

4.2.1 Detailed Description

This file contains the internal tools of the MBTB. It consists in updating the CADMBTB from the simulation step.

It also manages the output data.

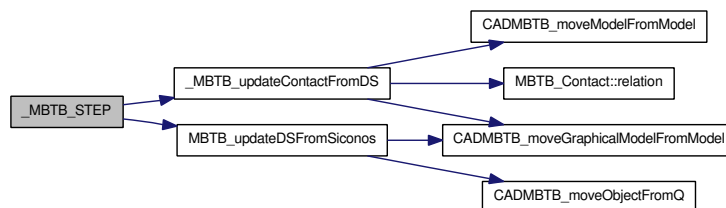
4.2.2 Function Documentation

4.2.2.1 void _MBTB_STEP ()

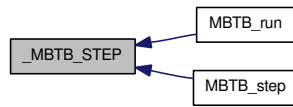
It performs a step.

It performs a step including the siconos call and the graphical update.

Here is the call graph for this function:



Here is the caller graph for this function:



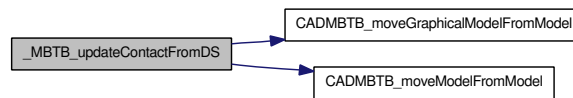
4.2.2.2 void _MBTB_updateContactFromDS (int numDS)

It updates the contacts CAD model from the body.

Parameters

← *int* numDS, update the cad model of contact related to the ds of id numDS.

Here is the call graph for this function:



4.3 MBTB_PYTHON_API

This file provides the python API of the MBTB module.

Enumerations

- enum `JOINTS_TYPE` { `PIVOT_0` = 0, `PIVOT_1` = 1, `PRISMATIC_0` = 2, `PRISMATIC_1` = 3 }

The joint type.

- enum `MBTB_CST` { `MBTB_ARTEFACT_P1P2` = 1, `MBTB_ARTEFACT_REACTION` = 2, `MBTB_ARTEFACT_NORMAL` = 4 }

Artefact constants.

Functions

- void `MBTB_init` (unsigned int NumOfBodies, unsigned int NumOfJoints, unsigned int NumberOfContacts)

To initialize the MBTB library. It calls the `CADMBTB_init`.

- void `MBTB_updateDSFromSiconos` ()

updating the CADs model

- void `MBTB_BodyLoadCADFile` (unsigned int numDS, const std::string &CADFile, unsigned int withGraphicModel)

Load the cad model of a body.

- void `MBTB_BodyBuild` (unsigned int numDS, const std::string &BodyName, double mass, SP::SimpleVector initPos, SP::SimpleVector initCenterMass, SP::SimpleMatrix inertialMatrix, const std::string &pluginFLib, const std::string &pluginFFct, const std::string &pluginMLib, const std::string &pluginMFct)

Build the MBTB_body and set to the initial position.

- void `MBTB_JointBuild` (unsigned int numJ, const std::string &JointName, unsigned int jointType, unsigned int indexDS1, unsigned int indexDS2, SP::SimpleVector jointPosition)

To build a joint.

- void `MBTB_setJointPoints` (unsigned int numJ, SP::SimpleVector G0C1, SP::SimpleVector G0C2)

To set the location where is computed the equivalente forces.

- void `MBTB_ContactLoadCADFile` (unsigned int contactId, const std::string &CADFile1, const std::string &CADFile2, unsigned int withGraphicModel1, unsigned int withGraphicModel2)

MBTB_ContactLoadCADFile load the cad model of a contact.

- void `MBTB_ContactSetDParam` (unsigned int paramId, unsigned int contactId, unsigned int idShape, double v)

To set a double parameter.(extendable, without modify the API).

- void `MBTB_ContactSetIParam` (unsigned int paramId, unsigned int contactId, unsigned int idShape, int v)

To set a interger parameter.(extendable, without modify the API).

- void `MBTB_ContactBuild` (unsigned int numContact, const std::string &ContactName, unsigned int indexBody1, int indexBody2, unsigned int withFriction, double mu, double en, double et)

To build a contact.

- void `MBTB_initSimu` (double hTS, int withProj)

It initializes the simulation.

- void `MBTB_run` (int nbSteps)

It runs the simulation.

- void `MBTB_step` ()

It does one step.

- void `MBTB_moveBodyToPosWithSpeed` (unsigned int numDS, SP::SimpleVector aPos, SP::SimpleVector aVel)

It is a warm start.

- void `MBTB_BodySetVelocity` (unsigned int numDS, SP::SimpleVector aVel)

It sets the velocity.

- void `MBTB_setGraphicFreq` (unsigned int freq)

It defines the graphic frequency.

- void `MBTB_setSolverIOption` (int i, int value)

It sets an integer value of the solver's parameters.

- void `MBTB_setSolverDOption` (int i, double value)

It sets a double value of the solver's parameters.

- void `MBTB_doProj` (unsigned int v)

It allows to enable/disable the projection algorithm.

- void `MBTB_doOnlyProj` (unsigned int v)

It allows to perform only the projection algorithm.

- void `MBTB_BodySetDParam` (unsigned int paramId, unsigned int bodyId, double v)

MBTB_BodySetDParam not yet used.

- void `MBTB_BodySetIParam` (unsigned int paramId, unsigned int bodyId, int v)

MBTB_BodySetDParam not yet used.

4.3.1 Detailed Description

This file provides the python API of the MBTB module. It provides :

- the functions to build the multi bobies system
- the functions to set internal parameters
- the functions to run the simulation.

4.3.2 Enumeration Type Documentation

4.3.2.1 enum JOINTS_TYPE

The joint type.

PIVOT_0, involves one ds. PIVOT_1, involves two ds. PRISMATIC, not used, the connection to siconos is not done.

4.3.2.2 enum MBTB_CST

Artefact constants.

Use with bit to bit test.

4.3.3 Function Documentation

4.3.3.1 void MBTB_BodyBuild (unsigned int numDS, const std::string & BodyName, double mass, SP::SimpleVector initPos, SP::SimpleVector initCenterMass, SP::SimpleMatrix inertialMatrix, const std::string & pluginFLib, const std::string & pluginFFct, const std::string & pluginMLib, const std::string & pluginMFct)

Build the MBTB_body and set to the initial postion.

This function build a mechanical body in the simulator.

Parameters

- ← *unsigned* int numDS, an identifier of body.
- ← *const* std::string& BodyName, a string for the body name.
- ← *double* mass, the mass.
- ← *SP::SimpleVector* initPos, a R^7 vector representing the position (translation in R^3 , vector in R^3 , angle in R) that must be appyed after the load to get the initial position of the object.
- ← *SP::SimpleVector* initCenterMass, coordinate of the mass center in the just loaded model
- ← *SP::SimpleMatrix* inertialMatrix, matrix in $R^{\{3,3\}}$
- ← *const* std::string& pluginLib, the path to the plugin library.

← *const* std::string& pluginFct, the name of the plugged fonction.

4.3.3.2 void MBTB_BodyLoadCADFile (unsigned int numDS, const std::string & CADFile, unsigned int withGraphicModel)

Load the cad model of a body.

Load the cad model of body numDS

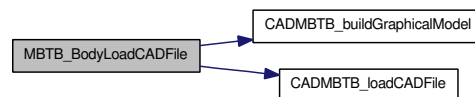
Parameters

← *unsigned* int numDS, identifier of Body.

← *const* std::string& CADFile, the cad file.

← *unsigned* int withGraphicModel, iff 0 the graphic model is not built

Here is the call graph for this function:



4.3.3.3 void MBTB_BodySetVelocity (unsigned int numDS, SP::SimpleVector aVel)

It sets the velocity.

It sets the siconos state.

Parameters

← *unsigned* int numDS, the id of the ds.

← *SP::SimpleVector* aVel, the target velocity.

4.3.3.4 void MBTB_ContactBuild (unsigned int numContact, const std::string & ContactName, unsigned int indexBody1, int indexBody2, unsigned int withFriction, double mu, double en, double et)

To build a contact.

It builds a relation of the convenient type doing the connection between the CAD model and the simulator.

Parameters

← *unsigned* int numContact, the id os the contact.

← *const* std::string& ContactName, the name of the contact.

← *unsigned* int indexBody1, the id of the body carrying the first contact shape.

- ← *unsigned* int indexBody2, the id of the body carrying the second contact shape.
- ← *unsigned* int withFriction, 0 or 1.
- ← *double* mu.
- ← *double* en.
- ← *double* et (not used).

4.3.3.5 void MBTB_ContactLoadCADFile (unsigned int *contactId*, const std::string & *CADFile1*, const std::string & *CADFile2*, unsigned int *withGraphicModel1*, unsigned int *withGraphicModel2*)

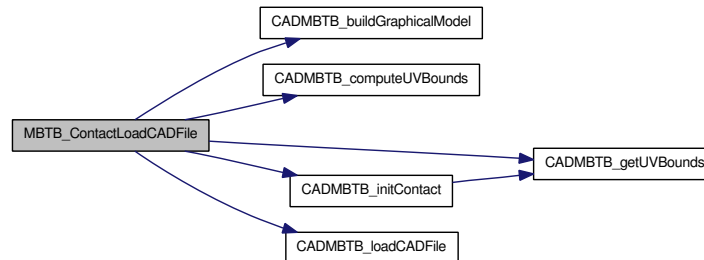
MBTB_ContactLoadCADFile load the cad model of a contact.

It is the shape involved in the contact. Each is include in a container. It contains either some faces or some edges.

Parameters

- [in] unsigned int *contactId*, identifier of Contact.
- [in] const std::string& *CADFile1*, the cad file 1, it must contain one or two faces
- [in] const std::string& *CADFile2*, the cad file 2, it must contains one or two faces, or one or two edges.
- [in] unsigned int *withGraphicModel1*: 1 to draw the corresponding object else 0;
- [in] unsigned int *withGraphicModel2*: 1 to draw the corresponding object else 0;

Here is the call graph for this function:



4.3.3.6 void MBTB_ContactSetDParam (unsigned int *paramId*, unsigned int *contactId*, unsigned int *idShape*, double *v*)

To set a double parameter.(extendable, without modify the API).

This type of function has been chosen to easly set any parameters without modify the module API.

Parameters

- ← *IdParam* : identifier of the param.
1 for offset.

- 2 for artefact length.
- 3 for artefact thershold.
- 4 for the nominal force.

- ← *idContact* : identifier of the contact.
- ← *idShape* : identifier of the shape of the contact (0 or 1).
- ← *v* : value.

4.3.3.7 void MBTB_ContactSetIParam (unsigned int paramId, unsigned int contactId, unsigned int idShape, int v)

To set a interger parameter.(extendable, without modifie the API).

This type of function has been chosen to easely set any parameters without modify the module API.

Parameters

- IdParam* : identifier of the param.
 - 0 for translate offset P1 parameters.
 - 1 normal from face 1.
 - 2 Global draw mode.
- idContact* : identifier of the contact.
- idShape* : identifier of the shape of the contact (0 or 1).
- v* : value.

4.3.3.8 void MBTB_doOnlyProj (unsigned int v)

It allows to perform only the projection algorithm.

This function is usefull only when proj has been activated in MBTB_init.

Parameters

- ← *unsigned* int v, iff 0 then only the projection algorithm is done, the mechanical equations are not simulated.

4.3.3.9 void MBTB_doProj (unsigned int v)

It allows to enable/disable the projection algorithm.

This function is usefull only when proj has been activated in MBTB_init.

Parameters

- ← *unsigned* int v, iff 0 then the projection is disabled.

4.3.3.10 void MBTB_init (unsigned int NumOfBodies, unsigned int NumOfJoints, unsigned int NumberOfContacts)

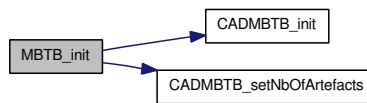
To initialize the MBTB library. It calls the CADMBTB_init.

To initialize the MBTB library (no yet dynamical memory).

Parameters

- ← unsigned int NumOfBodies, the number of bodies.
- ← unsigned int NumOfJoints, the number of joints.
- ← unsigned int NumberOfContacts, the number of contacts.

Here is the call graph for this function:



4.3.3.11 void MBTB_initSimu (double hTS, int withProj)

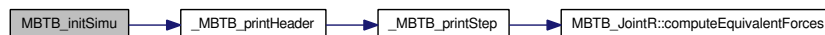
It initializes the simulation.

It consists in building the siconos simulation and al.

Parameters

- ← *double* hTs, time step size.
- ← *int* withProj, iff 0 the projection in done.

Here is the call graph for this function:



4.3.3.12 void MBTB_JointBuild (unsigned int numJ, const std::string & JointName, unsigned int jointType, unsigned int indexDS1, unsigned int indexDS2, SP::SimpleVector jointPosition)

To build a joint.

It builds the joint in the simulator.

Parameters

- ← *unsigned* int numJ, an identifier .
- ← *const* std::string& JointName, a string for the joint name.

- ← *unsigned* int jointType, see enum JOINTS_TYPE.
- ← *unsigned* int indexDS1, index of the first body attached to the joint.
- ← *unsigned* int indexDS2, index of the second body attached to the joint(ignored for JOINTS_TYPE _0).
- ← *SP::SimpleVector* jointPosition a R^7 vector representing the position (translation in R^3 , vector in R^3 , angle in R) that must be applied after the load of the first body.

4.3.3.13 void MBTB_moveBodyToPosWithSpeed (unsigned int numDS, SP::SimpleVector aPos, SP::SimpleVector aVel)

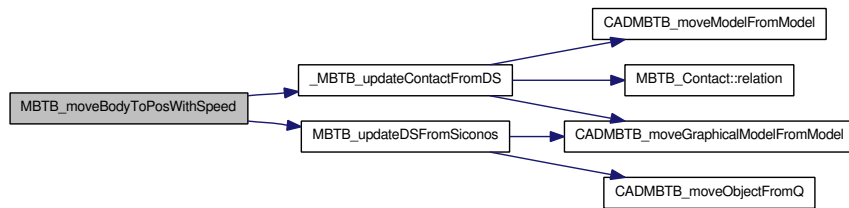
It is a warm start.

It sets the siconos states.

Parameters

- ← *unsigned* int numDS, the id of the ds.
- ← *SP::SimpleVector* aPos, the target position.
- ← *SP::SimpleVector* aVel, the target velocity.

Here is the call graph for this function:



4.3.3.14 void MBTB_run (int nbSteps)

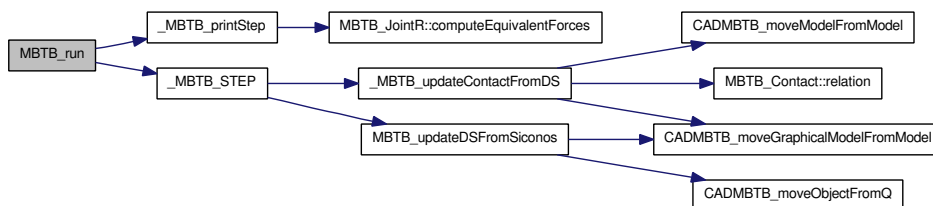
It runs the simulation.

It consists in running nbSteps simulation steps.

Parameters

- ← *int* nbSteps, the number of run step.

Here is the call graph for this function:



4.3.3.15 void MBTB_setGraphicFreq (unsigned int *freq*)

It defines the graphic frequency.

Parameters

← *unsigned* int numDS, the frequency.

4.3.3.16 void MBTB_setJointPoints (unsigned int *numJ*, SP::SimpleVector *G0C1*, SP::SimpleVector *G0C2*)

To set the location where is computed the equivalente forces.

It consists in defining the location of the points in view of compute the equivalente forces of the joint.

Parameters

← *unsigned* int numJ, an identifier .

← *vector* G0C1, where C1 Contact points where the forces of joint must be computed.

← *vector* G0C2, where C2 Contact points where the forces of joint must be computed.

4.3.3.17 void MBTB_setSolverDOption (int *i*, double *value*)

It sets a double value of the solver's parameters.

Parameters

← *int* i, index of the parameter.

← *double* v, the new value.

4.3.3.18 void MBTB_setSolverIOption (int *i*, int *value*)

It sets an integer value of the solver's parameters.

Parameters

← *int* i, index of the parameter.

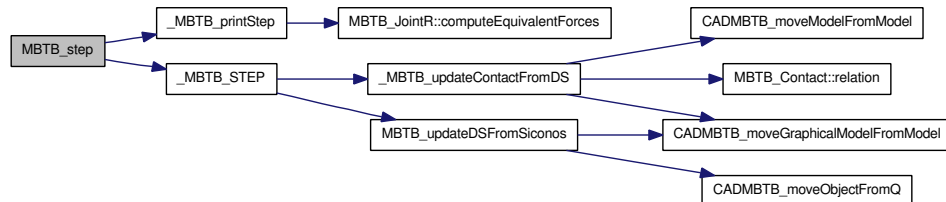
← *int* v, the new value.

4.3.3.19 void MBTB_step ()

It does one step.

It consists in doing one step, including the graphic and output update.

Here is the call graph for this function:

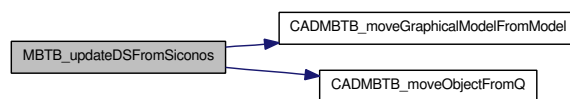


4.3.3.20 void MBTB_updateDSFromSiconos ()

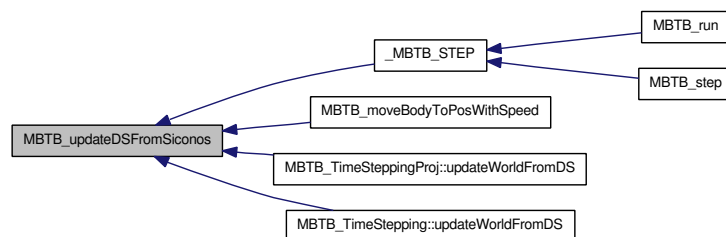
updating the CADs model

MBTB_updateDSFromSiconos It consists in updating the CADs model form the siconos quaternion.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4 CADMBTB_API

This module provides an API on the 3D modeler dedicated to the Multi Bodies simulation.

Functions

- void [CADMBTB_updateGraphic](#) ()
It updates the graphic.
- void [CADMBTB_init](#) (unsigned int NumberOfObj, unsigned int NumberOfContacts)
It initializes the CABMBTB library. It consists in allocating the working memory of n2qn1.
- void [CADMBTB_initContact](#) (unsigned int contactId)
CADMBTB_initContact.
- void [CADMBTB_reset](#) ()
CADMBTB_reset.
- void [CADMBTB_loadCADFile](#) (unsigned int id, const char *fileName)
CADMBTB_loadCADFile.
- void [CADMBTB_buildGraphicalModel](#) (unsigned int id)
CADMBTB_buildGraphicalModel.
- void [CADMBTB_moveModelFromModel](#) (unsigned int idModel1, unsigned int idModel2)
CADMBTB_moveModelFromModel.
- void [CADMBTB_moveGraphicalModelFromModel](#) (unsigned int idGraphicModel, unsigned int id-Model)
CADMBTB_moveGraphicalModelFromModel.
- void [CADMBTB_moveObjectFromQ](#) (unsigned int id, double &x, double &y, double &z, double &q1, double &q2, double &q3, double &q4)
CADMBTB_moveObjectFromQ.
- void [CADMBTB_setLocation](#) (unsigned int id, double &x, double &y, double &z)
CADMBTB_setLocation.
- void [CADMBTB_computeUVBounds](#) (unsigned int id)
CADMBTB_computeUVBounds.
- void [CADMBTB_getUVBounds2](#) (unsigned int id, double &U1, double &U2, double &V1, double &V2)
CADMBTB_getUVBounds2.
- void [CADMBTB_getUVBounds](#) (unsigned int id, double &U1, double &U2, double &V1, double &V2)

CADMBTB_getUVBounds.

- void [CADMBTB_getMinDistance](#) (unsigned int idContact, unsigned int id1, unsigned int id2, double &X1, double &Y1, double &Z1, double &X2, double &Y2, double &Z2, double &nX, double &nY, double &nZ, unsigned int normalFromFace1, double &MinDist)

CADMBTB_getMinDistance.

- void [CADMBTB_setNbOfArtefacts](#) (unsigned int nb)

CADMBTB_setNbOfArtefacts.

- void [CADMBTB_buildLineArtefactLine](#) (unsigned int id, double *X1, double *Y1, double *Z1, double *X2, double *Y2, double *Z2)

CADMBTB_buildLineArtefactLine.

- void [CADMBTB_buildOrientedLineArtefactLine](#) (unsigned int id, double *X1, double *Y1, double *Z1, double *X2, double *Y2, double *Z2)

CADMBTB_buildOrientedLineArtefactLine.

- void [CADMBTB_buildCylinderArtefactLine](#) (unsigned int id, double *X1, double *Y1, double *Z1, double *X2, double *Y2, double *Z2, double *radius)

CADMBTB_buildCylinderArtefactLine.

4.4.1 Detailed Description

This module provides an API on the 3D modeler dedicated to the Multi Bodies simulation. It contains the API used by the module CADMBTB.

It provides the 3D modeler features used for the MBTB.

4.4.2 Function Documentation

- ##### 4.4.2.1 void CADMBTB_buildCylinderArtefactLine (unsigned int *id*, double * *X1*, double * *Y1*, double * *Z1*, double * *X2*, double * *Y2*, double * *Z2*, double * *radius*)

CADMBTB_buildCylinderArtefactLine.

To build a cylinder artefact (forces).

- ##### 4.4.2.2 void CADMBTB_buildLineArtefactLine (unsigned int *id*, double * *X1*, double * *Y1*, double * *Z1*, double * *X2*, double * *Y2*, double * *Z2*)

CADMBTB_buildLineArtefactLine.

To build a line artefact (P1P2).

4.4.2.3 void CADMBTB_buildOrientedLineArtefactLine (unsigned int *id*, double * *X1*, double * *Y1*, double * *Z1*, double * *X2*, double * *Y2*, double * *Z2*)

CADMBTB_buildOrientedLineArtefactLine.

To build a oriented line artefact (n).

4.4.2.4 void CADMBTB_computeUVBounds (unsigned int *id*)

CADMBTB_computeUVBounds.

It computes the bounding box (in parameter space), do only once because OCC is very slow.

Here is the caller graph for this function:



4.4.2.5 void CADMBTB_getMinDistance (unsigned int *idContact*, unsigned int *id1*, unsigned int *id2*, double & *X1*, double & *Y1*, double & *Z1*, double & *X2*, double & *Y2*, double & *Z2*, double & *nX*, double & *nY*, double & *nZ*, unsigned int *normalFromFace1*, double & *MinDist*)

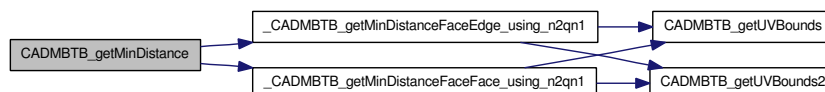
CADMBTB_getMinDistance.

To compute the distance between two objects, P1, P2 are the contact points in the abs frame. n is the normal, in the abs frame.

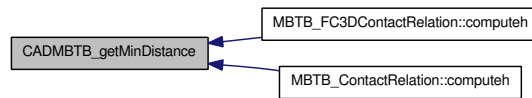
Parameters

- ← unsigned int *idContact*: id of contact(useful for drawing of artefacts).
- ← unsigned int *id1*: the identifier of the first object.
- ← unsigned int *id2*: the identifier of the second object.
- double first coordinate of P1.
- double second coordinate of P1.
- double third coordinate of P1.
- double first coordinate of P2.
- double second coordinate of P2.
- double third coordinate of P2.
- double first coordinate of n.
- double second coordinate of n.
- double third coordinate of n.
- [in/out],:* double distance.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.2.6 void CADMBTB_getUVBounds (unsigned int *id*, double & *U1*, double & *U2*, double & *V1*, double & *V2*)

CADMBTB_getUVBounds.

To get UV bound of the first elem (face or edge) of a shape

Parameters

id [in]: Identifier of the shape.

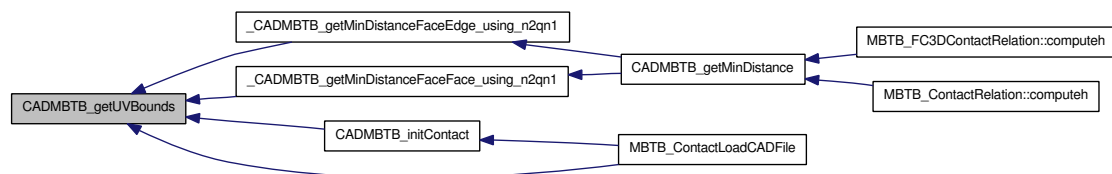
U1 [out]: inf U bound.

U2 [out]: sup U bound.

V1 [out]: inf V bound.

V2 [out]: sup V bound.

Here is the caller graph for this function:



4.4.2.7 void CADMBTB_getUVBounds2 (unsigned int *id*, double & *U1*, double & *U2*, double & *V1*, double & *V2*)

CADMBTB_getUVBounds2.

To get UV bound of the second elem (face or edge) of a shape

Parameters

id [in]: Identifier of the shape.

U1 [out]: inf U bound.

U2 [out]: sup U bound.

V1 [out]: inf V bound.

V2 [out]: sup V bound.

Here is the caller graph for this function:



4.4.2.8 void CADMBTB_init (unsigned int *NumberOfObj*, unsigned int *NumberOfContacts*)

It initializes the CABMBTB library. It consists in allocating the working memory of n2qn1.

Parameters

- ← *unsigned* int NumberOfObj.
- ← *number* of contacts.

Here is the caller graph for this function:



4.4.2.9 void CADMBTB_initContact (unsigned int *contactId*)

CADMBTB_initContact.

It is called to initialize the Bounding Box of the paremters.

It must be call as few as possible because of a bug in OCC: Computation becomming more and more slow.

It must be underline that the parameters used by qnb.f are not saved between the time steps.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.2.10 void CADMBTB_moveGraphicalModelFromModel (unsigned int *idGraphicModel*, unsigned int *idModel*)

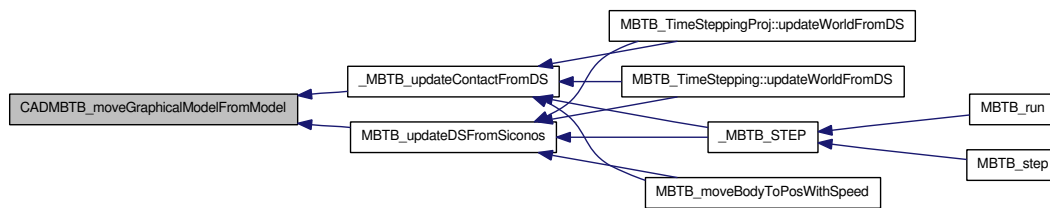
CADMBTB_moveGraphicalModelFromModel.

To move a Graphical model from an object, indeed the graphical model has to follow the mechanical model. Implementation: It consists to set the current transformation using `sGeomTrsf[idGraphicModel]` computed in the function `CADMBTB_moveObjectFromQ`.

Parameters

- ← *unsigned* int idGraphicModel, the identifier of the graphical model
- ← *unsigned* int idModel, the identifier of the referenced object

Here is the caller graph for this function:



4.4.2.11 void CADMBTB_moveModelFromModel (unsigned int idModel1, unsigned int idModel2)

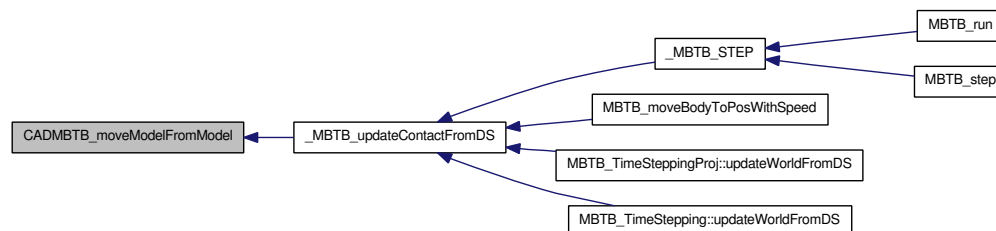
`CADMBTB_moveModelFromModel`.

To move a object from an other on. (useful for contact having the same position of DS) Implementation: It consists to apply the same DISPLACEMENT computed previously by the function `CADMBTB_moveObjectFromQ`. Warning : `sStartTopoDS[id]` is not updated.

Parameters

- ← *unsigned* int idModel1, the identifier of the moved object
- ← *unsigned* int idModel2, the identifier of the referenced object

Here is the caller graph for this function:



4.4.2.12 void CADMBTB_reset ()

`CADMBTB_reset`.

It resets the CABMBTB library: do nothing in current version.

4.4.2.13 void CADMBTB_setLocation (unsigned int *id*, double & *x*, double & *y*, double & *z*)

CADMBTB_setLocation.

Set the location of an object WITHOUT moving it. Useful to defined the coordinate system attached to an object during the initialization.

Parameters

- ← *unsigned* int idModel, the identifier of the object
- ← *double&* x, x translation
- ← *double&* y, y translation
- ← *double&* z, z translation

4.4.2.14 void CADMBTB_setNbOfArtefacts (unsigned int *nb*)

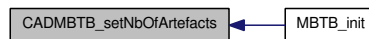
CADMBTB_setNbOfArtefacts.

To declare the number of artefacts: ie graphical decoration(forces, normal, P1P2)

Parameters

[nb],: number of artefacts

Here is the caller graph for this function:



4.4.2.15 void CADMBTB_updateGraphic ()

It updates the graphic.

It assume that CADMBTB_setGraphicContext has been called, else it does nothing.

4.5 CADMBTB_DATA

This file contains the static variables used by the module CADMBTB.

Defines

- #define `NB_OBJ` 50
Maximal number of objects managed by the CADMBTB module.
- #define `DEBUG_CONTACT`
The debug mode.

Enumerations

- enum `CADMBTB_TYPE` { `CADMBTB_TYPE_NONE` = 0, `CADMBTB_TYPE_FACE` = 1, `CADMBTB_TYPE_EDGE` = 2 }
The type of object.

Variables

- unsigned int `sNumberOfObj`
Number of obj, including both contacts and objects.
- TopoDS_Shape `sTopoDS` []
pointer on the first Shape loaded
- unsigned int `sTopoDSType` []
type of obj (Face or Contact)
- double `sTopoDSBinf` []
parameters borne inf
- double `sTopoDSBinf2` []
parameters borne inf of the second shape
- double `sTopoDSBsup` []
parameters borne sup
- double `sTopoDSBsup2` []
parameters borne sup of the second shape
- AIS_Shape * `spAISToposDS` []
Graphical objects.

- double `spAISTrans` []
Graphical objects transparency.
- `gp_Ax3 sStartTopoDS` []
It is the current position of the object. Named start because it is applied a displacement.
- `gp_Trsf sTrsfTopoDS` []
It is the displacement registred in the function `CADMBTB_moveObjectFromQ`, and applied to the related objects (graphic model, contact model).
- `Geom_Transformation sGeomTrsf` []
The location of the object.
- `AIS_InteractiveContext * pAIS_InteractiveContext`
The graphical context, gave by the user.
- `V3d_View * pV3d_View`
The 3D view, gave by the user.
- int `sCmpDump`
For automatic dump.
- unsigned int `sDumpGraphic`
For automatic dump.
- unsigned int `sCmpDumpMan`
For manual dump.
- unsigned int `sNumberOfArtefacts`
Number of artefacts.
- `AIS_Shape * sAISArtefacts` []
Graphical model of the artefacts.
- double `sMinLineLength`
The minimal lenght allowing to draw an artefact.
- double * `sWorkD`
Working memory for `n2qn1`.
- int * `sWorkInt`
Working memory for `n2qn1`.
- int `sNumberOfContacts`
Number of contacts.

4.5.1 Detailed Description

This file contains the static variables used by the module CADMBTB.

About the memory management, the total number of objects is define by NB_OBJ. NB_OBJ is used to allocate the necessary memory.

Each objet is typed with the CADMBTB_TYPE.

4.5.2 Variable Documentation

4.5.2.1 TopoDS_Shape sTopoDS[]

pointer on the first Shape loaded

FOR MBTB : sTopoDS[0 to [MBTB::sNbOfBodies-1](#)] contain the CAD model of the bodies.
sTopoDS[[MBTB::sNbOfBodies](#) to end] contain the CAD model of the contacts.

4.6 CADMBTB_INTERNALTOOLS

This module contains the API of the internal functions of the CADMBTB box.

Functions

- void [_CADMBTB_getMinDistanceFaceFace](#) (const TopoDS_Face &aFace1, const TopoDS_Face &aFace2, Standard_Real &X1, Standard_Real &Y1, Standard_Real &Z1, Standard_Real &X2, Standard_Real &Y2, Standard_Real &Z2, Standard_Real &nX, Standard_Real &nY, Standard_Real &nZ, Standard_Real &MinDist)
- void [_CADMBTB_getMinDistanceFaceFace](#) (unsigned int idFace1, unsigned int idFace2, Standard_Real &X1, Standard_Real &Y1, Standard_Real &Z1, Standard_Real &X2, Standard_Real &Y2, Standard_Real &Z2, Standard_Real &nX, Standard_Real &nY, Standard_Real &nZ, Standard_Real &MinDist)
- void [_CADMBTB_getMinDistanceFaceFace_using_n2qn1](#) (unsigned int idContact, unsigned int idFace1, unsigned int idFace2, Standard_Real &X1, Standard_Real &Y1, Standard_Real &Z1, Standard_Real &X2, Standard_Real &Y2, Standard_Real &Z2, Standard_Real &nX, Standard_Real &nY, Standard_Real &nZ, unsigned int normalFromFace1, Standard_Real &MinDist)
- void [_CADMBTB_getMinDistanceFaceEdge_using_n2qn1](#) (unsigned int idContact, unsigned int idFace1, unsigned int idFace2, Standard_Real &X1, Standard_Real &Y1, Standard_Real &Z1, Standard_Real &X2, Standard_Real &Y2, Standard_Real &Z2, Standard_Real &nX, Standard_Real &nY, Standard_Real &nZ, Standard_Real &MinDist)

4.6.1 Detailed Description

This module contains the API of the internal functions of the CADMBTB box. It provides the geometrical functions. It consists in computing the nearest points between objects.

It computes the cartesian coordinates and the corresponding normal.

For more details, see documentation about the functions.

4.6.2 Function Documentation

4.6.2.1 void [_CADMBTB_getMinDistanceFaceEdge_using_n2qn1](#) (unsigned int *idContact*, unsigned int *idFace1*, unsigned int *idFace2*, Standard_Real & *X1*, Standard_Real & *Y1*, Standard_Real & *Z1*, Standard_Real & *X2*, Standard_Real & *Y2*, Standard_Real & *Z2*, Standard_Real & *nX*, Standard_Real & *nY*, Standard_Real & *nZ*, Standard_Real & *MinDist*)

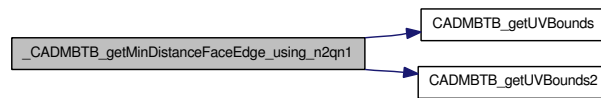
To compute distance using n2qn1 algorithm. This function manages the case where the object idFace1 contains one or two faces.

Parameters

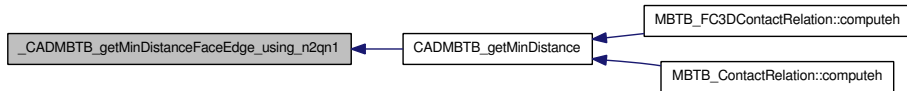
← *unsigned* int idContact the identifier of the contact.

- ← *unsigned int* idFace1 the identifier of the first object containing one or tow faces, redundancy parameter, must be equal to sNumberOfObj+(2*idContact-2*sNumberOfContacts).
- ← *unsigned int* idFace2 the identifier of the second object containing one edge, redundancy parameter, must be equal to sNumberOfObj+(2*idContact+1-2*sNumberOfContacts).
- *Standard_Real&* X1 the first coordinate of the point attached to the first object.
- *Standard_Real&* Y1 the second coordinate of the point attached to the first object.
- *Standard_Real&* Z1 the third coordinate of the point attached to the first object.
- *Standard_Real&* X2 the first coordinate of the point attached to the second object.
- *Standard_Real&* Y2 the second coordinate of the point attached to the second object.
- *Standard_Real&* Z2 the third coordinate of the point attached to the second object.
- *Standard_Real&* MinDist the distance between the points.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.2 void _CADMBTB_getMinDistanceFaceFace (unsigned int idFace1, unsigned int idFace2, Standard_Real & X1, Standard_Real & Y1, Standard_Real & Z1, Standard_Real & X2, Standard_Real & Y2, Standard_Real & Z2, Standard_Real & nX, Standard_Real & nY, Standard_Real & nZ, Standard_Real & MinDist)

To compute distance using OCC algorithm. Deprecated.

Parameters

- ← *TopoDS_Face&* aFace1 the first object.
- ← *TopoDS_Face&* aFace2 the second object.
- *Standard_Real&* X1 the first coordinate of the point attached to the first object.
- *Standard_Real&* Y1 the second coordinate of the point attached to the first object.
- *Standard_Real&* Z1 the third coordinate of the point attached to the first object.
- *Standard_Real&* X2 the first coordinate of the point attached to the second object.
- *Standard_Real&* Y2 the second coordinate of the point attached to the second object.
- *Standard_Real&* Z2 the third coordinate of the point attached to the second object.
- *Standard_Real&* MinDist the distance between the points.

4.6.2.3 `void _CADMBTB_getMinDistanceFaceFace (const TopoDS_Face & aFace1, const TopoDS_Face & aFace2, Standard_Real & X1, Standard_Real & Y1, Standard_Real & Z1, Standard_Real & X2, Standard_Real & Y2, Standard_Real & Z2, Standard_Real & nX, Standard_Real & nY, Standard_Real & nZ, Standard_Real & MinDist)`

To compute distance using OCC algorithm. Deprecated.

Parameters

- ← *TopoDS_Face&* aFace1 the first object.
- ← *TopoDS_Face&* aFace2 the second object.
- *Standard_Real&* X1 the first coordinate of the point attached to the first object.
- *Standard_Real&* Y1 the second coordinate of the point attached to the first object.
- *Standard_Real&* Z1 the third coordinate of the point attached to the first object.
- *Standard_Real&* X2 the first coordinate of the point attached to the second object.
- *Standard_Real&* Y2 the second coordinate of the point attached to the second object.
- *Standard_Real&* Z2 the third coordinate of the point attached to the second object.
- *Standard_Real&* MinDist the distance between the points.

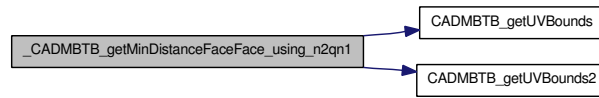
4.6.2.4 `void _CADMBTB_getMinDistanceFaceFace_using_n2qn1 (unsigned int idContact, unsigned int idFace1, unsigned int idFace2, Standard_Real & X1, Standard_Real & Y1, Standard_Real & Z1, Standard_Real & X2, Standard_Real & Y2, Standard_Real & Z2, Standard_Real & nX, Standard_Real & nY, Standard_Real & nZ, unsigned int normalFromFace1, Standard_Real & MinDist)`

To compute distance using n2qn1 algorithm. This function manages the case where the object idFace1 contains one or two faces.

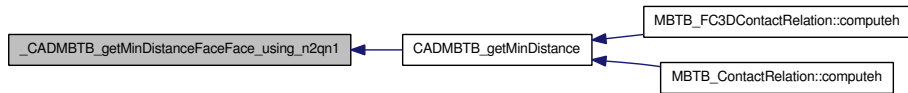
Parameters

- ← *unsigned* int idContact the identifier of the contact.
- ← *unsigned* int idFace1 the identifier of the first object containing one or two faces, redundancy parameter, must be equal to sNumberOfObj+(2*idContact-2*sNumberOfContacts).
- ← *unsigned* int idFace2 the identifier of the second object containing one face, redundancy parameter, must be equal to sNumberOfObj+(2*idContact+1-2*sNumberOfContacts).
- *Standard_Real&* X1 the first coordinate of the point attached to the first object.
- *Standard_Real&* Y1 the second coordinate of the point attached to the first object.
- *Standard_Real&* Z1 the third coordinate of the point attached to the first object.
- *Standard_Real&* X2 the first coordinate of the point attached to the second object.
- *Standard_Real&* Y2 the second coordinate of the point attached to the second object.
- *Standard_Real&* Z2 the third coordinate of the point attached to the second object.
- ← *unsigned* int normalFromFace1, if normalFromFace1 the normal is computed from the object id-Face1.
- *Standard_Real&* MinDist the distance between the points.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7 CADMBTB_PYTHON_API

This module contains the python API of the module CADMBTB.

Functions

- void [CADMBTB_loadArtefactCADFile](#) (const char *fileName, double trans)
- void [CADMBTB_setGraphicContext](#) (AIS_InteractiveContext &aisContext)
It consists to initialize the graphic context. (example : CADMBTB_setGraphicContext(display.Context) from PYTHON).
- void [CADMBTB_setGraphicView](#) (V3d_View &aView)
To set the current view.
- void [CADMBTB_disableGraphic](#) ()
To disable graphic.
- void [CADMBTB_setShapeDParam](#) (unsigned int IdParam, unsigned int idShape, double v)
CADMBTB_setShapeDParam To set a double parameter.(extendable, without modify the API).
- void [CADMBTB_setContactDParam](#) (unsigned int IdParam, unsigned int idContact, unsigned int idShape, double v)
To set a double parameter.(extendable, without modify the API).
- void [CADMBTB_setIParam](#) (unsigned int IdParam, int v)
To set a int parameter. (extendable, without modify the API).
- void [CADMBTB_DumpGraphic](#) ()
Manual dump of the 3D view.

4.7.1 Detailed Description

This module contains the python API of the module CADMBTB. It provides the graphic functions allowing to plug an 3D view, from OCC. The 3D view is optional.

If there is no 3D view, the internal graphical management will be ignored.

It also provides a functions to load an artefact CAD model and set internal parameters.

For more details see the documentation of the functions.

4.7.2 Function Documentation

4.7.2.1 void CADMBTB_loadArtefactCADFile (const char * *fileName*, double *trans*)

Load a step CAD file.

Parameters

const char * *fileName*, a CAD file.

4.7.2.2 void CADMBTB_setContactDParam (unsigned int *IdParam*, unsigned int *idContact*, unsigned int *idShape*, double *v*)

To set a double parameter.(extendable, without modify the API).

This type of function has been chosen to easly set any parameters without modify the module API.

Parameters

IdParam : identifier of the param.

0 for transparency.

idContact : identifier of the contact.

idShape : identifier of the shape of the contact (0 or 1).

v : value.

Here is the call graph for this function:

**4.7.2.3 void CADMBTB_setIParam (unsigned int *IdParam*, int *v*)**

To set a int parameter. (extendable, without modify the API).

This type of function has been chosen to easly set any parameters without modify the module API.

Parameters

IdParam : identifier of the param, used only for enable/disable the dump of the 3D view.

v : value.

4.7.2.4 void CADMBTB_setShapeDParam (unsigned int *IdParam*, unsigned int *idShape*, double *v*)

CADMBTB_setShapeDParam To set a double parameter.(extendable, without modify the API).

This type of function has been chosen to easly set any parameters without modify the module API.

Parameters

IdParam : identifier of the param.

0 for transparency.

idShape : identifier of the shape.

v : value.

Here is the caller graph for this function:



4.8 SliderCrankexample

This documentation is build from the python example Tests/SliderCrank, it contains an example of a simple multibody systems.

Variables

- int `bodydef::NBBODIES` = 3
REQUIRED BODIES DESCRIPTION REQUIRED number of bodies.
- int `bodydef::BATIE` = 1
Identifier of the word, an object attached to the referential frame.
- int `bodydef::PART1` = 0
Identifier of the first body.
- int `bodydef::PART2` = 1
Identifier of the second body.
- int `bodydef::PISTON` = 2
Identifier of the third body.
- tuple `bodydef::body` = `numpy.array(['part1','part2','slider'])`
REQUIRED the name of the bodies.
- tuple `bodydef::initPos`
REQUIRED the initial position of the bodies.
- tuple `bodydef::initVel`
REQUIRED the initial velocities of the bodies.
- tuple `bodydef::initCenterMass`
REQUIRED Initial position of the center of mass.
- tuple `bodydef::m` = `array.array('d',[0.038,0.038,0.076])`
REQUIRED mass of the bodies.
- tuple `bodydef::inertialMatrix`
REQUIRED inertial matrices.
- tuple `bodydef::afile`
REQUIRED the CAD files.
- string `bodydef::plugin` = `'/MBTB/plugin/SliderCrank/libSliderCrankPlugin.so'`
REQUIRED the library for the plugged forces.

- tuple `bodydef::fctf` = `numpy.array(['externalForcesB1','externalForcesB2','externalForcesS',])`
REQUIRED the external forces.
- tuple `bodydef::fctm` = `numpy.array(['','',''])`
REQUIRED the external momentums.
- int `bodydef::NBJOINTS` = 3
REQUIRED the number of joints.
- tuple `bodydef::jointName`
REQUIRED the name of the joints.
- tuple `bodydef::jointType`
REQUIRED the joint type.
- tuple `bodydef::jointBody1`
REQUIRED the index of the first body involved in the joint.
- tuple `bodydef::jointBody2`
REQUIRED the index of the second body involved in the joint.
- tuple `bodydef::jointPos`
REQUIRED the joint position.
- int `bodydef::NBCONTRACTS` = 3
REQUIRED the number of contacts.
- tuple `bodydef::contactName`
REQUIRED the names of each contact.
- tuple `bodydef::afileContact1`
REQUIRED the CAD file attached to the first body involved in the contact.
- tuple `bodydef::afileContact2`
REQUIRED the CAD file attached to the second body involved in the contact.
- tuple `bodydef::contactBody1`
REQUIRED the identifier of the first body involved in the contact.
- tuple `bodydef::contactBody2`
REQUIRED the identifier of the second body involved in the contact.
- tuple `bodydef::contactOffset`
the artificial offset sub to the geometrical computation.
- tuple `bodydef::contactOffsetP1`
defining if the offset is applied to the first surface.
- tuple `bodydef::contactNormalFromFace1`
defining if the normal is computed from the first surface.

- tuple `bodydef::contactType3D`
for each contact, 1 if friction 3d, 0 if unilateral constrainte.
- tuple `bodydef::contactmu`
for each contact, the mu parameter, useful in the case of friction.
- tuple `bodydef::contacten`
the restitution coeficient of each contact.

4.8.1 Detailed Description

This documentation is build from the python example Tests/SliderCrank, it contains an example of a simple multibody systems. This example is the body system corresponding to the example developed in the paper : MODELING AND ANALYSIS OF MULTIBODY SYSTEMS WITH TRANSLATIONAL CLEARANCE JOINTS BASED ON THE NSDS Paulo Flores, Remco Leine, Christoph Glocker. python ./run.py and result.gp lead to the Fig 11 (d) of this paper.

This example is based on cad files located in Multibody/Tests/CAO/SliderCrank.

The option WITH_CLEARANCE_ON_RODE can be set to 1 to add clearance between the rode 1 and the rode 2.

4.8.2 Variable Documentation

4.8.2.1 tuple `bodydef::afile`

Initial value:

```
numpy.array([SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/body1.step',
            SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/body2.step',
            SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/Slider.ste
            p'])
```

REQUIRED the CAD files.

4.8.2.2 tuple `bodydef::afileContact1`

Initial value:

```
numpy.array([
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/contact_b_cyl.step',
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/contact_h_cyl.step',
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/RingBody1.stp'
])
```

REQUIRED the CAD file attached to the first body involved in the contact.

4.8.2.3 tuple bodydef::afileContact2

Initial value:

```
numpy.array([
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/chamber.step',
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/chamber.step',
    SALADYN_DIR+'/trunk/Multibody/Tests/CAO/SliderCrank/AxisBody2.stp'
])
```

REQUIRED the CAD file attached to the second body involved in the contact.

4.8.2.4 int bodydef::BATIE = 1

Identifier of the word, an object attached to the referential frame.

4.8.2.5 tuple bodydef::contactBody1

Initial value:

```
array.array('I', [
    PISTON,
    PISTON,
    PART1])
```

REQUIRED the identifier of the first body involved in the contact.

4.8.2.6 tuple bodydef::contactBody2

Initial value:

```
array.array('i', [
    BATIE,
    BATIE,
    PART2])
```

REQUIRED the identifier of the second body involved in the contact.

4.8.2.7 tuple bodydef::contacten

Initial value:

```
array.array('d', [
    0.4,
    0.4,
    0.0])
```

the restitution coefficient of each contact.

4.8.2.8 tuple bodydef::contactmu

Initial value:

```
array.array('d',[
    0.01,
    0.01,
    0.01])
```

for each contact, the mu parameter, useful in the case of friction.

4.8.2.9 tuple bodydef::contactName

Initial value:

```
numpy.array([
    'contact_h',
    'contact_b',
    'contact_boby1_2'])
```

REQUIRED the names of each contact.

4.8.2.10 tuple bodydef::contactNormalFromFace1

Initial value:

```
array.array('I',[
    0,
    0,
    0])
```

defining if the normal is computed from the first surface.

4.8.2.11 tuple bodydef::contactOffset

Initial value:

```
array.array('d',[
    0.024,
    0.024,
    0.009])
```

the artificial offset sub to the geometrical computation.

4.8.2.12 tuple bodydef::contactOffsetP1

Initial value:

```
array.array('I',[
    0,
    0,
    0])
```

defining if the offset is applied to the first surface.

Useful to place the contact point.

4.8.2.13 tuple bodydef::contactType3D

Initial value:

```
array.array('I',[
    1,
    1,
    1])
```

for each contact, 1 if friction 3d, 0 if unilateral constrainte.

4.8.2.14 tuple bodydef::fctf = numpy.array(['externalForcesB1','externalForcesB2','externalForcesS',])

REQUIRED the external forces.

4.8.2.15 tuple bodydef::fctm = numpy.array(["","",""])

REQUIRED the external momentums.

4.8.2.16 tuple bodydef::inertialMatrix

Initial value:

```
numpy.array([( (1,0,0), (0,7.4e-5,0), (0,0,1)),
              ((1,0,0), (0,5.9e-4,0), (0,0,1)),
              ((1,0,0), (0,2.7e-6,0), (0,0,1))])
```

REQUIRED inertial matrices.

4.8.2.17 tuple bodydef::initCenterMass

Initial value:

```
numpy.array([(0.5*11,0.00,0.00),
              (0.5*12,0.00,0.00),
              (a,0,0)])
```

REQUIRED Initial position of the center of mass.

It is the position of the center of mass just after loading, ie with the position (0,0,0,1,0,0,0).

4.8.2.18 tuple bodydef::initPos**Initial value:**

```
numpy.array([(0,0,0, 0,0,1,0),
             (11,0.0,0.0, 0,0,1,0),
             (11+12-a,0,0, 0,1,0,0)])
```

REQUIRED the initial position of the bodies.

4.8.2.19 tuple bodydef::initVel**Initial value:**

```
numpy.array([(0,0,-0.5*w10*11,0,w10,0),
             (0,0,-0.5*w10*11,0,w20,0),
             (0,0,0,0,w30,0)])
```

REQUIRED the initial velocities of the bodies.

4.8.2.20 tuple bodydef::jointBody1**Initial value:**

```
array.array('I',[PART1,
                 PART2,
                 PART1])
```

REQUIRED the index of the first body involved in the joint.

4.8.2.21 tuple bodydef::jointBody2**Initial value:**

```
array.array('I',[0,
                 PISTON,
                 PART2])
```

REQUIRED the index of the second body involved in the joint.

4.8.2.22 tuple bodydef::jointName**Initial value:**

```
numpy.array(['Part1_0',
             'Part2_Piston',
             'Part1_2'])
```

REQUIRED the name of the joints.

4.8.2.23 tuple bodydef::jointPos

Initial value:

```
numpy.array([(0,1,0, 0.00,00,00),  
            (0,1,0, 12,00,00),  
            (0,1,0, 11,00,00)])
```

REQUIRED the joint position.

4.8.2.24 tuple bodydef::jointType

Initial value:

```
array.array('I',[mbtb.PIVOT_0,  
                mbtb.PIVOT_1,  
                mbtb.PIVOT_1])
```

REQUIRED the joint type.

4.8.2.25 tuple bodydef::m = array.array('d',[0.038,0.038,0.076])

REQUIRED mass of the bodies.

4.8.2.26 int bodydef::PART1 = 0

Identifier of the first body.

4.8.2.27 int bodydef::PART2 = 1

Identifier of the second body.

4.8.2.28 int bodydef::PISTON = 2

Identifier of the third body.

4.8.2.29 string bodydef::plugin = '/MBTB/plugin/SliderCrank/libSliderCrankPlugin.so'

REQUIRED the library for the plugged forces.

4.9 MbtbLocalOption

This documentation is build from the python example Tests/SliderCrank, it contains an example of local options.

Variables

- tuple `mbtbLocalOptions::bodyDraw` = `array.array('I',[1,1,1,0])`
To define if a body is drew.
- tuple `mbtbLocalOptions::bodyTrans`
To define the transparency level of the body.
- float `mbtbLocalOptions::ContactArtefactLength` = 0.1
- int `mbtbLocalOptions::ArtefactThershold` = 1
- tuple `mbtbLocalOptions::contactDraw1`
To define if the first surface of the contact is drew.
- tuple `mbtbLocalOptions::contactDraw2`
To define if the second surface of the contact is drew.
- tuple `mbtbLocalOptions::contactTrans1`
To define the transparency level of the first contact surface.
- tuple `mbtbLocalOptions::contactTrans2`
To define the transparency level of the second contact surface.
- int `mbtbLocalOptions::with3D` = 0
It must be set to 1 to run in a 3D view.
- int `mbtbLocalOptions::freqUpdate` = 10
3D viewer update frequency.
- int `mbtbLocalOptions::stepSize` = 2
Simulation parameters time step size.
- int `mbtbLocalOptions::stepNumber` = 5000
Simulation parameters number of steps.
- int `mbtbLocalOptions::withProj` = 0
To activate the projection algorithm.
- int `mbtbLocalOptions::withReduced` = 2
Solver option.
- int `mbtbLocalOptions::solverTol` = 1

Solver option.

- int `mbtbLocalOptions::solverIt` = 100000
Solver option.
- int `mbtbLocalOptions::gotoPos` = 0
- int `mbtbLocalOptions::NBARTEFACTS` = 1
The number of artefacts.
- tuple `mbtbLocalOptions::Artefactfile`
CAD file of the artefacts.
- tuple `mbtbLocalOptions::ArtefactTrans`
transparency of the artefacts

4.9.1 Detailed Description

This documentation is build from the python example Tests/SliderCrank, it contains an example of local options. This file is optional, if it doesn't exit, the default voptions will be use.

4.9.2 Variable Documentation

4.9.2.1 tuple `mbtbLocalOptions::Artefactfile`

Initial value:

```
numpy.array([
    SALADYN_DIR+' /trunk/Multibody/Tests/CAO/SliderCrank/artefact2.step' ])
```

CAD file of the artefacts.

4.9.2.2 tuple `mbtbLocalOptions::ArtefactTrans`

Initial value:

```
array.array('d', [
    0.8])
```

transparency of the artefacts

4.9.2.3 tuple `mbtbLocalOptions::bodyDraw = array.array('I',[1,1,1,0])`

To define if a body is drew.

4.9.2.4 tuple mbtbLocalOptions::bodyTrans

Initial value:

```
array.array('d',[
    2.5,
    0.7,
    0.5])
```

To define the transparency level of the body.

4.9.2.5 tuple mbtbLocalOptions::contactDraw1

Initial value:

```
array.array('I',[
    1,
    1,
    1])
```

To define if the first surface of the contact is drawn.

4.9.2.6 tuple mbtbLocalOptions::contactDraw2

Initial value:

```
array.array('I',[
    0,
    0,
    1])
```

To define if the second surface of the contact is drawn.

4.9.2.7 tuple mbtbLocalOptions::contactTrans1

Initial value:

```
array.array('d',[
    2.,
    2.,
    2.])
```

To define the transparency level of the first contact surface.

4.9.2.8 tuple mbtbLocalOptions::contactTrans2

Initial value:

```
array.array('d',[
    0.7,
    0.7,
    2.])
```

To define the transparency level of the second contact surface.

4.9.2.9 int mbtbLocalOptions::freqUpdate = 10

3D viewer update frequency.

4.9.2.10 int mbtbLocalOptions::stepNumber = 5000

Simulation parameters number of steps.

Useful if with3D=0.

4.9.2.11 int mbtbLocalOptions::stepSize = 2

Simulation parameters time step size.

4.9.2.12 int mbtbLocalOptions::with3D = 0

It must be set to 1 to run in a 3D view.

4.9.2.13 int mbtbLocalOptions::withProj = 0

To activate the projection algorithm.

Chapter 5

Class Documentation

5.1 MBTB_Body Class Reference

This class implements a body in a multi-bodies system. It inherits from `Siconos::NewtonEulerDS`.

```
#include <MBTB_Body.hpp>
```

Public Member Functions

- [MBTB_Body](#) (`SP::SimpleVector q0`, `SP::SimpleVector v0`, `double &mass`, `SP::SimpleMatrix I`, `SP::SimpleVector centerOfMass`, `const std::string &BodyName`, `const std::string &CADFile`, `const std::string &pluginLib`, `const std::string &plunginFct`)
- [MBTB_Body](#) (`SP::SimpleVector q0`, `SP::SimpleVector v0`, `double &mass`, `SP::SimpleMatrix I`, `SP::SimpleVector centerOfMass`, `const std::string &BodyName`, `const std::string &CADFile`)

Protected Attributes

- `const std::string &_mBodyName`
The name of the body.
- `const std::string &_cadFileName`
The cad file.

5.1.1 Detailed Description

This class implements a body in a multi-bodies system. It inherits from `Siconos::NewtonEulerDS`.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 MBTB_Body::MBTB_Body (*SP::SimpleVector* $q0$, *SP::SimpleVector* $v0$, *double* & $mass$, *SP::SimpleMatrix* I , *SP::SimpleVector* $centerOfMass$, *const std::string* & $BodyName$, *const std::string* & $CADFile$, *const std::string* & $pluginLib$, *const std::string* & $plunginFct$)

builder

Parameters

- ← *SP::SimpleVector* $q0$, initial position of the center of mass.
- ← *SP::SimpleVector* $v0$, initial velocity.
- ← *double* & $mass$, the mass.
- ← *SP::SimpleMatrix* I , matrix in $\mathbb{R}^{3,3}$
- ← *SP::SimpleVector* $centerOfMass$, coordinate of the mass center in the just loaded model
- ← *const std::string* & $BodyName$, a string for the body name.
- ← *const std::string* & $CADFile$, the cad file.
- ← *const std::string* & $pluginLib$, the path to the plugin library.
- ← *const std::string* & $plunginFct$, the name of the plugged fonction

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

- MBTB/MBTB_Body.hpp
- MBTB/MBTB_Body.cpp

5.2 MBTB_Contact Class Reference

A Contact class.

```
#include <MBTB_Contact.hpp>
```

Public Member Functions

- **MBTB_Contact** (unsigned int id, const std::string &contactName, unsigned int indexBody1, int indexBody2, unsigned int indexCAD1, unsigned int indexCAD2, int withFriction)
Builder. It builds the member _Relation either MBTB_ContactRelation or MBTB_FC3DContactRelation.
- SP::NewtonEulerRImpact **relation** ()
To get the relation.
- char * **contactName** ()
To get the name of the contact.

Public Attributes

- unsigned int **_id**
The id of the contact.
- unsigned int **_indexBody1**
The index of the body carrying the first face of the contact.
- int **_indexBody2**
The index of the body carrying the second face of the contact. -1 means the face doesn't move.
- unsigned int **_indexCAD1**
The index of the cad model.
- unsigned int **_indexCAD2**
The index of the cad model.
- int **_withFriction**
The value 0 means without friction 1 means with.
- double **_en**
The normal restitution coefficient.
- double **_et**
The tangential restitution coefficient(not used).
- double **_Offset**

The offset sub to the distance computation.

- unsigned int [_normalFromFace1](#)

If *normalFromFace1*, the normal is computed from the *face1* else from the *face2*.

- unsigned int [_OffsetP1](#)

To know if *P1* is translated of $_{Offset} * N$ or *P2*.

Protected Attributes

- char [_ContactName](#) [256]

The contact name.

- double [_dist](#)
- double [_curTimeh](#)
- SP::NewtonEulerRImpact [_Relation](#)

Friends

- class [MBTB_ContactRelation](#)

For the unilateral case.

- class [MBTB_FC3DContactRelation](#)

For the Coulomb friction case.

5.2.1 Detailed Description

A Contact class. This class describes a CAD contact: it contains the identifier of the CAD model of the contact. It is either only unilateral or unilateral with Coulomb friction, depending on *withFriction* parameter of the builder. It builds the corresponding member *_Relation* either [MBTB_ContactRelation](#) or [MBTB_FC3DContactRelation](#).

5.2.2 Constructor & Destructor Documentation

5.2.2.1 MBTB_Contact::MBTB_Contact (unsigned int *id*, const std::string & *contactName*, unsigned int *indexBody1*, int *indexBody2*, unsigned int *indexCAD1*, unsigned int *indexCAD2*, int *withFriction*)

Builder. It builds the member *_Relation* either [MBTB_ContactRelation](#) or [MBTB_FC3DContactRelation](#).

Parameters

← *id* an unsigned int, the identifier of the contact.

← *contactName* the name of the contact, is copied in the member *_ContactName*.

- ← *indexBody1* an unsigned int, index of the first body carrying the surface of contact.
- ← *indexBody2* an int, index of the first body carrying the surface of contact. If negatif, it means the body is the not movable (ex: the ground).
- ← *indexCAD1* an unsigned int the index of the cad model defined the surface of the indexBody1.
- ← *indexCAD2* an unsigned int the index of the cad model defined the surface of the indexBody2.
- ← *withFriction* an int, if 0, the contact is without friction.

5.2.3 Member Data Documentation

5.2.3.1 double MBTB_Contact::_curTimeh [protected]

To avoid unuseful call at the same date.

5.2.3.2 SP::NewtonEulerRImpact MBTB_Contact::_Relation [protected]

Built during the building of the [MBTB_Contact](#). A link to the relation, either [MBTB_ContactRelation](#) or [MBTB_FC3DContactRelation](#).

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

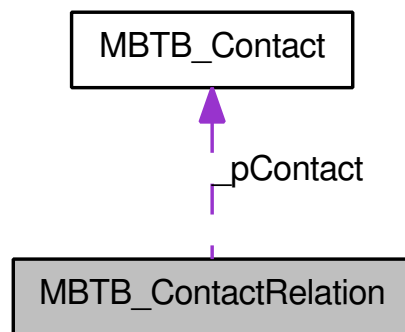
- MBTB/MBTB_Contact.hpp
- MBTB/MBTB_Contact.cpp

5.3 MBTB_ContactRelation Class Reference

It is a relation dedicated for the simple unilateral (ie: without Coulomb friction).

```
#include <MBTB_ContactRelation.hpp>
```

Collaboration diagram for MBTB_ContactRelation:



Public Member Functions

- [MBTB_ContactRelation](#) ([MBTB_Contact](#) *pC)
Builder.
- virtual void [computeh](#) (double time)
This function has to compute the distance between the objects.
- virtual [~MBTB_ContactRelation](#) ()
Doing nothing.

Protected Attributes

- [MBTB_Contact](#) * [_pContact](#)

5.3.1 Detailed Description

It is a relation dedicated for the simple unilateral (ie: without Coulomb friction). Aggregation to the class [MBTB_Contact](#), the member [_pContact](#) contains the CAD informations. It derives from `Siconos::NewtonEulerRImpact`. This class does the link between CAD and Siconos.

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

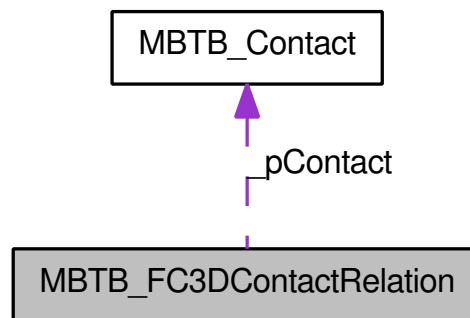
- MBTB/MBTB_ContactRelation.hpp
- MBTB/MBTB_ContactRelation.cpp

5.4 MBTB_FC3DContactRelation Class Reference

It is a relation dedicated for the unilateral constraint with Coulomb friction.

```
#include <MBTB_FC3DContactRelation.hpp>
```

Collaboration diagram for MBTB_FC3DContactRelation:



Public Member Functions

- [MBTB_FC3DContactRelation \(MBTB_Contact * _pContact\)](#)
Builder.
- virtual void [computeh](#) (double time)
This function has to compute the distance between the objects.
- virtual [~MBTB_FC3DContactRelation \(\)](#)
Doing nothing.

Protected Attributes

- [MBTB_Contact * _pContact](#)
The member containing the CAD properties.

5.4.1 Detailed Description

It is a relation dedicated for the unilateral constraint with Coulomb friction. Aggregation to the class [MBTB_Contact](#), the member `_pContact` contains the CAD informations. It derives from `Siconos::NewtonEulerRFC3D`. This class does the link between CAD and Siconos.

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

- MBTB/MBTB_FC3DContactRelation.hpp
- MBTB/MBTB_FC3DContactRelation.cpp

5.5 MBTB_JointR Class Reference

This class implements a joint in a multi-bodies system. It is an aggregation to the class `siconos::NewtonEulerR`. Mainly, it consists in adding members needed for the computation of joint forces.

```
#include <MBTB_JointR.hpp>
```

Public Member Functions

- void `computeEquivalentForces ()`
*It consists in building the system $_M * F = BLambda$ and solving it.*

Public Attributes

- `SP::NewtonEulerR _jointR`
it is assumed that `_jointR` is a pivot.
- `SP::NewtonEulerDS _ds1`
The first dynamical systems of the joint.
- `SP::SimpleVector _G0C1`
vector $G0C_i$, where C_i Contact points where the forces of joint must be computed.
- `SP::SimpleVector _G0C2`
vector $G0C_i$, where C_i Contact points where the forces of joint must be computed.
- `SP::SimpleVector _F`
Joint forces. $F1: F[0,1,2]$. $F2: F[3,4,5]$.
- `SP::SimpleMatrix _M`
*A matrix such that $_M * F = BLambda$.*

Friends

- class `PivotJointR`

5.5.1 Detailed Description

This class implements a joint in a multi-bodies system. It is an aggregation to the class `siconos::NewtonEulerR`. Mainly, it consists in adding members needed for the computation of joint forces.

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

- `MBTB/MBTB_JointR.hpp`
- `MBTB/MBTB_JointR.cpp`

5.6 MBTB_TimeStepping Class Reference

This class implements the time stepping of a multi-bodies system. It inherits from Siconos::TimeStepping. It consists in update the CAD word during the simulation.

```
#include <MBTB_TimeStepping.hpp>
```

Public Member Functions

- [MBTB_TimeStepping](#) (SP::TimeDiscretisation td, SP::OneStepIntegrator osi, SP::OneStepNSProblem osnspb_velo)
builder.
- virtual void [updateWorldFromDS](#) ()
Overloading of updateWorldFromDS.
- virtual bool [predictorDeactivate](#) (SP::UnitaryRelation ur, unsigned int i)
Overloading of predictorDeactivate.
- virtual bool [predictorActivate](#) (SP::UnitaryRelation ur, unsigned int i)
Overloading of predictorActivate.

5.6.1 Detailed Description

This class implements the time stepping of a multi-bodies system. It inherits from Siconos::TimeStepping. It consists in update the CAD word during the simulation.

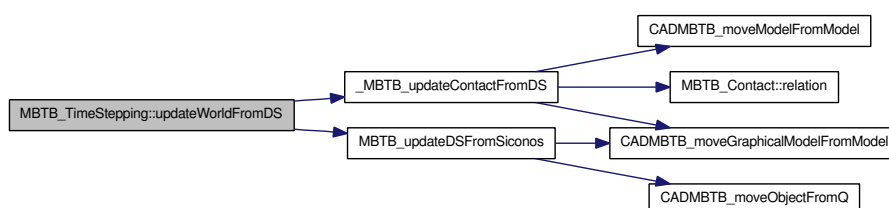
5.6.2 Member Function Documentation

5.6.2.1 void MBTB_TimeStepping::updateWorldFromDS () [virtual]

Overloading of updateWorldFromDS.

It consists in updating the cad model from siconos.

Here is the call graph for this function:



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

- MBTB/MBTB_TimeStepping.hpp
- MBTB/MBTB_TimeStepping.cpp

5.7 MBTB_TimeSteppingProj Class Reference

This class implements the time stepping with projection of a multi-bodies system. It inherits from Siconos::TimeSteppingProjectOnConstraints. It consists in update the CAD word during the simulation.

```
#include <MBTB_TimeSteppingProj.hpp>
```

Public Member Functions

- [MBTB_TimeSteppingProj](#) (SP::TimeDiscretisation td, SP::OneStepIntegrator osi, SP::OneStepNSProblem osnspb_velo, SP::OneStepNSProblem osnspb_pos)
builder.
- virtual void [updateWorldFromDS](#) ()
Overloading of updateWorldFromDS.
- virtual bool [predictorDeactivate](#) (SP::UnitaryRelation ur, unsigned int i)
Overloading of predictorDeactivate.
- virtual bool [predictorActivate](#) (SP::UnitaryRelation ur, unsigned int i)
Overloading of predictorActivate.

5.7.1 Detailed Description

This class implements the time stepping with projection of a multi-bodies system. It inherits from Siconos::TimeSteppingProjectOnConstraints. It consists in update the CAD word during the simulation.

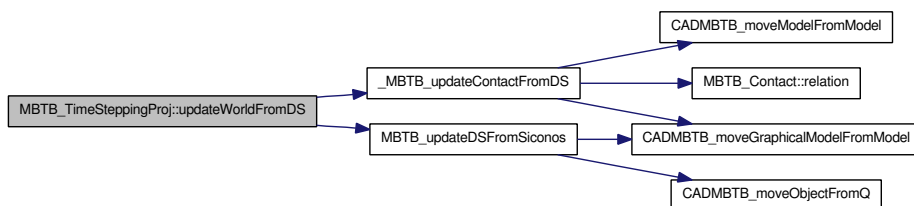
5.7.2 Member Function Documentation

5.7.2.1 void MBTB_TimeSteppingProj::updateWorldFromDS () [virtual]

Overloading of updateWorldFromDS.

It consists in updating the cad model from siconos.

Here is the call graph for this function:



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

- MBTB/MBTB_TimeSteppingProj.hpp
- MBTB/MBTB_TimeSteppingProj.cpp

Application Programming Interface of the geometrical module

O. Bonnefon

September 13, 2010

This document is an initial description of the geometrical module used in the SALADYN project. The goal is to define the API of this module and to define the functionalities waiting from OCC.

1 Project description

This section has to describe the global idea of the project. The name of the project is SALADYN. Partner are EDF, Schneider-Electric, LMGC, INRIA, LaMSID. Our goal is to simulate mechanical devices. We focus here about the geometrical aspects, two sub-projects are distinguished in the following subsections.

1.1 Using GEOM

In this case, the loading and the viewing is ensured by GEOM (included in SALOME). From GEOM (or from OCC?), we need:

- To get the OCC objects (surfaces, frames,...).
- To compute distances and points of contacts.
- To set and get positions of objects.
- To send a signal to GEOM to redraw the scene.
- To add simple graphical objects (lines and points) to see the interactions between the objects.

1.2 Using a simple viewer based on OCC

We also want to develop a sample viewer based on OCC (and Qt for example) to perform the mechanical simulation. Mainly it consists in using only OCC to load and explore the topology, to compute distance between objects and to update the scene during the simulation.

2 Application Programming Interface

This section has to describe the API used by the other modules (SICONOS-LMGC-SALADYN). Some of these functions will be implemented using OCC.

2.1 Detection collision

Get the bounding box of an object.

1. void getBoundingBox(in GeometricalModel, out Pinf, out Psup)

2.2 Punctual contact. Nonconforming contact

In the case of convex shapes, it consists in computing the couple of points where the minimal distance is reached, see Fig 1.

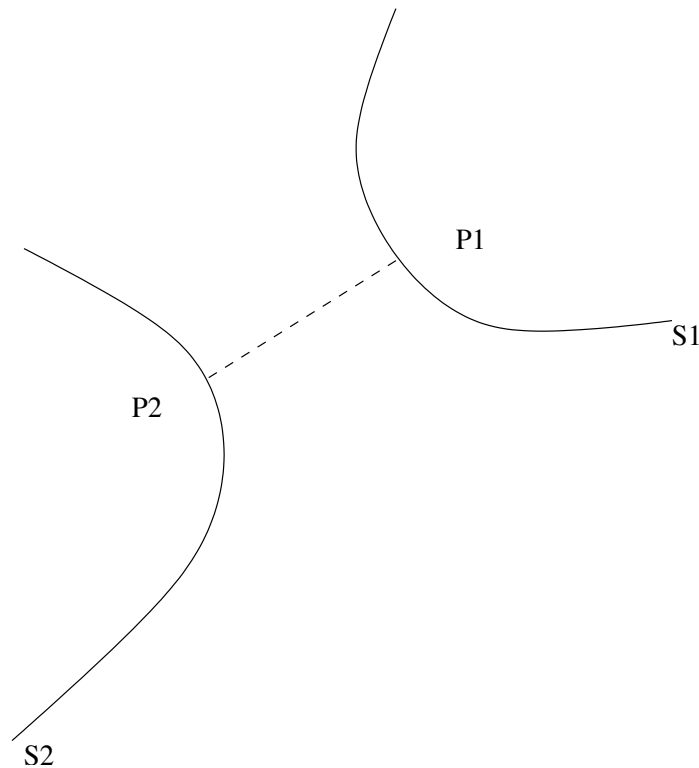


Figure 1: Simple case.

{fig:simpleCase}

In a more general case, for each part of two objects nearer than a given level, we need a couple of points, see Fig 2.

In the case of a unique point of contact, the API could be:

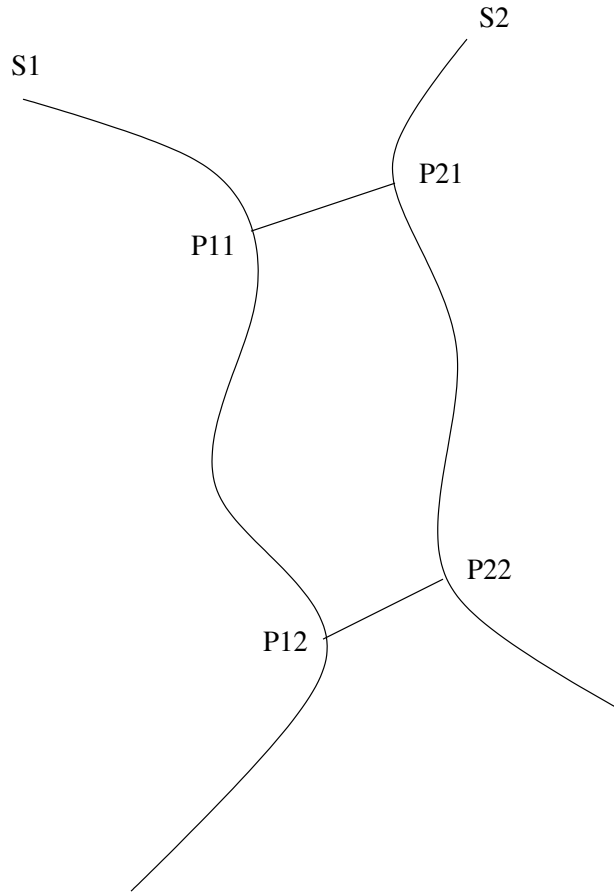


Figure 2: Multiple interaction.

{fig:MultiCase}

1. double getNearestPoint(in GeometricalModel G1,in GeometricalModel G1, out P1, out P2), returning the distance between points.

About the case of multi-contact the following API is proposed :

1. void computeContacts(in GeometricalModel G1,in GeometricalModel G2,in double dist)
2. int getContactCouple(out Point P1, out Point P2) return 1 while the list of punctual contact is not empty, else return 0.

Algorithm 1 Example 1

Require: G1 G2, Geometrical models.
computeContacts(G1,G2,0.1)
while getContactCouple(P1,P2) **do**
 Do what you want using P1, P2.
end while

{algo:example1}

2.3 Non punctual interaction. Conforming contact

In the case of a 2D interactions between objects, we distinguish two strategies: The first consists in computing the intersection (or an approximation), the second one is based on the computation of a common grid.

2.3.1 Compute the intersection

It consists in computing the geometrical intersection between two shapes. For example, the list of the closed-wire resulting of the intersection of two objects.

1. void computeWireContact(in GeometricalModel G1,in GeometricalModel G2)
2. int getWire(out Wire W), return 1 while the resulting list is not empty, else return 0. W is a wire (data structure that must be defined) resulting of the intersection computation.

Algorithm 2 Example 2

Require: G1,G2 Geometrical model.
computeWireContact(G1,G2)
while getWire(W) **do**
 Do what you want using W.
end while

{algo:example2}

2.3.2 compute a grid

It consists in computing a 2D grid separating two objects. The idea consists in representing the interactive zones between objects like a set of points.

1. void computeGrid(In GeometricalModel G1,In GeometricalModel G2, In double dist, In double gridTol ,....) where
 - (a) dist represents the thickness of the interactive zone.
 - (b) gridTol is the accuracy of the computed grid.
 - (c) ... must be defined, depending of the grid builder algorithm.
2. void getGridCouple(out P1, out p2, out id) return 1 while the resulting list is not empty, else return 0. where

- (a) P1 is a 3d point
- (b) P2 is a 3d point.
- (c) id identify a couple.

Algorithm 3 Example 3

Require: G1 G2 Geometrical model.
 computeGridContact(G1,G2,0.1,...)
while getCoupleGrid(P1,P2,id) **do**
 Do what you want.
end while

{algo:example3}

Question for LMGC and SICONOS: Is the grid plane ?

3 Functions waited from OpenCascade

This section lists the functionalities waited from OCC. These functions will be used to implement the geometrical module of SALADYN.

3.1 Basic needs

Following, the list of the essential functionalities:

- getBoudingBox(in TopoDS O,in P1,out P2): It computes the bounding box of an object.
- getNearestPoint(in TopoDS O1, in TopoDS O2, P1, P2): It computes the couple of points where the minimal distance is got.
- Set and get Position(TopoDS O1,in out Position): To set and get the position (and orientation) of an object.
- computeProjection(In TopoDS O1, in P1, out P2): It consists in projecting the 3D point P1 on the surfaces O1.
- How explore the topology after importing a step file?
- Build and delete basics geometrical objects.
- computeIntersection(TopoDS O1, TopoDS O2, out ListOfTopoDS_Wire): compute the list of wires resulting to the intersection of O1 and O2.
- In the case of qt application, redraw the scene.
- Get a normal at a point located on a surface.

3.2 Other functionalities

Following, the list of secondary functionalities.

- Offset a surface: Minkowski sum. (it could be a pre-computation done by GEOM)
- Get a couple of point for all interactive zones (see Fig. 2).

3.3 Questions

- Are there some 2D parameter represented a surface ?
- Is it possible to use a name or an identifier to exchange surfaces, wires or points between software?
- How is located a point on a surface?