MBSim - A Kind (Of) Introduction

by MBSim Development Team

10.06.2015

MBSim is a tool for the simulation of multibody systems. This manual gives an overview, explains the installation process and introduces easy examples.
1. Introduction

MBSim is a simulation tool to analyse the dynamic phenomenons of dynamical systems. Its root is the modeling of nonsmooth multibody systems. The mathematical background has been developed at the Institute of Applied Mechanics of the Technische Universität München. MBSim is developed further at the institute and by a private interest group. A summary concerning rigid body dynamics is given in the PhD thesis of Martin Förg [9] and the lecture notes [13]. The PhD thesis of Roland Zander [15] introduces the theory of flexible bodies. Ref. [16] shows an overview about the research at the institute. This reference includes simulation results of academic and industrial examples [5, 10, 12]. Extensions regarding hydraulics [14] and signal processing as well as parallelisation and cosimulation [7, 6] are unique in the field of nonsmooth dynamical systems.

1.1. Features

The MBSim environment consists of four programs, MBSim, OpenMBV, HDF5Serie and FMatVec.

FMatVec is a template-based matrix-vector library. Depending on the size of the matrices, calculations are either undertaken directly or provided by Lapack and Blas. One can also link to (parallel) Atlas or (parallel) Intel MKL. Reference counting is done either with boost::shared_array or with a built-in process.

HDF5Serie is a time series wrapper for HDF5. It offers also a plotting tool for graphical visualisations.

OpenMBV is a visualization tool for multibody simulation. It can handle large files with HDF5. Octave preprocessing, easy-to-handle XML, analytic curve representations and various interfaces with swig. The visualization is based on Coin.

MBSim is a tool for the simulation of dynamical systems. The kernel offers the simulation of multibody systems with impacts and friction. Surfaces may be described by Nurbs-interpolation. Different modules exist for control, hydraulics, flexible multibody systems with small and large deflections, electronics, powertrain applications and client-server simulations. Models can be written with C++ but also with XML. XML is also the basis for a GUI, which offers easy drag-and-drop modeling. An interface for FMI model export is available.

1.2. Objective

The goal of this introduction is to motivate the use of MBSim. It shows the installation of the necessary program parts, describes basically its components and program flow and gives some examples. You find it in the Download section of the MBSim webpage, where you can also find binary releases for Linux and
Windows. For more information, e.g. Doxygen description and current build status, visit the [Official Build System of the MBSim-Environment](#).

### 1.3. Acknowledgement

Many people have been helping to implement MBSim. Among them have been students and PhD students of the Institute of Applied Mechanics of the Technische Universität München. We mention the former PhD students, in alphabetical order, Mathias Bachmayer, Thomas Cebulla, Jan Clauberg, Bastian Esrefeld, Martin Förg, Markus Friedrich, Kilian Grundl, Robert Huber, Thorsten Schindler, Markus Schneider, Roland Zander.
2. Installation

This section summarizes the necessary steps to install MBSim.

2.1. Where to find the source code

The source code of MBSim together with some examples, the necessary FMatVec library, a HDF5 wrapper for output and the visualisation program OpenMBV can be found at https://github.com using git[1]. Everything is placed under LGPL[2].

2.2. Installation procedures

For the installation of the specific projects always the same procedures have to be applied. They are summarized in the following.

2.2.1. Installation

- AUTOMAKE:
  
  autoreconf -fi

- CONFIGURE:
  
  ./configure

  with defining a location for the installation

  --prefix=$HOME/.../Install

  possibly with project depending FLAGS which we find invoking

  --help

- INSTALL
  
  - make

  - make install

All procedures belong to the GNU-Build-System (cf. Sec. A.1).

2.2.2. Reinstallation/update

The procedure REINSTALL


2 see file COPYING in the root directory of the specific source code

5
make uninstall
make clean
./config.status --recheck
make install

newly installs a project with the same configure options used at the previous installation. These information are stored in the file config.status.
The procedure UPDATE is similar to the reinstallation:

make uninstall
make clean
git pull
./config.status --recheck
make install

For restoring a not-configured version of the project

make maintainer-clean

is used. After that configure has to be invoked.

### 2.2.3. Uninstallation

For uninstalling

make uninstall
make clean

has to be called in all directories. If the files created by configure should be deleted, we type

make distclean
Figure 1: Installation flow for MBSim
2.3. Installation of the simulation framework

It is assumed that a directory MBSim and a directory MBSim/Install have been created in the $HOME path of the Linux operating system. All projects depend on PKG package administration. The file $HOME/.bashrc has to be extended with

```
export PKG_CONFIG_PATH=
    "$HOME/MBSim/Install/lib/pkgconfig/:$PKG_CONFIG_PATH"
export LD_LIBRARY_PATH=
    "$HOME/MBSim/Install/lib/:$LD_LIBRARY_PATH"
```

2.3.1. FMatVec

It is assumed that boost is installed. For the installation the following instructions have to be completed:

```
cd $HOME/MBSim
git clone https://github.com/mbsim-env/fmatvec.git
```

Continue with the procedure AUTOMAKE.

```
mkdir build, cd build
```

The procedure CONFIGURE for dynamic compilation is used with

```
--prefix=$HOME/MBSim/Install
```

The code can be compiled and installed with a Doxygen HTML class documentation by make doc and the procedure INSTALL.

2.3.2. HDF5Serie

HDF5 is a hierarchical data format enabling the effective administration of plot and visualisation data. It can be downloaded as source code ("ALL Platforms") from [http://www.hdfgroup.org/HDF5/](http://www.hdfgroup.org/HDF5/) e.g. with version 1.8.15. Change to $HOME/MBSim/hdf5.

Use the procedure CONFIGURE for dynamic compilation with

```
--prefix=$HOME/MBSim/Install
```

and the additional FLAG

```
--enable-cxx
```

Compilation is done with the procedure INSTALL.
It is assumed that [MatVec] is installed. [HDF5Serie] is available by

```
cd $HOME/MBSim
git clone https://github.com/mbsim-env/hdf5serie.git
```

For having [MBSim] creating [HDF5] files invoke

```
cd $HOME/MBSim/HDF5Serie/hdf5serie
```
as well as the procedure `AUTOMAKE`.

```
mkdir build, cd build
Continue with `configure` for dynamic compilation with
--prefix=$HOME/MBSim/Install
make doc and `install` for installation and creation of a Doxygen HTML class
documentation.
Last, `.bashrc` can be extended with

```
alias h5lsserie"$HOME/MBSim/Install/bin/h5lsserie"
alias h5dumpserie"$HOME/MBSim/Install/bin/h5dumpserie"
```
to gain overall access to the commands `h5lsserie` and `h5dumpserie`.

### 2.3.3. OpenMBV

The installation for the simulation framework consists of three steps: first the
XMLUtils have to be installed, then [MBSim] needs OpenMBV-C++Interface to
create standard data for [OpenMBV] using C++ programs. The source code is
available by

```
cd $HOME/MBSim
git clone https://github.com/mbsim-env/openmbv.git
```

**XMLUtils** It is assumed that [MatVec], Octave, swig, Xerces and Xalan in-

```
cd $HOME/MBSim/OpenMBV/mbxmutils
```
and use the procedure `AUTOMAKE`.

```
mkdir build, cd build
Continue with `configure` for dynamic compilation with
--prefix=$HOME/MBSim/Install
and `install` for installation of an independent XML preprocessor to parse and
validate hierarchical XML-files.
**OpenMBV-C++Interface** It is assumed that HDF5Serie and XMLUtils are installed. Invoke

```
cd $HOME/MBSim/OpenMBV/openmbvcppinterface
```

and the procedure AUTOMAKE.

```
mkdir build, cd build
```

Continue with `CONFIGURE` for dynamic compilation with

```
--prefix=$HOME/MBSim/Install
```

Sometimes trouble with linking *swig* occurs; in this case just set

```
--with-swigpath=/nope
```

to some value such that *swig* is not found on your system.

**make doc** and **INSTALL** for installation and creation of a Doxygen HTML class documentation.

### 2.3.4. **MBSim**

Necessary for the installation of **MBSim** is OpenMBV-C++-Interface. For installation, one types

```
cd $HOME/MBSim
```

```
git clone https://github.com/mbsim-env/mbsim.git
```

**NURBS thirdparty package** If you like, you can install the NURBS thirdparty package first, which is necessary for some examples. Invoke the procedure AUTOMAKE.

```
mkdir build, cd build
```

Continue with `CONFIGURE` for dynamic compilation with the prefix

```
--prefix=$HOME/MBSim/Install
```

in `$HOME/MBSim/mbsim/thirdparty/nurbs++`.

**MBSim kernel** Then proceed and invoke the procedure AUTOMAKE.

```
mkdir build, cd build
```

Continue with `CONFIGURE` for dynamic compilation with

```
--prefix=$HOME/MBSim/Install
```

in `$HOME/MBSim/mbsim/kernel`.

**make doc** and **INSTALL** to install the basic module and to create a Doxygen HTML class documentation. In

```
$HOME/MBSim/mbsim/kernel/xmldoc
```

**invoke INSTALL** for an XML documentation in

```
$HOME/MBSim/Install/share/mbxmlutils/doc
```
**Modules**  The following modules are available in MBSim:

- mbsimControl
- mbsimHydraulics
- mbsimFlexibleBody
- mbsimElectronics
- mbsimPowerTrain
- mbsimInterface

The installation proceeds as follows:

```
cd $HOME/MBSim/mbsim/modules/mbsimControl
```

Invoke the procedure `AUTOMAKE`.

```
mkdir build, cd build
```

Continue with `CONFIGURE` for dynamic compilation with

```
--prefix=$HOME/MBSim/Install
```

make doc and `INSTALL` to install the signal processing and control module and to create a Doxygen HTML class documentation. In

```
$HOME/MBSim/mbsim/modules/mbsimControl/xmldoc
```

invoke `INSTALL` for an XML documentation in

```
$HOME/MBSim/Install/share/mbxmlutils/doc
```

Proceed in the same way for the other modules in the order as given above.

**MBSimXML**  MBSimXML offers the possibility to define mechanical systems with XML.

```
cd $HOME/MBSim/mbsim/mbsimxml
```

Invoke the procedure `AUTOMAKE`.

```
mkdir build, cd build
```

Continue with `CONFIGURE` for dynamic compilation with

```
--prefix=$HOME/MBSim/Install
```

and `INSTALL` to install the XML module which contains an executable to invoke the preprocessor. In

```
$HOME/MBSim/mbsim/mbsimxml/xmldoc
```

invoke `INSTALL` for an XML documentation in

```
$HOME/MBSim/Install/share/mbxmlutils/doc
```

11
MBSimFMI  MBSimFMI gives an interface for model export. Proceed as follows:

```bash
cd $HOME/MBSim/mbsim/mbsimfmi
```
Invoke the procedure `AUTOMAKE`.

```bash
mkdir build, cd build
```
Continue with `CONFIGURE` for dynamic compilation with

```bash
--prefix=$HOME/MBSim/Install
```
and `INSTALL`.

MBSimGUI  MBSimGUI offers a GUI for [MBSim](#). If we have installed [OpenMBV](#), we can install it.

```bash
cd $HOME/MBSim/mbsim/mbsimgui
```
Invoke the procedure `AUTOMAKE`.

```bash
mkdir build, cd build
```
Continue with `CONFIGURE` for dynamic compilation with

```bash
--prefix=$HOME/MBSim/Install
```
and `INSTALL`.

### 2.3.5. Examples

The examples are used for testing successful installation. There are two possibilities:

1. Change to the specific directory $HOME/MBSim/mbsim/examples/* and type `make` to create an executable. The simulation starts with the command `./main`. The results are visualised with the command `openmbv` and plotted with `h5plotserie` after having installed the visualisation framework (Sec. 2.4 and 3.3).

2. Use the script `python3 runexamples.py` in $HOME/MBSim/mbsim/examples to compile, run and test each example. See `python3 runexamples.py --help` for additional information.

### 2.4. Installation of the visualisation framework

It is assumed, that a directory [OpenMBV](#) and a directory [OpenMBV/Install](#) has been created in the `$HOME` path of the Linux operating system. The necessary software is described in Sec. 2.4. This subsection describes a static compilation, therefore the additional FLAG have to be used in each step

```bash
--disable-shared --enable-static
```
2.4.1. **HDF5**

Install [HDF5](#) and the [HDF5Serie](#) as described in Sec. [2.3](#) but in the directory OpenMBV and using a static compilation. For plotting of [HDF5](#) files it is assumed that Qwt with version 5 is installed. Invoke

```bash
cd $HOME/OpenMBV/HDF5Serie/h5plotserie
```
as well as the procedures AUTOMAKE.

```bash
mkdir build, cd build
```
Continue with configure for static compilation, make doc and INSTALL for installation and creation of a Doxygen HTML class documentation. `.bashrc` can be extended with

```bash
alias h5plotserie="$HOME/OpenMBV/Install/bin/h5plotserie"
```
to gain overall access to the command h5plotserie.

2.4.2. **OpenMBV**

**XML Utils**  Install XML Utils as described in Sec. [2.3](#) but in the directory OpenMBV and using a static compilation.

**OpenMBV-C++Interface**  Install OpenMBV-C++Interface as described in Sec. [2.3](#) but in the directory OpenMBV and using a static compilation.

**OpenMBV**  For the installation of a static visualisation using always the newest source files it is assumed that Coin3d, MBXMLUtils, [HDF5Serie](#), SoQt, Qwt with version 5 are installed. Use

```bash
cd $HOME/OpenMBV/OpenMBV/openmbv
```
and the procedure AUTOMAKE

```bash
mkdir build, cd build
```
Continue with configure for static compilation, make doc and install complete the installation of the viewer with an Doxygen HTML class documentation. `.bashrc` can be extended with

```bash
alias openmbv="$HOME/OpenMBV/Install/bin/openmbv"
```
to gain overall access to the command openmbv.
3. **MBSim**- Program Overview

From the software development point of view, [11] proposed a standard structure for multibody simulation frameworks distinguishing between bodies and interactions. The programs described in [2] and [3] follow also this approach. It is approved and used in **MBSim** using object-oriented C++ programming. The interface of all classes is documented in the source code using Doxygen. This documentation has been extracted for convenient study together with a class overview during installation with the command `make doc`. Just change to `~/.MBSim/*/doc/html` and open `index.html` with an appropriate browser. With this class documentation and self-explaining names in the user interface it is goal of this section to give an overview about the main features of **MBSim**.

Figure 3 shows the embedding of **MBSim** in the global simulation and analysing process. **MBSim** can handle a set of dynamical systems from various domains not only separated but also exchanging data of the exemplary smooth form

\[
\begin{align*}
q &= Y u, \\
M \dot{u} &= h(q, u, t) + W \lambda, \\
\dot{x} &= f(x), \\
(q, u, \lambda, t) &\in \mathcal{N}.
\end{align*}
\]

Though, the simulation of hydraulics, electronics, control and power train systems is included within several modules. **MBSim** is based on the interface **FMatVec** using either LaPack\(^3\) or ATLAS\(^4\) for fast evaluation of linear algebra routines. Further, with **HDF5Serie** it writes simulation result files in the hierarchical HDF5 file format\(^5\) even for large dynamical systems. These files can be read by **H5PlotSerie** for plotting or by **OpenMBV** for visualisation. Thereby, **OpenMBV** is based on the Coin implementation\(^6\) of the Open Inventor Library\(^7\). Also a co-simulation with Matlab/Simulink\(^8\), **HySim**\(^4\) for hydraulic components and **KetSim**\(^8\) for camshaft timing chains is possible \(^7\). **MBSim** is divided in a modelling part using C++ or XML and a simulation part. The simulation part is implemented quite modular distinguishing between the update of bodies and interactions concerning kinematics, kinetics as well as force laws and integration / nonlinear solution schemes. Also here external libraries are used where it is possible for having always a state-of-the-art numerical basis. A typical example of a dynamical system only from mechanics is given.

---

1. cf. [http://www.netlib.org/lapack/](http://www.netlib.org/lapack/)
by Figure 2. One has the environment and two Objects, in mechanics called "Body"s, holding the inertia terms. At a Body both Contours and Frames can be attached. Then, interconnecting Links like frictional Contacts between Contours and like ideal Joints between body Frames can be defined. External KineticExcitations act on body Frames. These ingredients and a schedule of the simulation are basically explained in the following from the mechanical point of view.

3.1. Description of the Components

The following classes can be used for modelling and simulating dynamical systems in MBSim.

3.1.1. DynamicSystem and DynamicSystemSolver

Hierarchically Objects and Links belong to DynamicSystems being explained in the following (cf. Fig. 3). Shaded one can see interfaces for the different components of the e.g. smooth modelling equations

- ExtraDynamicInterface

\[ \dot{x} = f(x) \]  

- ObjectInterface

\[ \dot{q} = Y u, \]  

\[ M(q) \ddot{u} = h(q, u, t), \quad M \text{ symmetric positive definite} \]
LinkInterface

\[(q, u, \lambda, t) \in \mathcal{N}, \quad (4)\]

\[h(q, u, t), W(q)\lambda \quad (5)\]

and Element for a load/save mechanism as well as plot and data administration. Class Graph is dotted as it is not available for modelling. A graph structure is automatically built during initialisation for an efficient simulation process evaluating the hierarchical modelling structure. Though, DynamicSystems can be hierarchically assembled in Groups. The top-most DynamicSystem is called DynamicSystemSolver. It also represents the interface to the integration schemes and allows the setting of environment variables like gravitation.

Important settings:

- addObject
  add an Object, e.g. a Body from mechanics

- addLink
  add a Link, e.g. a Contact or a Joint from mechanics

- addGroup
  add a DynamicSystem

- addFrame
  add a Frame

- addContour
  add a Contour

- readz0
  read initial state from HDF5 file

- writez
  write current state to HDF5 file

- setReorganizeHierarchy
  enforces the automatic creation of invisible graph structures only for simulation depending on the hierarchy of frames used for modelling

- setConstraintSolver
  solver for constraint equations on acceleration level (cf. setImpactSolver)

- setImpactSolver
  solver for constraint equations on velocity level; available constraint equation solution schemes are
    - LinearEquations
    - Cholesky solution scheme for regular linear systems (only bilateral constraints)
GaussSeidel
Gauss-Seidel solution scheme for piecewise linear systems (planar Coulomb friction)

FixedPointSingle
Gauss-Seidel solution scheme with fixed point search and relaxation strategy (spatial Coulomb friction)

RootFinding
damped and globalised Newton scheme (spatial Coulomb friction)

setLinAlg
linear equations solver for RootFinding; available solvers are

  LUDeocomposition
  LU decomposition is only valid for non-singular contact situations

  LevenbergMarquardt
  regularisation of contact matrix by Levenberg Marquardt factor

  PseudoInverse
  selecting the norm-minimal solution if there exist several

setStrategy
set relaxation strategy to local (contact wise) or global (all contacts)

setNumJacProj

setMaxIter
set the maximum number of iterations in the contact solver

setStopIfNoConvergence
the integrator stops integration, if there is no convergence in the contact solver

3.1.2. Frames

Frames are a basic concept in the kernel of MBSim to define an interface for multibody components concerning kinematic and kinetic expressions. They specify a unique point including position and orientation. New Frames can be added arbitrarily and recursively based on a predecessor Frame. That is why there has to be a first Frame for specific MBSim components, e.g. each DynamicSystem has got a stationary Frame "I" (inertial frame).

Important settings next to position and Cartesian orientation are defined according to the parametrisation

$$
\begin{pmatrix}
  J_a^P \\
  J^P
\end{pmatrix}
= \begin{pmatrix}
  J^P \\
  J^R
\end{pmatrix}
\dot{u} + \begin{pmatrix}
  J^P \\
  J^R
\end{pmatrix}
$$
of translational and angular acceleration. This is basically an affine relationship concerning the derivative of generalised velocities \( \dot{u} \), appropriate JACOBIAN matrices \( J \) and an additional summand \( \iota \) with gyroscopic and explicit time dependent contributions. Summarizing

```plaintext
ggetPosition
ggetOrientation
ggetVelocity
ggetAngularVelocity
ggetJacobianOfTranslation
ggetJacobianOfRotation
ggetGyroscopicAccelerationOfTranslation
ggetGyroscopicAccelerationOfRotation
enableOpenMBV
```

are important settings.

### 3.1.3. Bodies

Mechanical Bodys provide their portion of a positive definite mass matrix, a smooth right hand side, state and energy expressions according to Object, which is e.g. similar for some hydraulic objects, like pipes (cf. Figure 4). One difference is the kinematic description which is based on Frames in the case of mechanical Bodys. Also the connectors to other Bodys might follow other structural rules. Depending on the type of linkage, Frames or Contours occur in the case of Bodys. One distinguished between rigid and flexible bodies.

**Important settings:**

```plaintext
addFrame
add a Frame

addContour
add a Contour
```

**Rigid Bodies** For each RigidBody a Frame "C" in the centre of gravity is pre-defined. One perhaps newly created Frame of the RigidBody has to be chosen as frame for kinematics "K" with centre \( P \) and one Frame of another Body or a DynamicSystem has to be chosen as frame of reference "R" with centre \( O \). Both absolute –if the frame of reference belongs to a DynamicSystem– and relative
–if the frame of reference belongs to another RigidBody– kinematic structures are canonically given by the frame recursion. The motion of the frame for kinematics and so also of the RigidBody with respect to the frame of reference is defined by the individual generalised coordinates $q_{rel}$ of the RigidBody or by a time-dependent path. These settings can be defined individually on position, velocity and acceleration level according to

$$R_{OP} = R_{OP}(q_{rel}, t),$$

$$A_{RK} = A_{RK}(q_{rel}, t),$$

$$R_{vOP,rel} = R_{J,rel} u_{rel} + R_{T,rel},$$

$$R_{wRK} = R_{J,R,rel} u_{rel} + R_{\iota,rel},$$

$$\frac{d}{dt}(R_{v OP,rel}) = R_{J,T,rel} u_{rel} + \frac{d}{dt}(R_{J,T,rel}) u_{rel} + \frac{d}{dt}(R_{T,rel}),$$

$$\frac{d}{dt}(R_{w RK}) = R_{J,R,rel} \dot{u}_{rel} + \frac{d}{dt}(R_{J,R,rel}) u_{rel} + \frac{d}{dt}(R_{\iota,R,rel}).$$

In the referencing coordinate frame, Eq. (6) describes the translational position of the frame for kinematics and Eq. (7) its orientation. Then, the velocity level is given by an affin relation involving JACOBIAN matrices $J$, generalised velocities $u_{rel}$ and explicit time-dependent summands $\iota$. The acceleration level can be obtained by differentiation. Figure 5 shows the Frame recursion also including the link relationships of frames and contours. In a DynamicSystem two stationary Frames "I" and "I" are defined as frames of reference of the lower right or left RigidBody in an absolute parametrisation, respectively. For the left RigidBody the frame for kinematics is given by Frame "B", whereas for the lower right RigidBody the canonic Frame "C" in the centre of gravity is used. The lower right RigidBody is also the reference of another RigidBody altogether yielding a relative parametrisation. These two RigidBodys are Linked concerning their "B"-Frames as well. For completing the description, two additional Links between dotted contours are inserted. The drawback of this general description is a time-dependent mass-matrix also in the absolute kinematics case.

Important settings:

- setMass
- setInertiaTensor
  is defined with respect to "C", if no other Frame is given as second argument
- setFrameOfReference
  one possibility to define initial values
- setFrameForKinematics
Kinematics can be defined individually using the member functions

- `setTranslation`
- `setRotation`
- `setJacobianOfTranslation`
- `setJacobianOfRotation`
- `setDerivativeOfJacobianOfTranslation`
- `setDerivativeOfJacobianOfRotation`
- `setGuidingVelocityOfTranslation`
- `setGuidingVelocityOfRotation`
- `setDerivativeOfGuidingVelocityOfTranslation`
- `setDerivativeOfGuidingVelocityOfRotation`

For convenience it is sometimes not necessary to define all components. E.g. for a LinearTranslation as well as RotationAboutFixedAxis and CardanAngles everything else is derived automatically.

**Flexible Bodies** The equations of motion of a FlexibleBody is at the moment only available with respect to a stationary Frame. So, for flexible bodies the frame of reference must belong to a DynamicSystem. The following flexible bodies are available.

- **FlexibleBody1s21RCM**
  - planar beam using redundant coordinate method with three coordinates per finite element node, translation $x$, $y$ and rotation $\gamma$, as well as two additional bending deflections $c_1$, $c_2$

- **FlexibleBody1s33RCM**
  - spatial beam using redundant coordinate method with six coordinates per finite element node, translation $x$, $y$, $z$ and reversed Cardan rotation $\alpha$, $\beta$, $\gamma$, as well as four additional bending deflections $c_1$, $c_2$, $c_3$, $c_4$
3.1.4. LinkMechanics

LinkMechanics represents interconnections between mechanical bodies according to Figure 6 using the connectors of Bodies. If other connectors are used, other link classes have to be inherited. Links distribute locally to \( h \) and \( W \lambda \) in the equations of motion.

Important settings:

- `connect` connect mechanical links to Frames
- `setOpenMBVForceArrow` visualisation of link forces in OpenMBV
- `setOpenMBVMomentArrow` visualisation of link torques in OpenMBV

**SpringDamper**  A SpringDamper connects two Frames using a predefined spring force function.

Important settings:

- `setForceFunction`
- `setProjectionDirection` projection direction of the force, if it should not act in direction of the Frames’ connecting vector
- `setOpenMBVSpring` visualisation of a spring in OpenMBV

**KineticExcitation**  A KineticExcitation is connected to one Frame using a predefined excitation function.

Important settings:

- `setForce` force excitation function
- `setMoment` moment excitation function
- `setFrameOfReference` force / moment direction vectors, if not the connected Frame should be used
Joints  Joints connect two Frames with the force laws depending on the ideal normal relative kinematics. The constitutive law has to be chosen for the calculation of the force parameter.

Important settings:

- setForceDirection
  constraint force directions

- setMomentDirection
  constraint moment directions

- setForceLaw
  constitutive law on acceleration level

- setImpactForceLaw
  constitutive law on velocity level

Contacts and Impacts  Contacts and impacts are managed by the class Contact.

Important settings:

- setContactForceLaw
  constitutive normal law on acceleration level

- setContactImpactLaw
  constitutive normal law on velocity level

- setFrictionForceLaw
  constitutive friction law on acceleration level

- setFrictionImpactLaw
  constitutive friction law on velocity level

- setContactKinematics
  The relative kinematics is defined between Contour classes. On velocity level the contact kinematics is independent of the specific contour. For the calculations on position level the following contours are available.

  CircleHollow
  one dimensional sphere with contact from inside

  CircleSolid
  one dimensional sphere with contact from outside

  FlexibleBand
  flexible contour describing a band in a certain distance and direction of a neutral fibre

  Frustum
  frustum with its axis given by the second column of the contour reference frame
Line
affine one dimensional space
Plane
affine two dimensional surface
Point
most primitive rigid contour
Sphere
two dimensional sphere

Each contour has a contour fixed frame, with the third column of the orientation matrix being the binormal for planar contours and the first column being the normal of linear contours. Sometimes they can be visualised with enableOpenMBV.

Available contact kinematics on position level:
CircleFrustum
CircleSolidLine
PointLine
PointFrustum
PointPlane
PointFlexibleBand
SphereFrustum
SpherePlane

enableOpenMBVContactPoints
enables the visualisation of accompanying contact frames in OpenMBV

setOpenMBVNormalForceArrow
visualisation of the normal force arrow in OpenMBV

setOpenMBVFricctionArrow
visualisation of the friction force arrow in OpenMBV

**Constitutive Laws** Concerning the constitutive laws it is distinguished between contact laws on acceleration and impact laws on velocity level. Further, both flexible and rigid laws in normal and tangential direction are available.

Important possibilities:
UnilateralConstraint
set-valued on acceleration level

---

9 Modelling hint: There are contradictions between energy conservation in normal direction and dissipation due to friction when combining these features.
BilateralConstraint
set-valued on acceleration level

UnilateralNewtonImpact
set-valued on velocity level

BilateralImpact
set-valued on velocity level

PlanarCoulombFriction
set-valued on acceleration level

SpatialCoulombFriction
set-valued on acceleration level

PlanarStribeckFriction
set-valued on acceleration level

SpatialStribeckFriction
set-valued on acceleration level

PlanarCoulombImpact
set-valued on velocity level

SpatialCoulombImpact
set-valued on velocity level

PlanarStribeckImpact
set-valued on velocity level

SpatialStribeckImpact
set-valued on velocity level

RegularizedUnilateralConstraint
single-valued on acceleration level

RegularizedBilateralConstraint
single-valued on acceleration level

RegularizedPlanarFriction
single-valued on acceleration level

RegularizedSpatialFriction
single-valued on acceleration level
Conventions  For modelling own contact kinematics and constitutive laws some conventions are important.

- contact kinematics
  `update g` should define the normal distance, the possible contact locations and trihedral orientations, `update w b` are nonlinear kinematic terms on acceleration level

- accompanying contour trihedral
  the first column is the outward pointing normal, then the two tangentials follow with only the first tangent-pair having opposite sign

3.1.5. Functions

Depending on the number of arguments it is possible to derive new functions from `Function1`, `Function2` and `Function3` by specifying the template parameters. They are used at various places and are a general way of re-using functional descriptions.

3.1.6. Integration Schemes

Available integration schemes:

- `DOPRI5Integrator`
  Dormand-Prince one-step integration scheme of order 5 for nonstiff ODE with step size control

- `RADAU5Integrator`
  one-step integration scheme of order 5 for stiff ODE with step size control

- `TimeSteppingIntegrator`
  one-step semi-implicit integration scheme of order 1 for nonstiff MDE

3.2. Program Flow

Conceptionally the program flow is defined by the election of the integration scheme. It can always be stopped using `Ctrl-C` also enforcing the closing of the plot functionality. With

`kill -USR2 <PID of simulation thread>`

a flush of the plot routine is asked for.
3.2.1. Timestepping Integration

Timestepping integration solves the whole equations of the system including the contacts on velocity level with fixed time step size. In detail one has the following work flow.

1. \( \text{DS::plot}\left(t, q, u\right) \)

2. \( q \leftarrow q + \text{DS::deltaq}\left(t, q, u\right) \)

3. \( t \leftarrow t + \Delta t \)

4. \( \text{DS::update}\left(t, q, u\right) \)

   \( \text{DS::updateStateDependentVariables} \)
   update variables depending on the generalised state and the structure of the system with one independent group and several recursive calculation graphs

\( \text{DS::updateg} \)

   - update of the relative position kinematics independent of the system structure using the order
     \[
     \text{Link} \rightarrow \text{LinkMechanics} \rightarrow \text{ContactKinematics}
     \]

   - several contacts points are possible from the kinematical point of view, whereby the maximum number is calculated in \text{ContactKinematics}

\( \text{DS::checkActiveg} \)

   - determine the state of the relative kinematics concerning the activity of links
   - redefine global memory references using indices

\( \text{DS::updatedg} \)

   - update of the relative velocity kinematics independent of the system structure
   - can be done in the child classes of \text{LinkMechanics}

\( \text{DS::updateT} \)

updates the linear transformation matrix \( \dot{q} = Tu \) independent of the system structure

\( \text{updateJacobians} \)

updates the JACOBIANS for projecting forces in generalised directions dependent on the system structure
updateh
updates the right hand sides with the possibility to account for internal forces of objects and external forces of links independent of the system structure

updateM
updates the mass matrix independent of the system structure

dacLLM
- computes the CHOLESKY decomposition of the mass matrix dependent on the system structure
- group calculates the matrix inverse locally per object
- graph calculates the matrix inverse globally

updateW
updates the JACOBIAN between in general set-valued link-force parameters and generalised coordinates

updateV
- the decomposition of the in general set-valued link-forces
  \[ W\lambda = W_N\lambda_N + W_T\lambda_T \]
  in a normal and tangential part allows to separate the single-valued slip case
- for affected links it is
  \[ \ddot{W}\lambda = (\ddot{W}_N + \mu\ddot{W}_T)\lambda_N = \dot{V}\lambda_N \]
  altogether this is a reduction of the set-valued equations being expressed by the projection
  \[ V\lambda^* \]

updateG
- the force action matrix
  \[ G = W^TM^{-1}V \]
  must be calculated by the most global view, namely the DynamicSystemSolver
- the size of \( G \) is reduced due to the introduction of \( V \) but is non-symmetric
- for a time-stepping scheme it is \( V = W \)

5. DS::solveImpacts(t, q, u)
• the constrained equations are solved on velocity level using sparse matrix structures (cf. MKL sparse matrix storage format)
• block structures are not evaluated

6. \( u \leftarrow u + DS::\text{deltau} \)
7. \( x \leftarrow x + DS::\text{deltax} \)
8. \( DS::\text{projectGeneralizedPositions} \)

3.2.2. Event-Driven Integration

Currently, \texttt{LSODAR} is the only event-driven integrator with automatic switch between stiff and non-stiff equations.

1. \( DS::\text{computeInitialCondition} \)
   checks for system configuration and creates the necessary contact container
2. \( DS::\text{plot}(t,q) \)
3. \( DS::\text{plot}(t,q) \)

\( DS::\text{zdot}(t,q,u) \) is available for standard and inverse kinetics calculations

• \texttt{wb} means \( \ddot{w} \) and describes the acceleration terms in the constraint kinematics
• \texttt{computeConstraintForces} uses a least square algorithm to solve the Delassus equations, assume \( Ax = b \) with a \( m \times n \) full-rank matrix \( A \), then there are two cases
  - \( m \geq n \) (skinny) can always be solved by \( \|Ax - b\| \rightarrow \text{min} \) and so by SVD
    analytically the solution is given by the normal equations \( x = (A^T A)^{-1} A^T b \)
  - \( m < n \) (fat) has an infinite dimensional solution space, one has to pick one solution
    \( \|x\| \rightarrow \text{min}, Ax = b \) which is analytically given by \( x = A^T (A^T A)^{-1} b \),
    again numerically a SVD solves the problem most efficiently

\( DS::\text{getsv} \) the stop vector defines the root function concerning contacts and stick-slip-transitions for the DAE solver

• it can be only set by \texttt{Link}
• contains kinematics for not-active directions and kinetics for active directions
• the last entry is used for position and velocity projections
• is solved by integrator concerning a tolerance

4. DS::shift is invoked, if there is a sign change in the stop vector
   • drift compensation if indicated by stop vector
   • project to slightly positive gaps to avoid instantaneous appearance of new shift point
   • updateCondition should impact or differential equations be solved, after earlier mentioned reconfiguring?
• case studies
  • impact has highest priority and changes overall configuration
  • impact requires D::checkAllgd because of possible slip-stick transition
    - no difference between $\Lambda$ and $\lambda$
    - gdn means $\dot{g}^+$
      • impact involves new configuration and so also the equations of motion have to be solved
    - checkActivegdd has to be done with the same tolerance like in the nonlinear equations solver
  • gActive means a contact is closed
  • gdActive means a contact remains closed

3.3. Plot Routines

3.3.1. Usage

The result of a simulation with MBSim are a mbsh5 file for plot analysis as well as ombvh5 and ombvx files for visualisation. Additionally there is information concerning the integrator in *.plt and *.sum files; for visualisation *.iv files might appear.

For getting data from MBSim a HDF5 wrapper is used. Plotting the multibody system data can be done with

```
h5plotserie <h5-file>
```

The usage is quite canonic and documented in the online help. Interesting features are

• superimposing graphs by <shift> and left click in the data list.
• change axes by <ctrl> and left click in the curve list.
• disabling graphs by right click in the curve list.

29
With

    h5lsserie <h5-file>

basically the content can be shown. Several possible options are explained by typing -h:

    -d shows the description of the data to plot
    -l shows the column labels of the data to plot
    -f follows external links in a set of HDF5 files to avoid redundant data (HDF5 is possible per DynamicSystem)

The specific names in HDF5 format are specialised by reading from right to left. The url to specific data is given by a path and can be used in

    h5dumpserie <path>

perhaps also in other tools. The column one is interested in is declared using a colon. Also several columns can be appended, whereby shorter ones are enlarged by nan entries. Altogether, it is possible to use the dump by

    gnuplot "<dump" u *:* w l

or in MatLab by

    h5dump(’<path>’)

OpenMBV files can be opened by just typing openmbv in the specific directory. The usage should be quite canonic and is explained in the online help.

3.3.2. Implementation

In MBSim plotting is done using plotFeatures being defined in element.h. They are set in dynamic_system_solver.cc. One plot-file comprises time-series of rowvectors with the same data type in all entries.
Figure 3: Dynamic system type classes in MBSim (UML).
Figure 4: Object type classes in MBSim (UML).

Figure 5: Rigid kinematics in MBSim.
Figure 6: Link type classes in \texttt{MBSim} (UML).
4. Examples

4.1. Example: One-Mass-Oscillator With Recursive Structure and Impact

4.1.1. Physics - system

```cpp
#ifndef ONEMASSOSCILLATOR_H
#define ONEMASSOSCILLATOR_H

#include "mbsim/dynamic_system_solver.h"
#include <string>

class System : public MBSim::DynamicSystemSolver {
    public:
        System(const std::string &projectName);
    }
#endif
```

1,2,13 avoids including Header more than once
4,5 includes
7 class definition for one mass oscillator, which is derived by
9 constructor

```cpp
#include "system.h"
#include "mbsim/rigid_body.h"
#include "mbsim/spring_damper.h"
#include "mbsim/environment.h"
#include "mbsim/contours/sphere.h"
#include "mbsim/contact.h"
#include "mbsim/constitutive_laws.h"

#include "openmbvcppinterface/coilspring.h"

using namespace MBSim;
using namespace fmatvec;
using namespace std;

System::System(const string &projectName) : DynamicSystemSolver(projectName) {
    // acceleration of gravity
```
Vec grav(3);
grav(1) = -9.81;
MBSimEnvironment::getInstance() -> setAccelerationOfGravity(grav);

// frames on environment
this -> addFrame("L", Vec(3, INIT, 1.), SqrMat(3, EYE));

// bodies
RigidBody *mass1 = new RigidBody("Mass1");
RigidBody *mass2 = new RigidBody("Mass2");

// attributes
mass1 -> setMass(1.);
mass1 -> setInertiaTensor(SymMat(3, EYE));
mass1 -> setTranslation(new LinearTranslation("[0.58; 0.58; 0.58]"));
mass1 -> setFrameOfReference(getFrame("L"));
mass1 -> setFrameForKinematics(mass1 -> getFrame("C"));
mass2 -> setMass(2.);
mass2 -> setInertiaTensor(SymMat(3, EYE));
mass2 -> setTranslation(new LinearTranslation("[0.58; 0.58; 0.58]"));
mass2 -> setFrameOfReference(mass1 -> getFrame("C"));
mass2 -> setFrameForKinematics(mass2 -> getFrame("C"));
mass2 -> setInitialGeneralizedPosition(-nrm2(Vec(3, INIT, 1)));

// add body to dynamical system
this -> addObject(mass1);
this -> addObject(mass2);

// spring
SpringDamper *spring1 = new SpringDamper("Spring1");
spring1 -> setForceFunction(new LinearSpringDamperForce(1, 1, 0));
spring1 -> connect(mass1 -> getFrame("C"), this -> getFrame("I"));
SpringDamper *spring2 = new SpringDamper("Spring2");
spring2 -> setForceFunction(new LinearSpringDamperForce(100, 1, 0));
spring2 -> connect(mass2 -> getFrame("C"), this -> getFrame("I"));

// add spring to dynamical system
this -> addLink(spring1);
this -> addLink(spring2);

// contact
Sphere *sphere1 = new Sphere("Sphere1");
sphere1 -> setRadius(0.2);
sphere1 -> enableOpenMBV();
mass1->addContour(sphere1, Vec(3, INIT, 0.), SqrMat(3, EYE));
Sphere *sphere2 = new Sphere("Sphere2");
sphere2->setRadius(0.2);
sphere2->enableOpenMBV();
mass2->addContour(sphere2, Vec(3, INIT, 0.), SqrMat(3, EYE));
Contact *contact = new Contact("Contact");
contact->connect(sphere1, sphere2);
contact->setContactForceLaw(new UnilateralConstraint());
contact->setContactImpactLaw(new UnilateralNewtonImpact(0.3));
this->addLink(contact);

// visualisation
OpenMBV::CoilSpring* openMBVspring1=new OpenMBV::CoilSpring;
openMBVspring1->setSpringRadius(0.1);
openMBVspring1->setCrossSectionRadius(0.01);
openMBVspring1->setNumberOfCoils(5);
spring1->setOpenMBVSpring(openMBVspring1);

OpenMBV::CoilSpring* openMBVspring2=new OpenMBV::CoilSpring;
openMBVspring2->setSpringRadius(0.1);
openMBVspring2->setCrossSectionRadius(0.01);
openMBVspring2->setNumberOfCoils(5);
spring2->setOpenMBVSpring(openMBVspring2);

4.1.2. Time integration – main

#include "system.h"
#include <mbsim/integrators/integrators.h>

using namespace std;
using namespace MBSim;

int main (int argc, char* argv[])
{
    // build single modules
    DynamicSystemSolver *sys = new System("TS");

    // add modules to overall dynamical system
    sys->init();

    TimeSteppingIntegrator integrator;

    // other code
integrator.setStepSize(1e-4);
integrator.setEndTime(10.0);
integrator.setPlotStepSize(1e-3);

integrator.integrate(*sys);
cout << "finished" << endl;

delete sys;

return 0;
}

4.1.3. Makefile

The building process of a program is controlled by a Makefile:

# Name of the Executable
Target = main

# Defining Sources
sources = main.cc system.cc

# Do Not Edit the Following Lines
CXX = g++
libsources =
objects = $(sources:.cc=.o)
CPPFLAGS= -g3 -Wall -Werror -Wno-unknown-pragmas 'pkg-config --cflags mbsim'
CXXFLAGS= -std=c++17

$(Target) : $(objects)
	$(CXX) -o $@ $(objects) 'pkg-config --libs mbsim'

%.d: %.cc
	set -e; $(CXX) -MM $(CPPFLAGS) $< |
	| sed 's/\(/.*\)/.o[ :]*/\1.o \1.d : /g' > $@; |
	| -s $@ ] || rm -f $@

include $(sources:.cc=.d)

.PHONY : clean
 clean :
	-rm $(Target) $(objects) $(sources:.cc=.d)
4.2. 2D-Slider Crank Mechanism

Use the following plan.

1. rigid bodies with
   • *RigidBody* for crank, pistin and block
   • mass/ length/ width/ inertia tensor
   • Jacobian matrices (translation / rotation)
   • reference frames
   • OpenMBV-bodies (probably one needs an additional translation as OpenMBV-bodies sometimes use another reference)

2. elastic connecting rod with
   • *FlexibleBody1s21RCM*
   • length/ width / height (cross-sectional area) / Young's modulus (10e8)/ area moment of inertia / density/ damping
   • number of finite elements
   • reference frames
   • initial generalised coordinates

3. frames/ contours in body coordinate system

4. link definition

5. external loads

6. add to dynamic system solver

7. integrator / time step size

4.3. 3D-Slider Crank Mechanism

Use the following plan.

1. rigid bodies with
   • *RigidBody* for crank, connecting rod, piston and block
   • mass/ length/ width / inertia tensor
   • Jacobian matrices (translation / rotation)
   • reference frames
   • OpenMBV-bodies
2. frames / contours in body coordinate system
3. link definition
4. external loads
5. add to dynamic system solver
6. integrator / time step size
A. Trouble-Shooting

A.1. GNU-Build-System

The role of the GNU-Build-System (Figure 7) can be summarized as follows.\[10\]

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7.png}
\caption{GNU build system}
\end{figure}
### A.2. Third Party Software

Necessary for the installation of a static visualisation part are the following preliminary steps with everything being installed by package service or from scratch with some hints:

- export PKG_CONFIG_PATH=/home/OpenMBV/local/lib/pkgconfig
- Coin3d with version 3 or newer (3D scenegraphs)
  
  ./configure --prefix=/home/OpenMBV/local --disable-shared --enable-static

- hdf5 with version 1.8.13 or newer (file handling)
  
  ./configure --prefix=/home/OpenMBV/local --disable-shared --enable-static --enable-cxx --with-zlib=no

• Qt with version 4.4 or newer (2D user interface)
  
  ./configure -prefix /home/OpenMBV/local -static -nomake examples
  -nomake demos
  -nomake docs -nomake translations -no-gif -no-libtiff -qt-libpng
  -no-libmng
  -qt-libjpeg -no-openssl -no-glib

• HDF5Serie (wrapper for file handling)
  
  ./configure --prefix=/home/OpenMBV/local --disable-shared --enable-static

• SoQt with version 1.4.1 or newer (e.g. event invocation in Coin due to
  hardware events)
  
  in SoQtComponent.cpp:103’ change "unsigned long key" to "uintptr_t
  key"
  export QTDIR=/home/OpenMBV/local
  export CONFIG_QTLIBS="$(pkg-config --libs QtGui QtCore Qt3Support
  QtOpenGL)"
  ./configure --prefix=/home/OpenMBV/local --disable-shared --enable-static

• Qwt with version 5 (GUI elements)
  
  in qwtconfig.pri change INSTALLBASE=/home/OpenMBV/local
  /home/OpenMBV/local/bin/qmake

• Octave with version 3.0 or newer (for XML preprocessing)

A.3. Path Information

After recognising difficulties concerning path information mistakes in the ob-
jects should be ruled out by make clean. Then, path information in .bashrc
and finally in the affected .pc-files should be checked. The location of the
.pc-files can be found by

pkg-config --cflags mbsim
pkg-config --libs mbsim

A.4. Often Needed Linux Advices

• After editing $HOME/.bashrc the shell has to be restarted or the command
  source $HOME/.bashrc has to be invoked

B. MBSim - Coding Standard

In the following the Coding standards of MBSim and the associated projects is
defined.
// MBSim - Coding Standard

// General

/*
 * - Use english names and comments.
 * - A class name starts with a uppercase character.
 *   The second and more words start with uppercase characters.
 *   Don't use underscores.
 * - A function name starts with a lowercase character.
 *   The second and more words start with uppercase characters.
 *   Don't use underscores.
 * - An indent consists of 2 spaces.
 *   (Use "set cindent shiftwidth=2" in .vimrc)
 * - Don't use tabs at all. So 4 indents are 8 spaces and NOT 1 tab.
 *   (Use "set expandtab" in .vimrc)
 * - Use const where possible.
 * - Implementation in the same order like declaration.
 * - Set pointers to NULL after declaration and delete.
 * - & and * belongs to the variable, so leave no space in between
 * - no using namespace in h-file
 * - for all new features there should exist a test case with reference data
   in the examples directory for automatic compilation / running / comparison
   with reference data
 * - before new "git push": solve conflicts in compiling
   -> run test script to also avoid link and conceptional problems
   -> check for memory leakage, runtime and thread savity with valgrind
   (--tool=memcheck, callgrind, helgrind)
*/
// Code Style

// Long comments:
/*
 * long comment
 * over more than one line
 */

// Short comments:
// short comment only one line

// Class with doxygen comment:
/**
 * \brief short description of the class
 * \author author of the first version
 * \date yyyy-mm-dd changelog description (name of author)
 * \todo possible improvements TODO
 */
class Foo : public Bar {
    public:
        // constructor
        // virtual destructor

        /* INHERITED INTERFACE */
        virtual void foo(); // which interface, no doxygen comments necessary?
    /**********************************************************************************

        /* INTERFACE */
        virtual void foo();
    /**********************************************************************************

        /* GETTER / SETTER */
        const string& getName() const; // getter
        string& getName() const; // getter
        setName(string &s); //setter
        bool hasBoolean() const; // boolean
        bool isBoolean() const;

    private:

44
void foo();

protected:
/**
 * \brief description of the function
 * \param parameter description in the order of the function syntax
 * \return return value description
 */
int bar(int i, double x); // some comment
float inlineFoo() { return 5.3; }
float inlineBar()
{
    foo();
    return 5.2;
}

string s;
bool bl;

// (Element) Funktion:
void Foo::foo() {
    bar();
}

// namespace
namespace Fritz { Oskar; }
namespace Fritz {
    Oskar;
}

// short if:
if (i == 4) printf("test\n");
// or
if (i == 4)
    printf("test\n");

// long if:
if (i == 4) {
    printf("test1\n");
    printf("test2\n");
// short if else:
if (i==4)
    printf("test1\n");
else
    printf("test2\n");

// long if else:
if (i==4) {
    printf("test1a\n");
    printf("test1b\n");
}
else {
    printf("test2a\n");
    printf("test2b\n");
}

// short for
for (int i = 0; i < 5; i++)
    cout << "bla" << endl;
// or
for (int i = 0; i < 5; i++)
    cout << "bla" << endl;

// long for
for (int i = 0; i < 5; i++) {
    cout << "bla" << endl;
}

// short while:
while (i!=4) printf("test2a\n");
// or
while (i!=4)
    printf("test2a\n");

// long while:
while (i!=4) {
    printf("test1\n");
    printf("test2\n");
}
// short do-while:
do
    printf("test\n");
while(i==5);

// long do-while:
do {
    printf("test1\n");
    printf("test2\n");
}
while(i==5);

// switch-case:
switch(a) {
    case 1: printf("test1\n"); break
    case 2:
        printf("test2a\n");
        printf("test2b\n");
        break;
    default:
        printf("test3\n");
}

// Header file (filename myhead.h):
#ifndef MYHEAD_H
#define MYHEAD_H

// header code

#endif // MYHEAD_H

// Long preprocessor if:
#ifdef FOO
    foo1();
    foo2();
#else // not FOO
    bar1();
    bar2();
#endif // FOO
References


