HOTINT

Version 1.2.196

October 8, 2015

Contents

Chapter 1

General Information

Introduction

Development history and background information

The code HOTINT has been initiated by Johannes Gerstmayr in 1997 and, until now, gone over the following steps:

- solution methods and basic linear algebra routines for static solver (diploma thesis of the main developer, 1997)
- addition of time integration methods for the accurate solution of large-scale flexible and discontinuous multibody systems (up to 2004)
- integration with graphical interface in 2003 (with Yury Vetyukov)
- implementation of various structural finite elements, such as flexible beam and plate elements based on the absolute nodal coordinate formulation
- implementation of the floating frame of reference concept, as well as the component mode synthesis
- HOTINT made available to and further developed by Linz Center of Mechatronics (since 2007)
- HOTINT made available to and further developed by Austrian Center of Competence in Mechatronics (from 2008 to 2013)
- User version of HOTINT V1.1 available as freeware (2013)
- A open source version of HOTINT is available (end of 2013)

Current State of HOTINT

HOTINT mainly consists of the multibody kernel, the solver and linear algebra kernel, and the graphics and user interface, and currently comprises several hundred thousand lines of code. It has been particularly developed for the use of arbitrary classes of fully implicit Runge Kutta (IRK) methods. The IRK-tableaus can be defined in an external text-file and are given for several methods for 1 to 10 stages. The code makes advantage of the very high order reached through the use of fully implicit methods, which makes it especially then fast, when higher

accuracy is needed.
In the current version, the K-form of IRK-equations has been implemented for the fast integration of 2^{nd} order (mechanical) systems. Instead of trying to invert the mass matrix, which leads to large terms in the case of symbolic inversion, or instead of trying to add the system as a constraint equation (this has been done by some people who implemented their system into existing codes), you can now provide the mass matrix and the right hand side separately and the solver only solves one large system, but does not need the accelerations to be written explicitly as function of the remaining unknowns.

Summarizing, advanced methods from flexible multibody dynamics cover

- the efficient geometrical description for moving rigid bodies and bodies with superimposed small deformation,
- the application of special finite element methods, which are well suited for simulating large deformations of structural elements,
- high-order implicit time-integration schemes, in order to enforce stability for the numerical solution,
- a sophisticated treatment of algebraic equations for the arbitrary coupling of bodies, and for the incorporation of certain (boundary) conditions,
- and finally the reduction of the system size by a component mode synthesis (CMS).

General Information

Chief developer

Johannes Gerstmayr

Further developers

Larissa Aigner, Markus Dibold, Alexander Dorninger, Rafael Eder, Peter Gruber, Alexander Humer, Karin Nachbagauer, Astrid Pechstein, Daniel Reischl, Martin Saxinger, Markus Schörgenhumer, Michael Stangl, Yury Vetyukov, Simon Weitzhofer

Contact

support@hotint.org

Linz Center of Mechatronics GmbH
Altenbergerstr. 69, 4040 Linz, AUSTRIA
http://www.lcm.at

Thanks

The help and support from the contributors of the Institute of Technical Mechanics and Institute of Numerical Mathematics at the Johannes Kepler University of Linz is greatly appreciated.

I would like to acknowledge the important grant of the FWF ("Fond zur Förderung Wissenschaftlicher Forschung" - the Austrian National Science Fund) within the project P15195-N03 and the APART project of the Austrian Academy of Sciences.

Parts of this software have been developed in the project "Nachhaltig ressourcenschonende elektrische Antriebe durch höchste Energie- und Material-Effizienz" (sustainable and resource saving electrical drives through high energy and material efficiency) which is part of the European Union program "Regionale Wettbewerbsfähigkeit OÖ 2007-2013 (Regio 13)" sponsored by the European Regional Development Fund (ERDF) and the Province of Upper Austria.

Parts of this software have been developed with the support of the Comet K2 Austrian Center of Competence in Mechatronics (ACCM).

Link

`http://www.hotint.org`

Copyright and license

HotInt General License (Version 1.0)

Copyright (c) 1997 - 2013 Johannes Gerstmayr, Linz Center of Mechatronics GmbH, Austrian Center of Competence in Mechatronics GmbH, Institute of Technical Mechanics at the Johannes Kepler Universitaet Linz, Austria. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This program contains SuperLU 4.3, ExtGL, BLAS 3.5.0 and LAPACK 3.5.0 covered under the following licenses:

SuperLU 4.3

Copyright (c) 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
(3) Neither the name of Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

ExtGl

Copyright (c) 2002, Lev Povalahev. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

BLAS 3.5.0

The reference BLAS is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web. Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.

```
Like all software, it is copyrighted. It is not trademarked, but we do ask the
   following:

#      If you modify the source for these routines we ask that you change the
   name of the routine and comment the changes made to the original
#      We will gladly answer any questions regarding the software. If a
   modification is done, however, it is the responsibility of the person who
   modified the routine to provide support.


LAPACK 3.5.0
==========

Copyright (c) 1992-2013 The University of Tennessee and The University
                        of Tennessee Research Foundation.  All rights
                        reserved.
Copyright (c) 2000-2013 The University of California Berkeley. All
                        rights reserved.
Copyright (c) 2006-2013 The University of Colorado Denver.  All rights
                        reserved.

$COPYRIGHT$

Additional copyrights may follow

$HEADER$

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer listed
  in this license in the documentation and/or other materials
  provided with the distribution.

- Neither the name of the copyright holders nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

The copyright holders provide no reassurances that the source code
provided does not infringe any patent, copyright, or any other
intellectual property rights of third parties.  The copyright holders
disclaim any liability to any recipient for claims brought against
recipient by any third party for infringement of that parties
intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The following libraries are not linked to HOTINT directly but are requirements of the software libraries BLAS and LAPACK.

LIBGFORTRAN
=======

Copyright (C) 2002-2013 Free Software Foundation, Inc.
Contributed by Paul Brook <paul@nowt.org>, and
Andy Vaught <andy@xena.eas.asu.edu>

Libgfortran is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version.

Libgfortran is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Under Section 7 of GPL version 3, you are granted additional permissions described in the GCC Runtime Library Exception, version 3.1, as published by the Free Software Foundation.

You should have received a copy of the GNU General Public License and a copy of the GCC Runtime Library Exception along with this program; see the files COPYING3 and COPYING.RUNTIME respectively. If not, see <http://www.gnu.org/licenses/>.

LIBGCC
=======

Copyright (C) 2005-2014 Free Software Foundation, Inc.

GCC is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version.

GCC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Under Section 7 of GPL version 3, you are granted additional permissions described in the GCC Runtime Library Exception, version 3.1, as published by the Free Software Foundation.

You should have received a copy of the GNU General Public License and a copy of the GCC Runtime Library Exception along with this program; see the files COPYING3 and COPYING.RUNTIME respectively. If not, see <http://www.gnu.org/licenses/>

LibQuadmath
=======

```
GCC Quad-Precision Math Library
Copyright (C) 2010, 2011 Free Software Foundation, Inc.
Written by Francois-Xavier Coudert <fxcoudert@gcc.gnu.org>
This file is part of the libquadmath library.
Libquadmath is free software; you can redistribute it and/or
modify it under the terms of the GNU Library General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.
Libquadmath is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Library General Public License for more details.
You should have received a copy of the GNU Library General Public
License along with libquadmath; see the file COPYING.LIB.  If
not, write to the Free Software Foundation, Inc., 51 Franklin Street - Fifth Floor,
Boston, MA 02110-1301, USA.


Libwinpthread
=============
Copyright (c) 2011 mingw-w64 project

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.


/*
 * Parts of this library are derived by:
 *
 * Posix Threads library for Microsoft Windows
 *
 * Use at own risk, there is no implied warranty to this code.
 * It uses undocumented features of Microsoft Windows that can change
 * at any time in the future.
 *
 * (C) 2010 Lockless Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 *
 *  * Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *  * Neither the name of Lockless Inc. nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AN
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
GNU LIBRARY GENERAL PUBLIC LICENSE Version 2
=======

                  GNU LIBRARY GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1991 Free Software Foundation, Inc.
 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the library GPL.  It is
 numbered 2 because it goes with version 2 of the ordinary GPL.]

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

  This license, the Library General Public License, applies to some
specially designated Free Software Foundation software, and to any
other libraries whose authors decide to use it.  You can use it for
your libraries, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
```

if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify

that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse

engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the

Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the library's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
>
> You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110−1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
>
> <signature of Ty Coon>, 1 April 1990
> Ty Coon, President of Vice

That's all there is to it!

GNU GENERAL PUBLIC LICENSE Version 3

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING

WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

GCC Runtime Library Exception Version 3.1

GCC RUNTIME LIBRARY EXCEPTION

Version 3.1, 31 March 2009

Copyright (C) 2009 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This GCC Runtime Library Exception ("Exception") is an additional permission under section 7 of the GNU General Public License, version 3 ("GPLv3"). It applies to a given file (the "Runtime Library") that bears a notice placed by the copyright holder of the file stating that the file is governed by GPLv3 along with this Exception.

When you use GCC to compile a program, GCC may combine portions of certain GCC header files and runtime libraries with the compiled program. The purpose of this Exception is to allow compilation of non-GPL (including proprietary) programs to use, in this way, the header files and runtime libraries covered by this Exception.

0. Definitions.

A file is an "Independent Module" if it either requires the Runtime Library for execution after a Compilation Process, or makes use of an interface provided by the Runtime Library, but is not otherwise based on the Runtime Library.

"GCC" means a version of the GNU Compiler Collection, with or without modifications, governed by version 3 (or a specified later version) of the GNU General Public License (GPL) with the option of using any subsequent versions published by the FSF.

"GPL-compatible Software" is software whose conditions of propagation, modification and use would permit combination with GCC in accord with the license of GCC.

"Target Code" refers to output from any compiler for a real or virtual target processor architecture, in executable form or suitable for input to an assembler, loader, linker and/or execution phase. Notwithstanding that, Target Code does not include data in any format that is used as a compiler intermediate representation, or used for producing a compiler intermediate representation.

The "Compilation Process" transforms code entirely represented in non-intermediate languages designed for human-written code, and/or in Java Virtual Machine byte code, into Target Code. Thus, for example, use of source code generators and preprocessors need not be considered part of the Compilation Process, since the Compilation Process can be understood as starting with the output of the generators or preprocessors.

A Compilation Process is "Eligible" if it is done using GCC, alone or with other GPL-compatible software, or if it is done without using any work based on GCC. For example, using non-GPL-compatible Software to optimize any GCC intermediate representations would not qualify as an Eligible Compilation Process.

1. Grant of Additional Permission.

You have permission to propagate a work of Target Code formed by combining the Runtime Library with Independent Modules, even if such propagation would otherwise violate the terms of GPLv3, provided that all Target Code was generated by Eligible Compilation Processes. You may then convey such a combination under terms of your choice, consistent with the licensing of the Independent Modules.

2. No Weakening of GCC Copyleft.

The availability of this Exception does not imply any general presumption that third-party software is unaffected by the copyleft requirements of the license of GCC.

Chapter 2

HOTINT User Manual

2.1 Multibody formulation

The present code is based on a redundant coordinate formulation for the modeling of the motion and deformation of bodies. This means that e.g. every rigid body has its own six degrees of freedom (DOF), no matter how this body is constrained by other bodies or even if it is fixed to the ground. The main reason for this formulation is the simple extensibility of the code regarding the development of new elements, constraints, forces, etc. . The numerical efficiency is gained by adapted solvers for the sparse structure of the system equations, which leads to a similar effort as in recursive and minimal coordinate approaches.

Several main points have been focused in the multibody kernel:

- The application of implicit time integration algorithms shall be efficient
- The code shall be capable of structural and solid finite elements
- The code shall be extendable and open for new elements (e.g. non-mechanical, variable mass, variable topology, etc.)

Some things you should know:

Dimensions: dimensions are chosen by user, but should use standard international units: kg/m/s.

Numbering: All lists, arrays or other ordering numbers start with 1 if not specified differently.

Elements: Bodies and connectors are elements. If you search for bodies or connectors (e.g. for editing) in the HOTINT program, you should look for elements.

2.1.1 Solution vector

The multibody system and solver always have two solution vectors. One containing either the initial vector or the actual solution (this is the solution vector) and another one that is used for the graphics drawing which is called drawing solution vector. The latter vector is utilized to independently draw the solution of a certain computed time instant during the computation (e.g. if the computation lasts very long or is of indefinite length).
The solution vector is split into a “position level” (not necessarily a real position) and “velocity level” part for the case of the second order differential variables. Assume that there are n second order differential equation variables, then the solution vector will contain first n position level coordinates and after that n velocity level coordinates. The local coordinates of a body (e.g. accessible via the sensor) are ordered in a similar way. The local second order differential variables of a body contain first m position level coordinates and after that another m velocity level coordinates.

2.1.2 Main structure of the multibody kernel

There were some main points to be fulfilled with the present multibody kernel:

- The formulation shall be easily accessible and maintainable via C++ functions
- The formulation shall be easily accessible and maintainable via the Windows user interface.

In the current implementation there is one base multibody system object which contains all information about the system. On top of this structure, there is a dynamic and static solver class, i.e. an implicit time integration method and a incremental nonlinear solver. The solver requires the multibody system to provide residuals and derivatives of the differential and algebraic equations based on assumed values.

The multibody system consists of the following components:

- Elements
- Nodes
- Loads
- Sensors
- Geometric elements

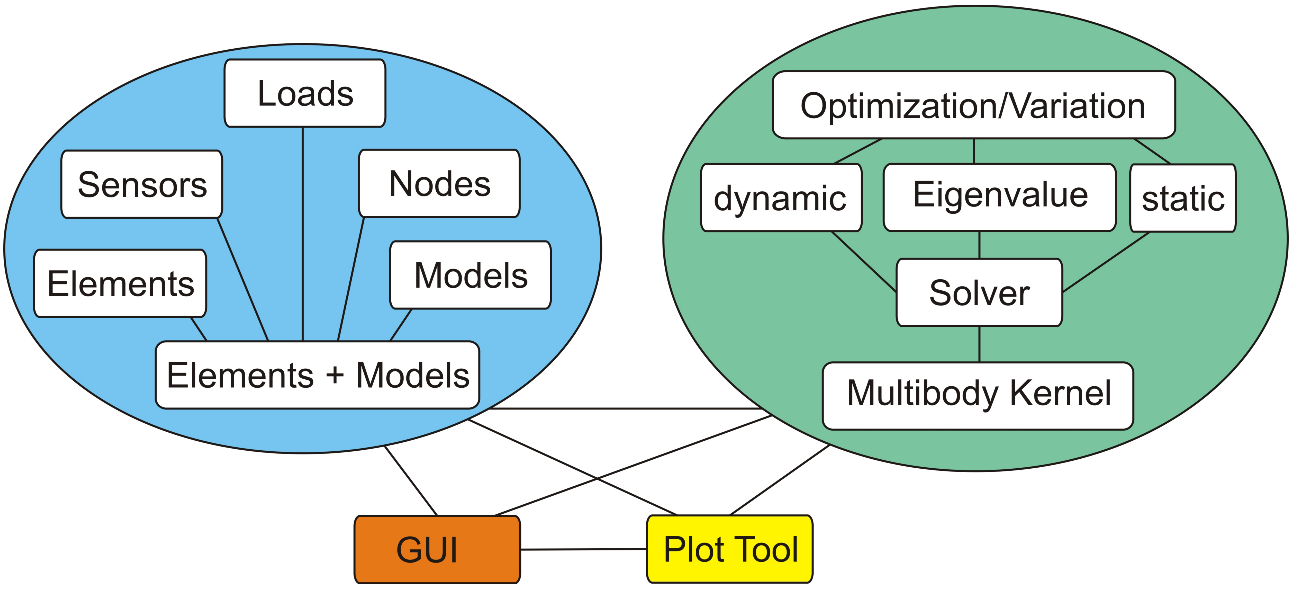


Figure: Multibody system core and windows interface.

Every object of a multibody system, see Sec. 2.1.3 and Fig. 2.1, adds a certain set of their own (local) equations to the whole set of (global) equations. The crucial task of the *Multibody System kernel* is to assemble these global equations based on the connections of the multibody system, and to provide the system equations to the solver. Apart from that, the kernel is responsible for setting up the model, steering the simulation, organizing in- and output of file data, as well as accessing or modifying specific element data.

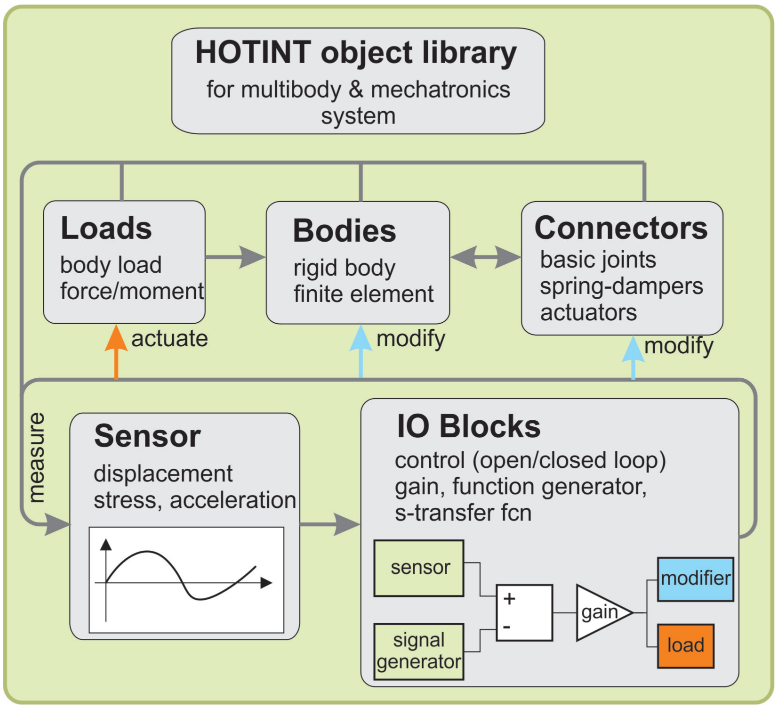


Figure 2.1: Structure of the multibody system (MBS).

2.1.3 Object library

The object library provides a set of rigid bodies (links), basic joints, loads and sensors, similar to any other simulation code. As a main feature of HOTINT, there exists a variety of flexible bodies (Finite Elements), connectors (actuators, springs, and dampers), loads, sensors, and IO-Blocks (controllers) – as outlined in Fig. 2.1.

Among flexible bodies are structural Finite Elements for beams, available either in geometrically exact formulation (for large deformation processes, ropes, cables, etc.), or in linearized form (faster). Joints are designed such that complex combinations of bodies and joints are possible, e.g. a point of body may move along a deformable body's axis (sliding joint). Since also *flexible* bodies are available, the object definition is based on generalized (redundant) coordinates for the bodies.

In the core part of the code, objects are represented by means of first order and second order differential equations, algebraic equations and jump or switching conditions. Furthermore, the objects include standardized coupling conditions (for joints and loads), graphical representation and measurable quantities (for sensors). These objects define bodies (links) and joints and may be easily extended.

2.1.4 The dynamic solver – implicit time integration

The numerical time-integration tool included in HOTINT is designed to compute the numerical solution of mixed first and second order differential equations (ODE) and differential-algebraic equations (DAEs) up to an index of 3. The numerical solution is obtained by using implicit Runge-Kutta (IRK) schemes like Gauss, Radau and Lobatto formulas. The code is developed for an arbitrary number of stages, so far 20 stages have been tested resulting in the conclusion that computing with as much as 10 stages can improve the speed of the numerical simulation before the machine precision is limiting the convergence of the underlying Newton method.

Different IRK schemes are defined by tableaus of coefficients which are defined by means of ASCII-Files (file "tableaus.txt"). These files are automatically generated by means of built in functions of the code Mathematica 5.0. While it is known that multi-step solvers can integrate DAEs of index 2 (e.g.\ BDF), it has been found out that the inability to restart the multi-step method quickly after a discontinuous step makes it unattractive for discontinuous problems. Furthermore, the order of multi-step methods is limited by a comparatively low upper bound, while it is possible to show that an order of 20 for the integration is possible and can even be most efficient.

It shall be mentioned that in the special case, where a high accuracy of the solution of the DAE is needed, e.g. for sensitivity analysis or optimization methods, the very high order of IRK methods is very advantageous.

For the present case of the freeware HOTINT code, only low order IRK formulas are available, while the higher order methods will be available in future versions.

For a description of the methods see the paper (download via homepage of HOTINT):

J. Gerstmayr, M. Stangl. High-Order Implicit Runge-Kutta Methods for Discontinuous Multibody Systems, In: Proceedings of the XXXII Summer School APM' 2004, June 24- July 1, Editor D.A. Indeitsev, pp. 162-169, St. Petersburg, Russia, 2004.

2.1.4.1 Index 2 Formulation

In the present implementation, only the index 2 formulation can be chosen. The index 2 multibody formalism transforms all constraints to the velocity level. This leads to a highly stable and efficient formalism (the velocity level can be solved much easier than the position level).

The time integration algorithm forces the constraint conditions at the velocity level in each time step (at the integration points of each time step). The integration over the velocity does not exactly give the fulfillment of the position level constraint, thus a small drift occurs. The drift becomes considerably smaller with smaller step size and can be usually ignored.

Recommendations: *Do not select too large time steps.* If you have fast rotating bodies, it is important to guarantee sufficient time steps during each rotation of the bodies., It is usually sufficient to use between 20 and 100 steps per one rotation in order to get sufficient accuracy and small drift.

Future implementations: Stabilization techniques are already included in HOTINT, but they need to be built into the general framework. The stabilization as well as the error control of the drift will be available in future versions of HOTINT.

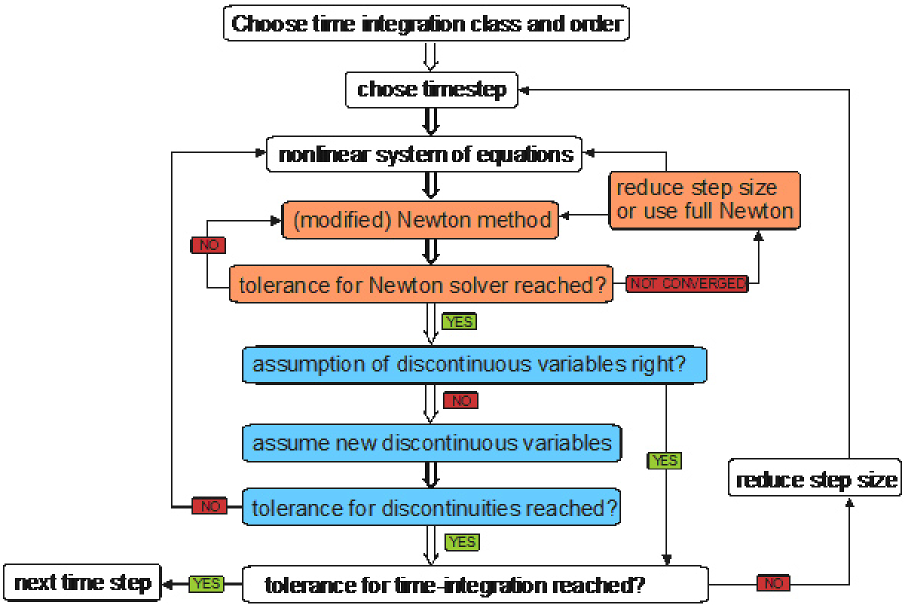

Choose time integration class and order
chose timestep
nonlinear system of equations
(modified) Newton method
reduce step size or use full Newton
NO
tolerance for Newton solver reached?
NOT CONVERGED
YES
assumption of discontinuous variables right?
NO
assume new discontinuous variables
YES
reduce step size
NO
tolerance for discontinuities reached?
YES
next time step
YES
tolerance for time-integration reached?
NO

Figure: Scheme of the dynamic solver.

2.1.5 The static solver – incremental loading

The nonlinear solver sets all velocities and acceleration terms to zero. The solver tries to find a static solution (if possible) starting with the initial configuration. All loads are increased linearly between the virtual time 0 and 1, in order to achieve convergence for very nonlinear problems. The (virtual) time step (=load increment) can be theoretically set to 1, but then the load is applied in one step and the nonlinear problem needs to be solved at once. The static solver tries to decrease the load increment as far as necessary in order to achieve convergence, however, it is advantageous to specify a certain load increment which can help the solver to speed up the computation and avoid failed steps.
The static solver does not work for kinematical systems (statically underdetermined systems). Small rotational or translational springs can be added in order to transform the system to a statically determined system.

2.1.6 Eigenmode computation

There are different methods used in order to compute eigenmodes of the multibody system. The different methods are described below. The eigensolver in HOTINT does not work yet for general Lagrange-multiplier constraints, although it is known how to compute eigenmodes for problems with Lagrange multipliers, [11]. Presently, all penalty-based constraints can be used and constraints can be applied on single coordinates, e.g. in order to obtain clamped constraint conditions.

How to compute the eigenvalues and eigenmodes for a still standing multibody system:

Equations of motion: $\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{K}(\mathbf{x})\mathbf{x} = \mathbf{0}$

Computed are the eigenvalues/modes of the first order system $\mathbf{A}$:

$$\ddot{\mathbf{x}} = \mathbf{A}\,\mathbf{x} \;=\; \ddot{\mathbf{x}} = \left[-\mathbf{M}^{-1}\mathbf{K}\right]\mathbf{x} \tag{2.1}$$

$$\mathbf{K}\,\mathbf{v} \;=\; \lambda\mathbf{M}\,\mathbf{v} \tag{2.2}$$

$\mathbf{M}$ ... mass matrix of the multibody system
$\mathbf{K}$ ... stiffness matrix of the multibody system
$\mathbf{v}$ ... eigenvectors of matrix $\mathbf{A}$
λ ... eigenvalues of matrix $\mathbf{A}$

1.) Open the menu Edit Solver Options

2.) Set general options, they are independent from selected solver
2 a) Eigensolver.do_eigenmode_computation ... if checked → eigenvalue computation on button START.
2 b) Eigensolver.linearize_about_actual_solution ... use actual solution as configuration for linearization of $\mathbf{K}/\mathbf{M}$. Eigenvalues are computed for linearization around stored solution vector of last static/dynamic solution! All velocities are set to zero.
2 c) Eigensolver.use_gyroscopic_terms ... make sure that box is not checked
2 d) Eigensolver.eigenmodes_scaling_factor ... scaling factor for eigenmodes, eigenvectors are multiplied with this factor
2 e) Eigensolver.eigenmodes_normalization_mode ... $0 \rightarrow$ standard mode, $max(v) = 1$; $1 \rightarrow v^T v = 1$;
2 f) Eigensolver.use_n_zero_modes ... flag is not used in current version
2 g) Eigensolver.reuse_last_eigenvectors ... flag is not used in current version

3.) Set the subtree Eigensolver.solver_type ... define the solver type
0 ... LAPACK dsygv direct solver, LAPACK package used
The solver will calculate all possible eigenvalues/eigenmodes of the multibody system. Solver options are not offered. For information about the accuracy see the LAPACK documentation.

1 ... Arnoldi iterative solver (Matlab), Matlab licence is needed
Eigensolver.max_iterations ... maximum number of iterations for iterative eigenvalue solver
Eigensolver.accuracy ... tolerance for iterative eigenvalue solver
Eigensolver.n_eigvals ... number of eigenvalues and eigenmodes to be computed for sparce iterative methods
Eigensolver.n_zero_modes ... number of zero eigenvalues (convergence check)

2 ... LOBPCG iterative solver, implemented in HOTINT
Eigensolver.max_iterations ... maximum number of iterations for iterative eigenvalue solver
Eigensolver.accuracy ... tolerance for iterative eigenvalue solver
Eigensolver.use_preconditioning ... if checked → set a value for lambda in Eigensolver.preconditioner_lambda, $inv(K + \lambda M)$
Eigensolver.n_eigvals ... number of eigenvalues and eigenmodes to be computed for sparce iterative methods
Eigensolver.n_zero_modes ... number of zero eigenvalues (convergence check)

How to compute the eigenvalues and eigenmodes for a multibody system with gyroscopic terms:

Equations of motion: $\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{G}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{K}(\mathbf{x})\mathbf{x} = \mathbf{0}$

Computed are the eigenvalues/modes of the first order system **A**:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{E} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{G} \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \ddot{\mathbf{x}} \end{bmatrix} = \mathbf{A}\begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} \tag{2.4}$$

$$\mathbf{A}\,\mathbf{v} = \lambda\mathbf{v} \tag{2.5}$$

M ... mass matrix of the multibody system
K ... stiffness matrix of the multibody system
G ... gyroscopy matrix of the multibody system
v ... eigenvectors of matrix **A**
λ ... eigenvalues of matrix **A**

1.) Open the menu Edit Solver Options

2.) Set general options, they are independent from selected solver
2 a) Eigensolver.do_eigenmode_computation ... if checked → eigenvalue computation on button START.
2 b) Eigensolver.linearize_about_actual_solution ... use actual solution as configuration for linearization of **K**/**M**. Eigenvalues are computed for linearization around stored solution vector of last static/dynamic solution!
2 c) Eigensolver.eigenmodes_scaling_factor ... scaling factor for eigenmodes, eigenvectors are multiplied with this factor
2 d) Eigensolver.eigenmodes_normalization_mode ... 0 → standard mode, $max(\bar{\mathbf{v}}) = 1$; 1 → $\bar{\mathbf{v}}^T\bar{\mathbf{v}} = 1$; Attention: For a proper drawing representation the vector $\bar{\mathbf{v}}$ (used for normalization) contains only the the positions $\left(\mathbf{v} = \left[\bar{\mathbf{v}}, \dot{\bar{\mathbf{v}}}\right]\right)$.
3.) Check Eigensolver.use_gyroscopic_terms ... use gyroscopy terms for eigenvalue computation The eigenvalues/modes of the nonsymmetric matrix **A** are computed with the LAPACK dgeev solver. Other options for this solver are not necessary. Relating to the accuracy see the LAPACK documentation.

The computation of the eigenvalues/eigenmodes requires a inversion of the full mass matrix of the dynamic system. This could be a problem for very large systems.

How to create a campbell diagram, e.g. for rotordynamics:
It is very simple to create a campbell diagram with HOTINT. To create a campbell diagramm do the following steps:

1.) Set up a rotor model, e.g. by adding RotorBeamXAxis and NodalDiskMass3D elements and Node3DR123 nodes to the multibody system

1 a) Initialize all nodes Node3DR123 in dependence of the variable you want to vary, e.g. in the case of the campbell diagram the rotor speed omega:
Initialization.node_initial_values $= [0, 0, 0, 0, 0, 0, 0, 0, 0, \mathrm{omega}, 0, 0]$
2.) Set the eigensolver, see "How to compute the eigenvalues and eigenmodes for a multibody system in motion"
3.) Set a parametervariation for omega (range and step size)
4.) Perform computation, the eigenvalues and varied parameter are stored in the solution file, e.g. solpar.txt in the output folder
5.) Open the plot tool and load output file:
5 a) Click External file and select the output file, e.g. solpar.txt
5 b) Select n_rot and a eigenvalue of your choice, e.g. eigval1 and create a x/y plot
6.) For a campbell diagram it is necessary to add a line with the frequency of the rotor speed. Create a txt file with the following lines:

Example

```
%Comment: y=x/60 (x in 1/min, y in Hz)
%1     2
%n_rot frequency
0 0
x y
```

Replace the x with the max. rotor speed and y with calculated frequency value, load the file and create a x/y plot.
7.) Label the plot

In the following is an excerpt of such a rotor example (the full example is included in examples/campbell):

Example

```
% Rotor Beam Example -> Campbell diagram
% parameters
%...

n_rot = 1000 % rpm, vary this parameter
omega = 2*Pi*n_rot/60 % rad/s

%=====================================================================
% rotordynamics model
%=====================================================================

% add materials (the material does not depend on omega)
%...

% add node 1 with initial velocity
n
{
  node_type = "Node3DR123"
  name = "node_1"
  Geometry.reference_position = [0,0,0]
  Initialization.node_initial_values =
  [0, 0, 0, 0, 0, 0, 0, 0, 0, omega, 0, 0]
  % initial values for all DOF of node: 1...6 => pos, 7...12 => vel
}
n1 = AddNode(n)
% ...similar for all other nodes

% add rotor beams
% ...
% add nodal disk masses
% ...
% add bearings
% ...

%======================================================================
% set parameter variation
%======================================================================
solveroptions.parametervariation.activate = 1 % do parameter variation
solveroptions.parametervariation.start_value = 0 % rpm
solveroptions.parametervariation.end_value = 80000 % rpm
solveroptions.parametervariation.arithmetic_step = 1000 % rpm
solveroptions.parametervariation.mbs_edc_variable_name = "n_rot" % name

%======================================================================
% set eigensolver
%======================================================================
SolverOptions.Eigensolver.use_gyroscopic_terms = 1 % use gyro terms
SolverOptions.Eigensolver.do_eigenmode_computation = 1 % must be set to 1
```

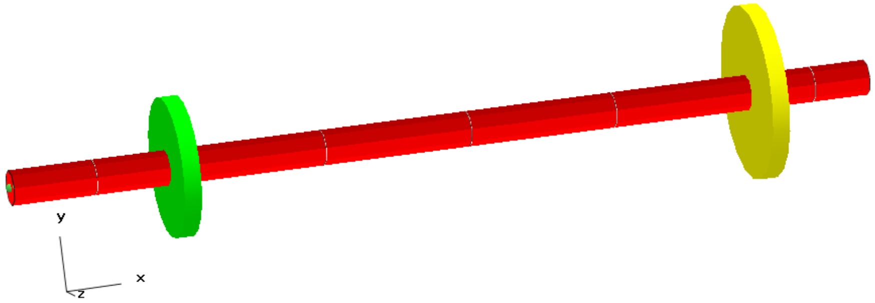


Figure: Two disk rotor created with file campbell.txt

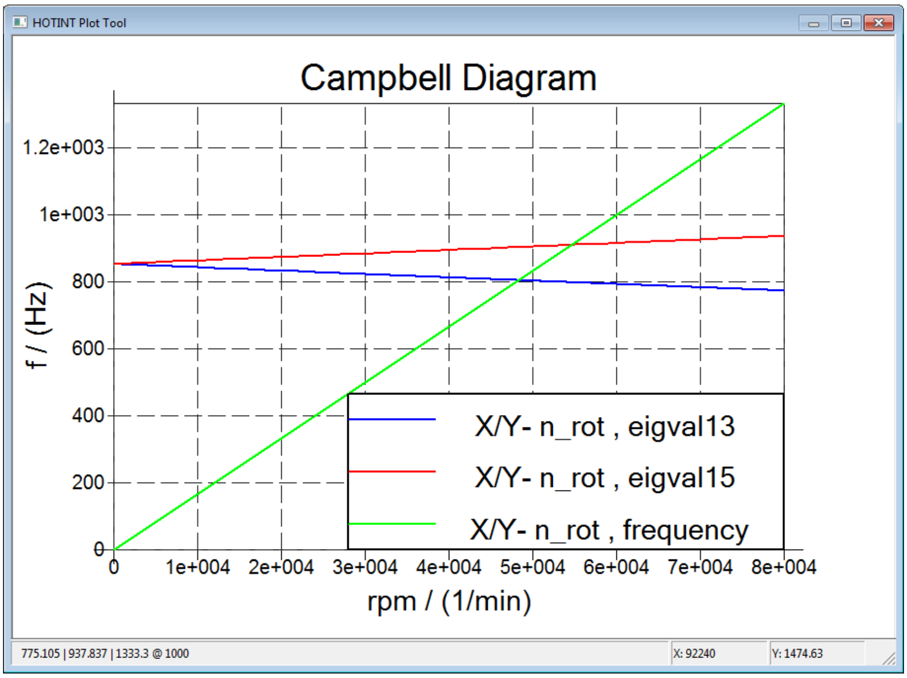


Figure: Campbell diagram created with the file examples/campell. The eigenfrequencies are the first bending modes.

2.1.7 Parameter Variation, Sensitivity Analysis, Identification and Optimization

In order to study the global influence of certain parameters to the simulation results, a parameter variation can be performed, which e.g. gives a set of results with respect to one or two varied parameters. In order to investigate the local influence of specific parameters on the solution, a sensitivity analysis can be performed, which results in a matrix which shows the dependence of a cost function with respect to the change of parameters. Based on the functionality of the parameter variation, it is possible to perform optimization and parameter identification in HOTINT. The implemented genetic parameter identification algorithm, documented in [12], is used to search the best fitting model parameters in a systematic way. The cost function for the identification/optimization can be based on the difference of a reference solution, e.g. from measurements and simulated results. The algorithm searches the optimal parameters in a given parameter space (e.g. parameter ranges). Multiple minima of the cost function may occur and are no problem for the genetic algorithm. In contrast to NewtonŠs method, derivatives of the cost function with respect to the parameters are not required. In a first step, for each set of randomly chosen parameters, a simulation is performed and the cost function is evaluated. A specified number of best parameters is taken into account for the next generation of parameters, the surviving parameters. Based on these parameters a new set of parameters (children) are generated using the principle of mutation and the parameter search range is reduced. This procedure is repeated until the optimum is nearly reached.

Parameter Variation: In the menu Edit Solver Options under the subtree SolverOptions.ParameterVariation the parameter variation can be set.

For the start of the parameter optimization, following things have to be done:

1.) Set up your HOTINT model which contains a parameter for variation

2.) SolverOptions.ParameterVariation.MBS_EDC_variable_name ... define EDC-name of the parameter (e.g. "i")

3.) Define the range of the parameter value. The variation is repeated as long as the $p_i \leq p_n$.
3 a) SolverOptions.ParameterVariation.start_value ... start value of the parameter p_0
3 b) SolverOptions.ParameterVariation.end_value ... end value of the parameter p_n

4.) Define arithmetic or geometric step method ... arithmetic: $p_i = p_{i-1} + \Delta p$; geometric: $p_i = p_{i-1} f$; p_i ... parameter value at step i;
4 a) SolverOptions.ParameterVariation.geometric ... check for geometric step, else arithmetic step
4 b) SolverOptions.ParameterVariation.arithmetic_step ... set Δp
4 c) SolverOptions.ParameterVariation.arithmetic_step ... set f

5.) Activate variation algorithm
5 a) Check field SolverOptions.ParameterVariation.activate

In the following there is a simple example code of a parameter variation. A parameter i is varied from 1 to 5 with a step size of 1 and displayed in the output window.

Example

```
% Test for printing in combination with parameter variation
HOTINT_data_file_version="1.1.498"
i=1

Print("The number is ")
Print(i)
Print("\n")

SolverOptions.ParameterVariation.activate = 1
SolverOptions.ParameterVariation.start_value = 1
SolverOptions.ParameterVariation.end_value = 5
SolverOptions.ParameterVariation.arithmetic_step = 1
SolverOptions.ParameterVariation.MBS_EDC_variable_name = "i"
```

Genetic Optimization: Generally, the subtree SolverOptions.Optimization in the menu Edit Solver Options contains methods for optimization or in other words minimization of certain computation values (or cost function) from sensor signals in form of a search of the best-matching parameters. See the excerpt of the hid file of a two mass oscillator (examples/two_mass_oscillator), which shows the optimization of a unknown spring stiffness. The optimization is based on the difference to a reference two mass oscillator example.

For the start of the parameter optimization, following things have to be done:

1.) Set up your HOTINT model with at least one sensor (e.g. Two-Mass-Oscillator)

2.) Define computation value(s) from sensor signal(s) with SolverOptions.Optimization.sensors. They are minimized by the optimization; if more than one sensor computation value is defined, the sum of the computational value will be minimized

3.) Define optimized parameters and their limits
3 a) SolverOptions.Optimization.number_of_params ... number of parameters, which are optimized (e.g. 1)
3 b) SolverOptions.Optimization.param_name1 ... define EDC-name of optimized parameter(e.g. "k1_var")
3 c) SolverOptions.Optimization.[min|max]val1 ... define limits for the parameter search (e.g. SolverOptions.timization.minval = 0 and SolverOptions.Optimization.maxval = 1)
3 d) Repeat a)-c) until all parameters and limits are defined
4.) Check the Genetic Optimization Options SolverOptions.Optimization.Genetic

This option should only be modified, if the computation time or accuracy of the optimization process should be changed. For more accurate results increase the
SolverOptions.Optimization.Genetic.initial_population_size,
SolverOptions.Optimization.Genetic.surviving_population_size,
SolverOptions.Optimization.Genetic.number_of_children or try to change the other options in SolverOptions.Optimization.Genetic. This is a very critical point, because the accuracy but also the computation time is increased. Further descriptions and more detailed insight to the influence of the genetic optimization parameters can be found in previous work (R. Ludwig and J. Gerstmayr, AUTOMATIC PARAMETER IDENTIFICATION FOR GENERIC ROBOT MODELS, MULTIBODY DYNAMICS 2011, ECCOMAS Thematic Conference, J.C. Samin, P. Fisette (eds.), Brussels, Belgium, 4-7 July 2011).

5.) Activate optimization algorithm
5 a) Check field SolverOptions.Optimization.activate
5 a1) Optional: check field SolverOptions.Optimization.run_with_nominal_parameters (for checking the consistency of the model and the nominal sensor computation value)
5 a2) Optional: check field SolverOptions.Optimization.restart (if genetic optimization should be restarted with already known parameters from previous genetic optimizations). This option saves computation time if results from previous optimization(s) should be used.
5 b) Set option SolverOptions.Optimization.method = "Genetic". Further algorithms are planned.

6.) Press OK-Button in Edit Solver Options, then Start! in the main window
The optimization repeats the simulation with different parameter sets and writes usually further informations about the optimization process into the Computation Output - Window. Furthermore, a result file is written into the path GeneralOptions.Paths.sensor_output_path with the filename defined in the option SolverOptions.Solution.ParameterFile.parameter_variation_filename (e.g. solpar.txt). This file is needed for the SolverOptions.Optimization.restart option, see 5 a2).

7.) Check result
7a) Use optimized parameters as nominal parameters, simulate once (e.g. with 5 a1))
7b1) If results are accurate enough → optimization process finished.
7b2) otherwise repeat points 3.)-7.)

Important notes:
For a higher speed of the optimization it is useful to close the GUI Data Manager as well as the Computation Output. If you want to see some information about the optimization progress during the computation open the static output window instead (Results → Show Static Output). It is also recommended to uncheck SolverOptions.Solution.write_solution in Edit Solver Options. Otherwise the sensor data of every optimization step is written to the output file

SolverOptions.Solution.SolutionFile.output_filename (e.g. sol.txt).

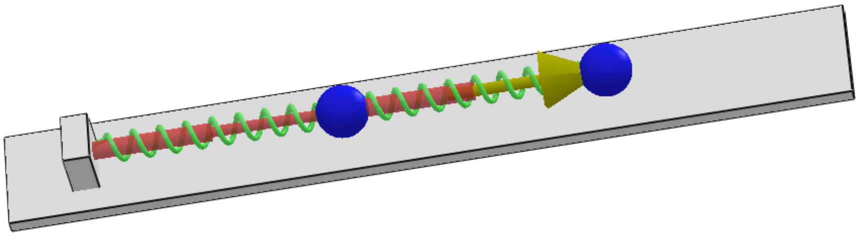


Figure: Two mass oscillator with stiffness and damping optimization.

An excerpt of the full example (included in examples/two_mass_oscillator) is written below:

Example

```
% parameters:

% ...
k1= 500   % N/m, stiffness spring 1, nominal value
k1_var= 350 % N/m, stiffness spring 1, arbitrary value, not used
                % for optimization, this value is used if
                % run_with_nominal_parameters= 1
d2= 30    % N/(m/s), damping spring 2, nominal value
d2_var= 10  % N/(m/s), damping spring 2, arbitrary value


%======================================================================
% read vectors from file (measurement data) and create math functions
%======================================================================

t = LoadVectorFromFile("...path...",1)
disp_m1 = LoadVectorFromFile("...path...",2)
vel_m1 = LoadVectorFromFile("...path...",3)
disp_m2 = LoadVectorFromFile("...path...",4)
vel_m2 = LoadVectorFromFile("...path...",5)


mathFunction
{
    MathFunction
    {
      piecewise_mode= 1
      piecewise_points= t
      piecewise_values= disp_m1
    }
  }
}
nSensDisp1Ref = AddElement(mathFunction)

%... similar for other math functions ...
```

```
%======================================================================
% model with varied parameters
%======================================================================

% ... add masses, spring dampers, sensors with varied parameters (in this
% case only the stiffness of spring damper 1 differs to nom. parameters)

spring_damper1
{
  % ...
  Physics.Linear.spring_stiffness= k1_var
}
nSpringDamper1Var= AddConnector(spring_damper1)

spring_damper2
{
  % ...
  Physics.Linear.damping= d2_var
}
nSpringDamper2Var= AddConnector(spring_damper2)

%======================================================================
% optimization
%======================================================================

nSensor= AddSensor(...) % this sensor measures the cost function (in
% this example it is the average of sum of the quatratic errors of the
% displacments and velocities of the masses)

SolverOptions
{
  Optimization
  {
    activate= 1  % set this flag for optimization
    run_with_nominal_parameters= 0  % 1..perform single simulation
    restart= 0  %0..create new parameter file
    method= "Genetic"  % genetic: optimize using random parameters,
                       % best parameters are further tracked.
    sensors= nSensor % sensor which measures the cost function
  }
  Genetic
  {
    initial_population_size= 20  % size of initial trial values.
    surviving_population_size= 15  % values which are further tracked
    number_of_children= 15  % number of children of surviving population
    number_of_generations= 4  % number of generations in genetic optimization
    range_reduction_factor= 0.5  % reduction of range of possible mutations
    randomizer_initialization= 0  % initialization of random function
    min_allowed_distance_factor= 0  % set to value greater than zero
  }
  Parameters
  {
    number_of_params= 2  %Number of parameters to optimize.
    param_name1= "k1_var"  %Parameter name.
    param_minval1= 3e2  %Lower limit of parameter.
    param_maxval1= 7e2  %Upper limit of parameter.
    param_name2= "d2_var"  %Parameter name.
    param_minval2= 10  %Lower limit of parameter.
    param_maxval2= 70  %Upper limit of parameter.
  }
 }
}
```

Results of the optimization taken from solpar.txt file:

k1 = 498.642 N/m
d2 = 30.0115 N/(m/s)
cost function = 3.21674e-005; The cost function in this example is the sum of squares of deviations between positions and velocities (see example file for more detail).

For a visualization of the optimization results open Results → PlotToolDialog. Select External File as Data Sources and choose the optimization file (e.g. solpar.txt as default filename) in the output folder. Click k1_var and Ctrl + cost_function_value and select Add x/y. Change the Point Style to X and the Line Style to invisible. Add a Title label X-Axis and Y-Axis. Save the picture and repeat the procedure for d2_var. You should get the figures below.

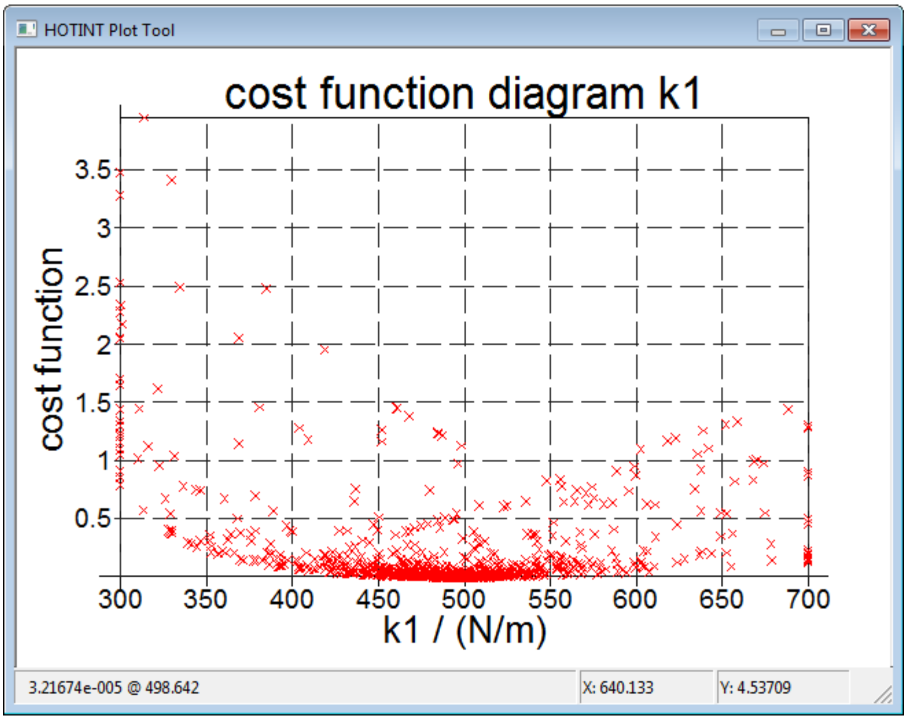
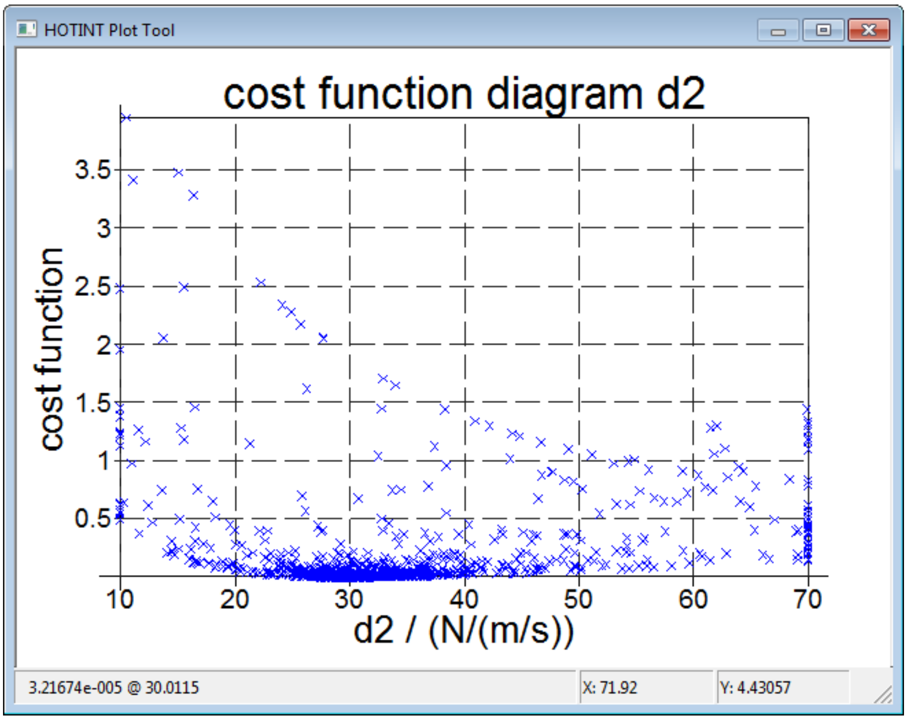


Figure: Cost function in dependence of spring stiffness k1.



3.21674e-005 @ 30.0115

X: 71.92

Y: 4.43057

Figure: Cost function in dependence of spring damping d2.

2.1.8 The Element Concept

Elements: Bodies and connectors are elements. In fact, an element only needs to provide a set of differential and algebraic equations and it can add forces to other elements. A rigid body modeled with Euler parameters includes one constraint for the four Euler parameters and a hydraulic actor includes a differential equation for the pressure build-up equations. Therefore, bodies and connectors are treated within the same framework. Even forces or sensors could be elements, however, there would be too much overhead in treating just everything as an element.

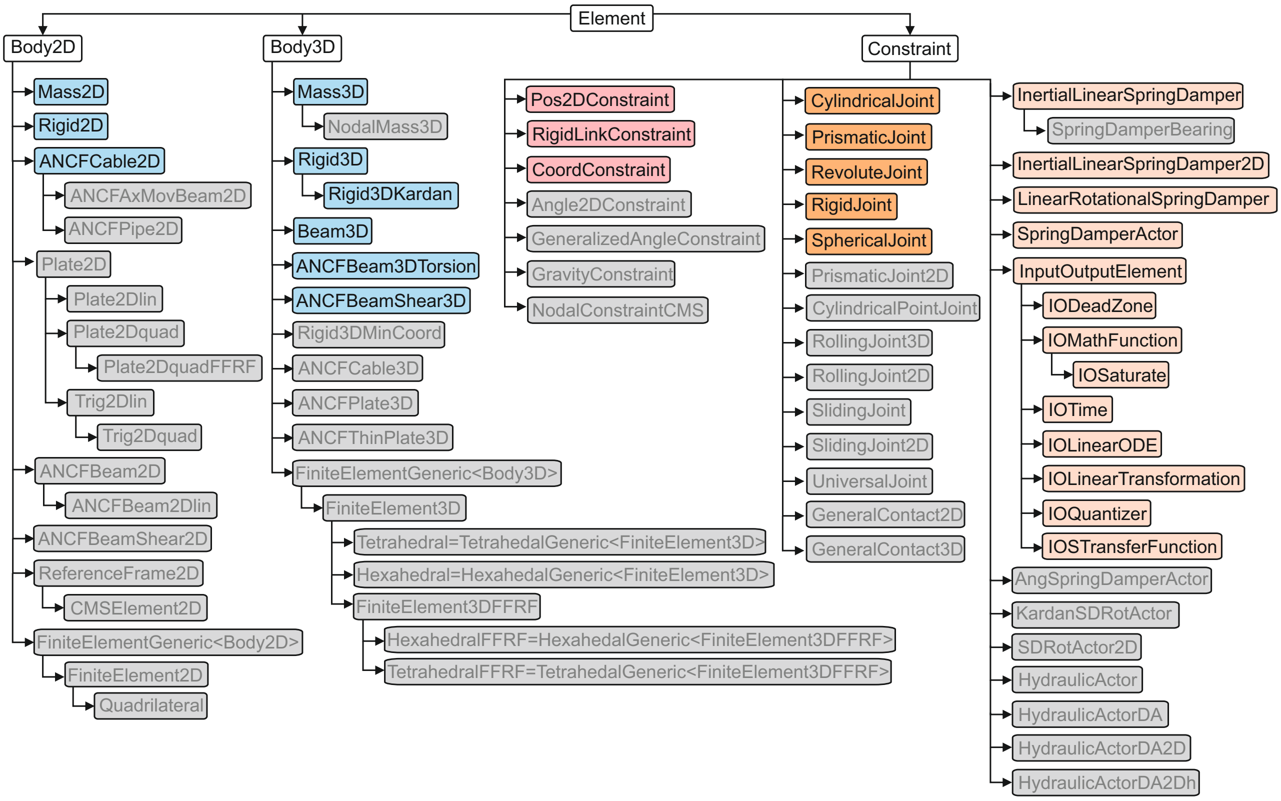


Figure: Element class structure; elements which are not yet available in the current freeware version via the script language (cf. section 2.4.2) are greyed out.

2.1.9 Nodes for Direct Connection of Finite Elements

Sometimes it is more efficient no connect two elements without the application of constraints. E.g. in the case of nodal finite elements it is advantageous if the connected elements share nodal coordinates. Therefore, it is possible in HOTINT to define nodes, which can be afterwards used to assign nodal coordinates to elements.

A node is defined only for a certain number of coordinates (degrees of freedom – DOF), e.g. for a 2D position node DOF = 2, for a 3D position node DOF = 3, for a node using position and gradient, the DOF = 12 per node. Additionally, the nodal position in the reference configuration can be assigned to the node. This position can be later on used to find nodes or to automatically determine the nodal number depending on the nodal coordinate.

The nodes are consecutively ordered starting with the nodal number 1. The elements can afterwards refer to this number. When editing nodes, the available nodal numbers are shown.

2.1.10 The Concept of Loads

Loads are used to add forces at the right hand side of the second order differential equations that describe the dynamics of a body. Loads are directly linked to bodies and they do not have own generalized coordinates (unknowns). However, loads can depend on the body coordinates or body deformation (e.g. in the case of pressure).

The loads can have a time-dependency which is evaluated in every step of the computation.

Loads can only be applied to bodies that provide according information of the work of external linear, angular or integrated loads.

2.1.11 Sensors for Measuring

Sensors are used to measure certain quantities of the multibody system at the current state of the computation. The output of a sensor is usually written to output files at certain time steps (See Computation Settings dialog). The solution file "sol.txt" contains the output of all sensors, each sensor in a row, versus the time (first row). Apart from output and controllers, sensors do not influence the computation.

While local_DOF_sensors can be used to measure the coordinates of any element (e.g. of a constraint), the position, angle, distance and deflection sensors can only be applied to elements of the type body.

Note that local second order differential variables of a body contain first $[1 \ldots\ m]$ position level coordinates and another $[m{+}1 \ldots\ 2m]$ velocity level coordinates.

Sensors can not have own generalized coordinates (unknowns).

2.1.12 Geometric Elements for Bodies with Complex Geometry

Geometric elements are used to represent a realistic shape of complex bodies in the multibody simulation. Usually, a geometric element is either used to define objects in the background or it is attached to a (rigid) body.

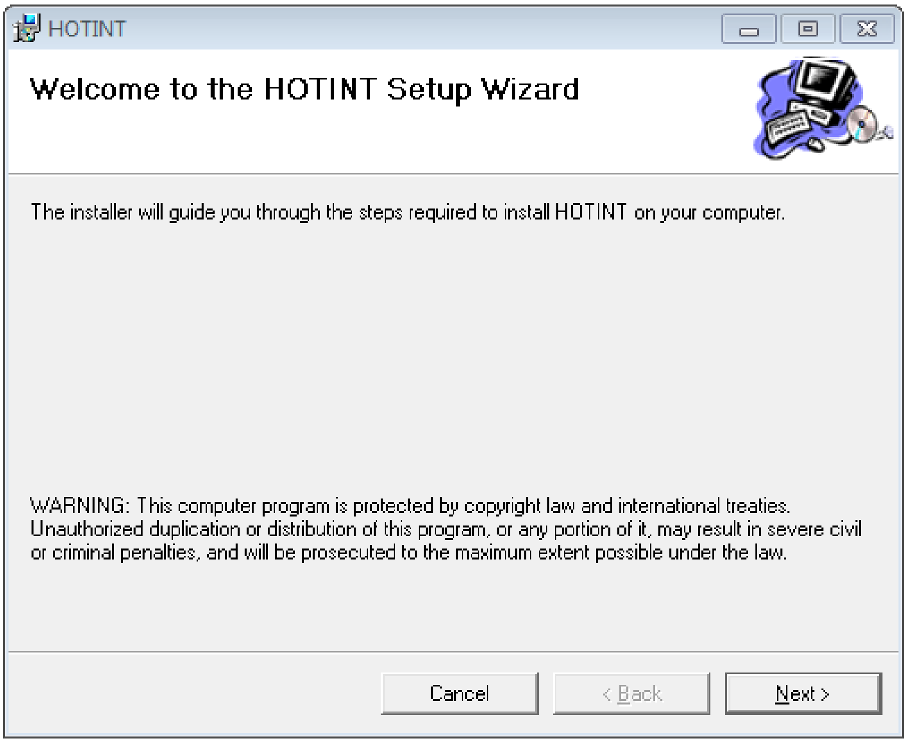
Geometric elements can be either defined with geometric primitives or by triangular meshes (see the Section about GeomMesh). The only influence to the computation by GeomElements is present by the automatic computation of mass, volume and inertia from the GeomElements. Usually, the complexity of GeomElements does not influence the computational time (CPU time), except for the drawing and loading/saving of multibody models. In the case of big GeomMesh models, it is recommended that the redrawing time is set to a high value, e.g. set the redrawing to every 20 seconds.

2.2 Getting started

2.2.1 Instructions for installing HOTINT on a MS-Windows computer

To begin with, you need to download the HOTINT zip-archive, and extract it to a folder of your choice using a program such as Winzip or 7-Zip. Then run the executable "setup.exe", and follow the setup instructions as shown below:

First, you will see the start screen of the HOTINT setup wizard:



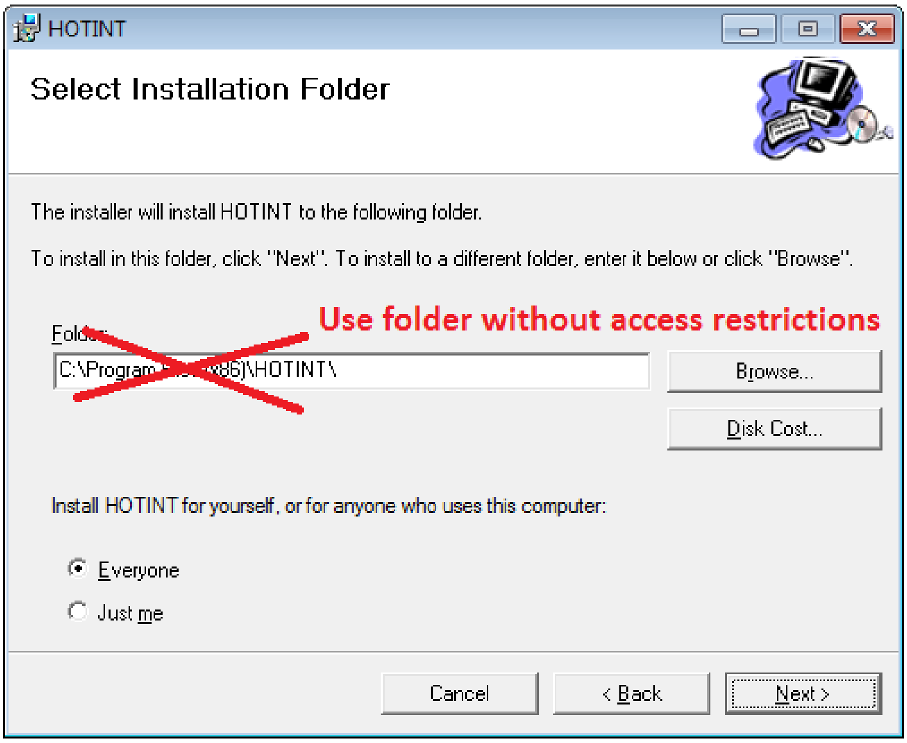
Welcome to the HOTINT Setup Wizard

The installer will guide you through the steps required to install HOTINT on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Click "Next" to proceed. Now you can choose the installation folder, and specify whether to install HOTINT for all users on your computer, or just for you.

ATTENTION: Do not use a folder which is locked by windows for admin use only, e.g. the default program folder. It is recommended to use a folder within the "documents" or "user" folders.



Select Installation Folder

The installer will install HOTINT to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Use folder without access restrictions

Fold

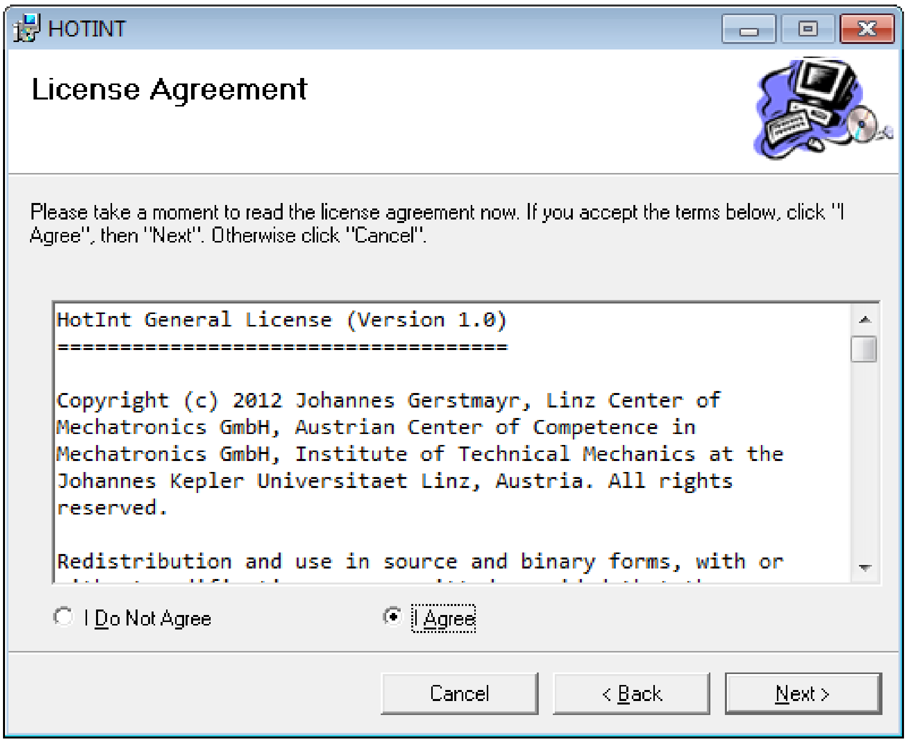
C:\Program [illegible]x86)\HOTINT\

Install HOTINT for yourself, or for anyone who uses this computer:

● Everyone

○ Just me

Click “Next”, read the license agreement, check “I Agree”, and click “Next” in order to proceed; click “Cancel” otherwise.



License Agreement

Please take a moment to read the license agreement now. If you accept the terms below, click "I Agree", then "Next". Otherwise click "Cancel".

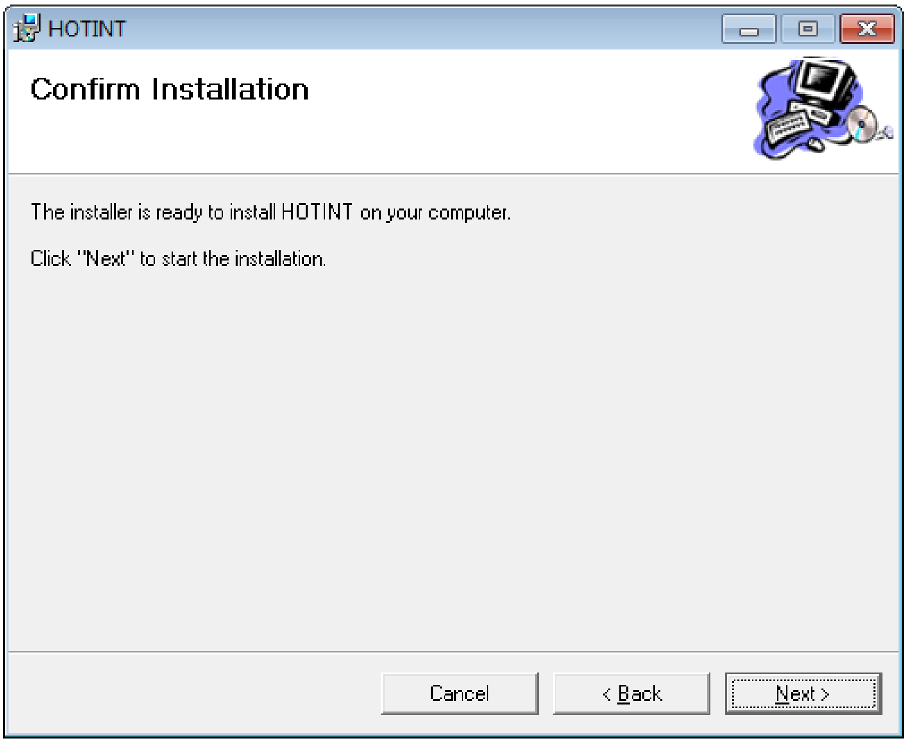
```
HotInt General License (Version 1.0)
====================================

Copyright (c) 2012 Johannes Gerstmayr, Linz Center of
Mechatronics GmbH, Austrian Center of Competence in
Mechatronics GmbH, Institute of Technical Mechanics at the
Johannes Kepler Universitaet Linz, Austria. All rights
reserved.

Redistribution and use in source and binary forms, with or
```

○ I Do Not Agree ● I Agree

HOTINT is now ready to be installed on your computer; click “Next” to start the installation.

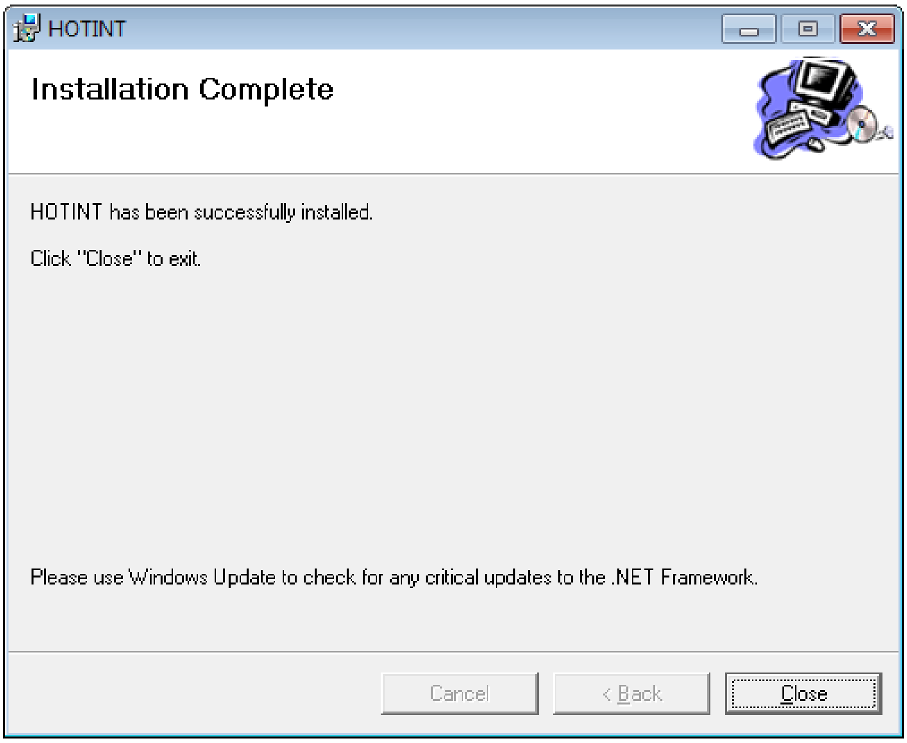


Confirm Installation

The installer is ready to install HOTINT on your computer.

Click "Next" to start the installation.

After the installation process, the following screen appears:



Installation Complete

HOTINT has been successfully installed.

Click "Close" to exit.

Please use Windows Update to check for any critical updates to the .NET Framework.

Click “Close” to exit the setup wizard.

The installation of HOTINT now is complete, and your your chosen installation directory should contain a number of “.dll”-files, as well as the following folders:

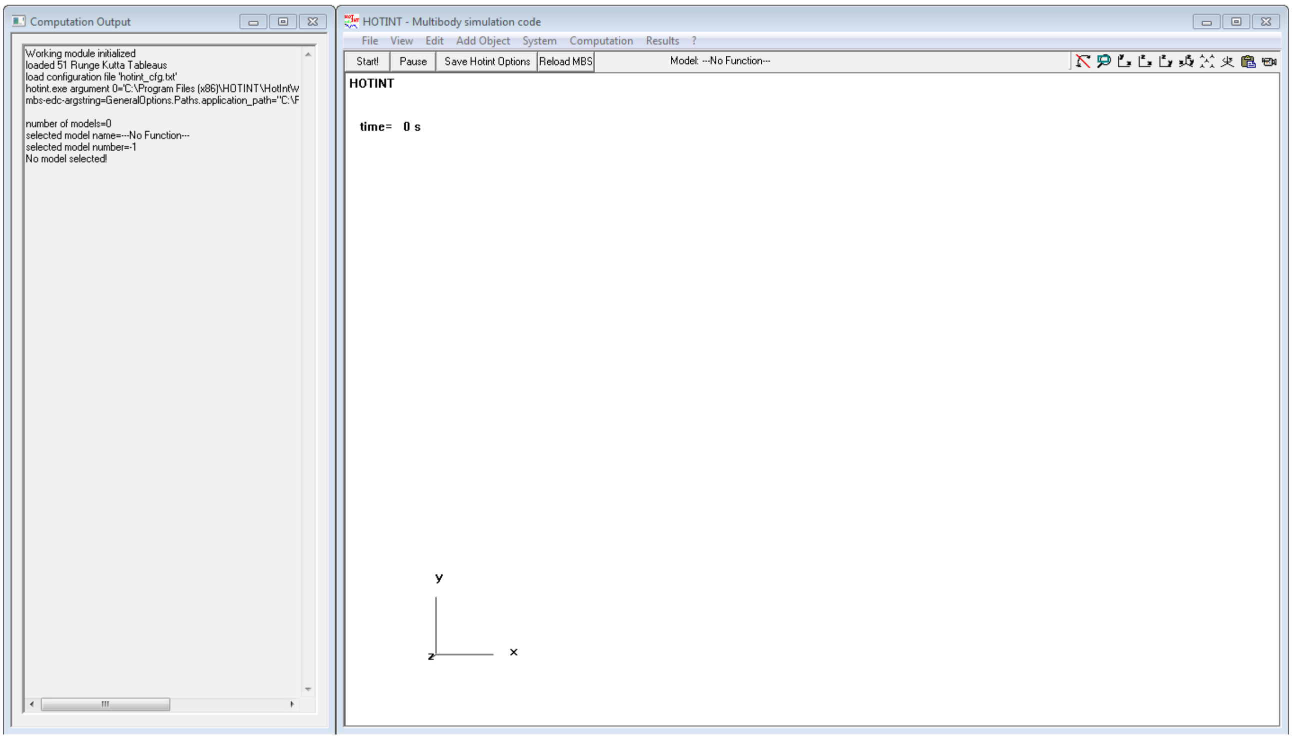
documentation	Contains the HOTINT user documentation, an “.rtf” license text file, and an “example” folder with ready-to-use “.hid” example model

	files.
HotIntWin32	Contains the subfolder release, where the HOTINT executable “hotint.exe” and the configuration file “hotint_cfg.txt” are located.
output	This is, by default, the output directory where the solution file “sol.txt” containing sensor data is stored. Furthermore, the solution data files are created here in a subdirectory “solution_data”, if the flag store_data_to_files is checked in the solver options under “Solution” (see 2.5.8 or section 3.13 for details).
userdata	This folder can be used for your user-defined model files (cf. 2.4).

HOTINT is started by running the executable “hotint.exe” in the “HotIntWin32\release” folder. For convenience, it is recommended to create a shortcut (e.g., on your desktop or in the start menu directory) referencing that executable. The following section guides you through your first steps in HOTINT.

2.2.2 First steps

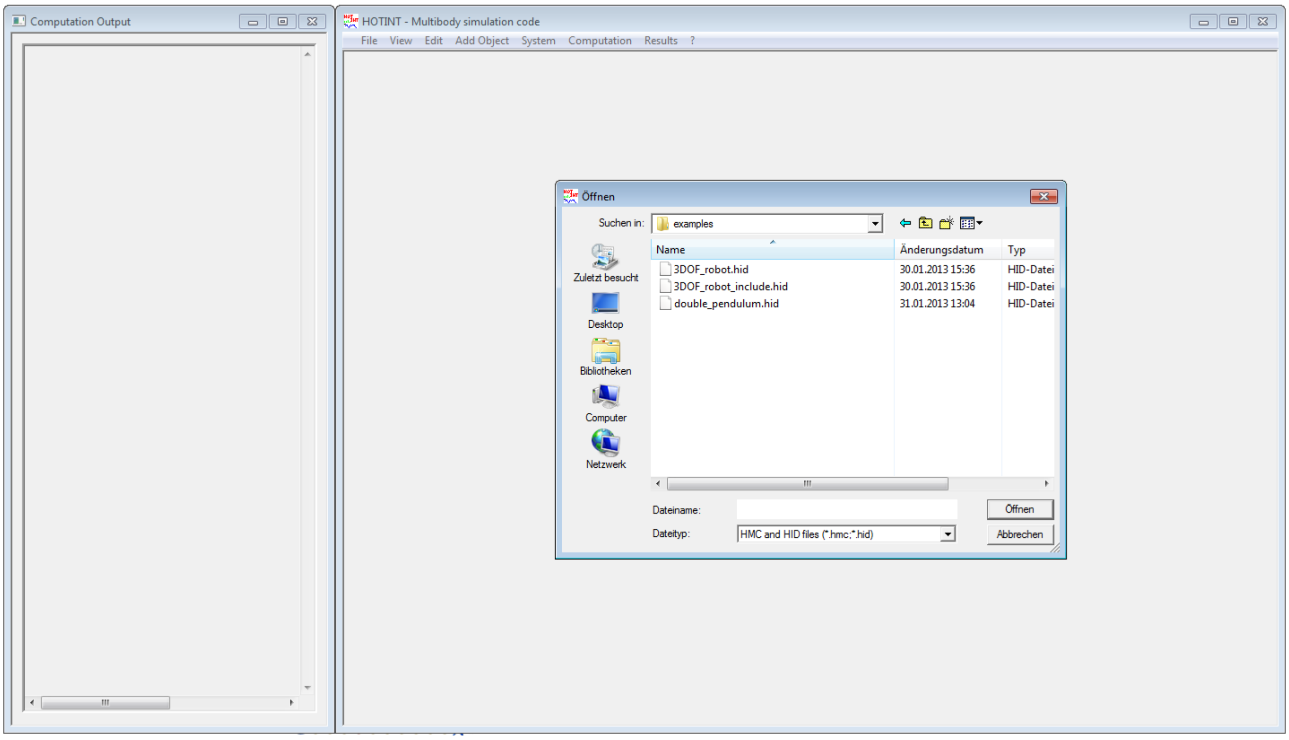
Start HOTINT by double-clicking “hotint.exe” located in the subfolder “HotIntWin32\release” in your installation directory, or a corresponding shortcut. The program starts with an empty multibody model.



```
Working module initialized
loaded 51 Runge Kutta Tableaus
load configuration file 'hotint_cfg.txt'
hotint.exe argument 0='C:\Program Files (x86)\HOTINT\HotIntW
mbs-edc-argstring=GeneralOptions.Paths.application_path="C:\F

number of models=0
selected model name=---No Function---
selected model number=-1
No model selected!
```

The best way to experience the capabilities of HOTINT is to load one of the examples included in the subfolder “examples” and start to experiment. Select “OpenMBS” in the “File”-menu, and navigate to the examples located in “documentation\examples”:



Suchen in: examples

Name	Änderungsdatum	Typ
3DOF_robot.hid	30.01.2013 15:36	HID-Datei
3DOF_robot_include.hid	30.01.2013 15:36	HID-Datei
double_pendulum.hid	31.01.2013 13:04	HID-Datei

Dateiname:

Dateityp: HMC and HID files (*.hmc;*.hid)

Here, we choose the file “double_pendulum.txt”...

2.2.3 Command Line Usage

HOTINT can also be configured and started via the command line (or from MATLAB). The syntax is

```
hotint [option1=value [option2=value [...]]]
```

Some remarks:

- When starting hotint.exe from DOS/Matlab, the current directory MUST be the root directory of hotint.exe.
- To start the Windows command prompt, run the executable cmd.exe (under Windows 7, just type “cmd.exe" in the search bar in the start menu and hit enter; alternatively, or in other Windows versions, use the “Run" command). Any settings of HOTINT options via the command line are accounted for after the HOTINT model file – possibly containing specifications for some options too – has been read in.
- Single option specifications must not include spaces; mutually, they are separated by spaces.
- Use \" instead of ".
- In order to run several instances of HOTINT in parallel, append the &-character at the end of each line which calls hotint.exe.

In order to open a model with HOTINT from command line use the options

```
GeneralOptions.ModelFile.hotint_input_data_filename
```

if you want to open a script model (hid-file) or

```
GeneralOptions.ModelFile.internal_model_function_name
```

if you want to run a C++ model (models compiled with HOTINT).
A few examples for starting HOTINT via the command line:

```
hotint.exe GeneralOptions.ModelFile.hotint_input_data_filename=\"D:\models\hotint_file.hid\"
  GeneralOptions.Application.start_computation_automatically=1
hotint.exe SolverOptions.Solution.output_filename=\"dir/myfile.txt\"
hotint.exe SolverOptions.Timeint.tableau_name=\"RadauIIA\"
```

One example using MATLAB:

```
dos('hotint.exe SolverOptions.Solution.output_filename=\"matlabfile1.txt\" &')
```

In C++ you can have user-defined options. You can set these options via the command line too:

```
hotint.exe MyOptions.usemodeNr=2
```

2.2.4 Configure Notepad++ for HOTINT

As described in 2.8.1 it is possible to set up systems with text-files. These files can be written and changed in any editor, e.g. notepad++. Some editors provide the functionality of syntax highlighting and an auto-complete function for user-defined languages.
For this purpose 2 specific files are stored on your computer during the installation process in the folder documentation:

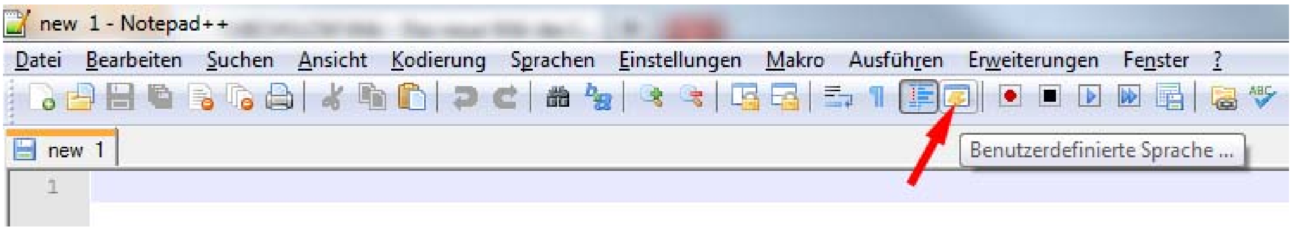
- HOTINT.xml
- hotint_highlight_notepad.xml

In the following it is described how to set up notepad++, such that these functionalities can be used. If you are using a different editor, the steps may be very similar.

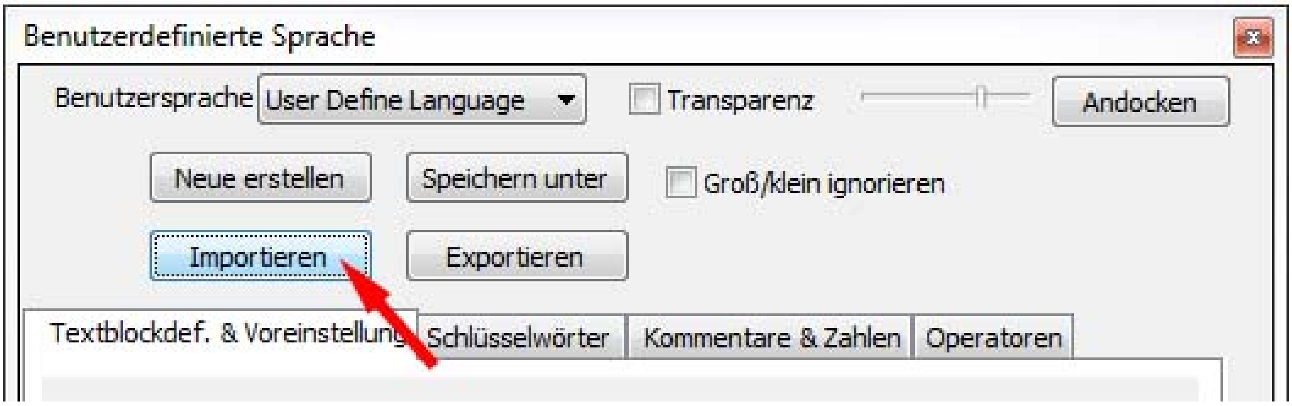
1. save file ’HOTINT.xml’ to the notepad folder ’plugins\APIs’
   (e.g. C:\Program Files (x86)\Notepad++ \plugins\APIs)

2. open NOTEPAD++

3. click on icon



4. import file ’hotint_highlight_notepad.xml



If you open a file with the extensions 'txt' or 'hid' with notepad++ there should be 2 new features now:

- highlighting of known keywords
- auto complete (ctrl + space) for known keywords

2.3 HOTINT Windows User Interface

2.3.1 Using the graphics window

The 3D graphics window is used to visualize the multibody model by user-defined representation of the bodies, joints and forces. The graphical representation might be a simplification of the parameters used to perform the dynamical simulation.

2.3.2 Mouse control

Rotation: Press the right mouse button and move up/downwards and left/right to rotate the model.

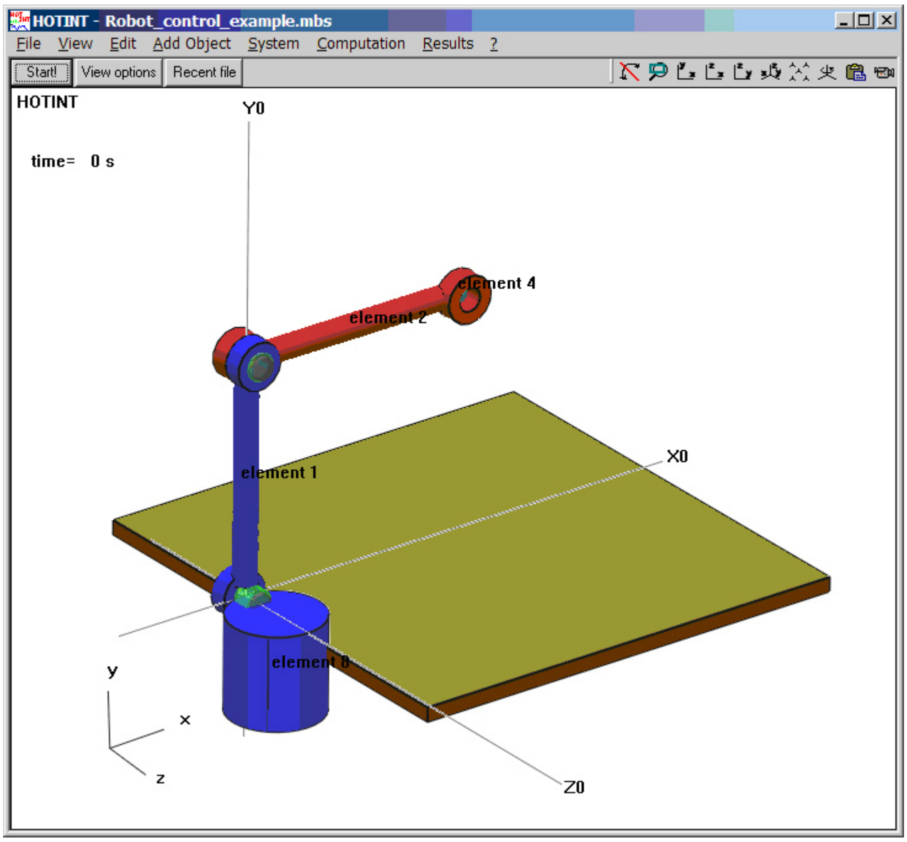
Zooming: Use the scroll wheel to zoom in / out or press the right mouse button and "Shift" and move up/downwards.

Zoom selection: Use "Shift" and the left mouse button and select a rectangle to be zoomed into.

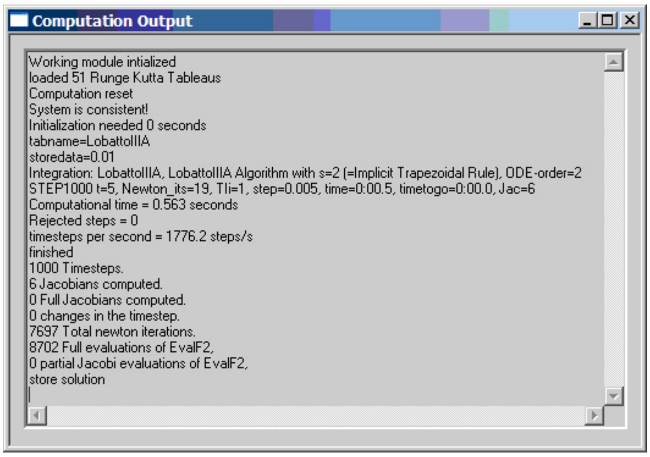
Moving: Press the left mouse button to move the model on the screen.

Perspective: Press the right mouse button, "Shift" and "Ctrl" and move up/downwards to change the distance of the camera to the object in order to change its perspective (the closer you zoom, the more distorted it gets).

2.3.3 HOTINT main application window



The main HOTINT window is used to load, save and edit models, start the computation, or modify computation parameters and viewing settings. After a computation the results can plotted as well as animated.



```
Working module intialized
loaded 51 Runge Kutta Tableaus
Computation reset
System is consistent!
Initialization needed 0 seconds
tabname=LobattoIIIA
storedata=0.01
Integration: LobattoIIIA, LobattoIIIA Algorithm with s=2 (=Implicit Trapezoidal Rule), ODE-order=2
STEP1000 t=5, Newton_its=19, TIi=1, step=0.005, time=0:00.5, timetogo=0:00.0, Jac=6
Computational time = 0.563 seconds
Rejected steps = 0
timesteps per second = 1776.2 steps/s
finished
1000 Timesteps.
6 Jacobians computed.
0 Full Jacobians computed.
0 changes in the timestep.
7697 Total newton iterations.
8702 Full evaluations of EvalF2,
0 partial Jacobi evaluations of EvalF2,
store solution
```

The "Computation Output" window is used to print important messages, show computation results, the computation state, computation background information (e.g. number of Newton iterations), error and warning messages.

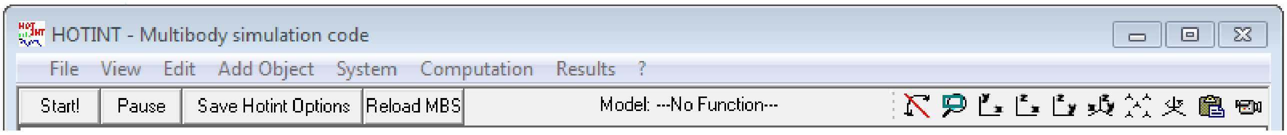
2.3.4 Specific buttons

The following buttons are available in the main view in HOTINT:

Start!	Start computation of multibody system
Pause	Pause computation of multibody system
Stop	Stop computation of multibody system
Restart	Restart computation of multibody system
Save Hotint Options	Save HOTINT options (i.e. the configuration except for the solver settings)
Reload MBS	Reload the selected (internal) model or the open skript file
	Enable/Disable rotation of model – for planar examples
	Zoom whole model
	Show x-y plane
	Show x-z plane
	Show y-z plane
	Show user-defined view (see viewing options)
	Choose / hide axes position
	Automatic rotation
	Save single image, directory is specified in record frames dialog
	Open the record frames dialog in order to capture a series of images for an animation
1 2 3	Click to change viewing options to stored ones Ctrl + Click to store current viewing options 3 independent settings are possible Filenames can be set in GeneralOptions.SavedViewingOptions

2.3.5 HOTINT Main Menu

Outlining, the HOTINT main menu comprises the following entries which are described in more detail below:



- File
- View
- Edit
- Add Object
- System
- Computation
- Results
- ?

2.3.5.1 File

Select Model	Select a multibody model
New MBS	Create a new multibody model
Open MBS	Open an existing MBS file
Save MBS (as)	Save the current multibody model (as...)
Exit	Exit program
Recent:	List of recent files

2.3.5.2 View

Edit Hotint Options	Open the Hotint options dialog: Access all options concerning the MBS and the program, except for the solver options. See subsection 2.5.2 or section 3.13 in the reference manual for details.
Save Hotint Options	Saves the current configuration of Hotint options
Show Data Manager	Open dialog for viewing and animating the results of the computation; see subsection 2.5.7 for further information.
Show Output Window	Show the output window which reports important information, current state of the simulation, errors, etc. during the computation and modeling
Viewing Options	Open the viewing options dialog: configure redrawing, animation settings, grid (raster), standard view; see subsection 2.5.3 or section 3.13 in the reference manual for details.
OpenGL Options	Set the options for OpenGL 3D graphics: define lights positions and intensities, transparency, shading model and lighting; see subsection 2.5.4 or section 3.13 in the reference manual for details.
FE Drawing options	Dialog mainly to change settings for finite elements: Contour-iso plots, color/grey mode, shrinking factor, stress-type, tiling, resolution, line thickness; see subsection 2.5.5 or section 3.13 in the reference manual for details.
Body / Joint Options	Used to configure the user-input and drawing of bodies and joints: Rotation input mode, show body number, body frame, body transparency Show joints, joint transparent and joint number; see subsection 2.5.6 or section 3.13 in the reference manual for details.
X-Y / X-Z / Y-Z	View X-Y / X-Z / Y-Z plane
Default View	Select the default viewing orientation, defined in the viewing options

2.3.5.3 Add Object

Add Element	Add a rigid or flexible body.
Add Connector	Add a joint/constraint/connector or a control element.
Add Load	Add a load to a rigid or flexible body: generalized coordinate load, body load, force vector, moment vector
Add Material	Add material for finite elements
Add BeamProperties	Add the properties of beam elements
Add Node	Add a node for finite elements
Add Sensor	Add a sensor in order to measure quantities of the computation: DOF sensor, position sensor, angle sensor, distance sensor, deflection sensor, multiple sensor
Add GeomElement	Add a geometric element to a body (preferably to rigid bodies): mesh, mesh imported from STL file, cylinder, sphere, cube
Add Set	Add a set of nodes or elements

For further information, refer to section 2.4 or sections 3.2–3.10 in the reference manual.

2.3.5.4 Edit

Undo	Undo the last add, delete or edit command
Edit Element, Load, Material,...	Edit the properties of the already added objects

For further information, refer to section 2.4 or sections 3.2–3.10 in the reference manual.

2.3.5.5 Delete

Delete Element, Load, Material,...	Delete an already added object

2.3.5.6 System

Show System Properties	Show some of the properties of the actual multibody system (number of elements, number of coordinates, constraints, etc.)
Verify System	Check some of the system properties such as if all element, constraint, sensor and geometric element references are valid. Check if constraints and sensors are only attached to valid bodies, etc.
Show global variables	Access and edit all parameters defined in the model data file
Run Macro (Add variable)	You can load a (small) txt file in order to enter anything available in the script language. This can be used to add global variables.

2.3.5.7 Computation

Edit Solver Options	Access and edit all solver options (such as for time integration, the static solver, the non-linear Newton solver or eigensolver, and settings concerning the in- and output of solution files, sensor data and parameter files. See subsection 2.5.8 or section 3.13 in the reference manual for details.
Save Solver Options	Save the solver options to a configuration file
Reset Simulation	The call to this function is necessary to reset the system to its initial state when it was built. This function is called every time an element is added, removed or changed. The function includes: • Restore to initial vector stored in elements • Reset starting time to t=0 • Remove all output from data manager • Assemble the system • Fit the model onto the screen
Start Simulation	Run the simulation from the starting time till the end time using the settings defined in the solver options
Stop Simulation	Terminate the simulation
Pause	Pause the computation which can be continued later
Load Initial Vector	Load a solution vector, which defines the initial conditions of the system, from a file. This vector can be smaller than the actual vector of initial unknowns, e.g. only initial positions can be loaded, while the initial velocities are used from the initial conditions defined in the elements.
Store Solution Vector	Store the solution at the current time instance in a file
Print CPU Statistics	Prints the approximate usage of CPU power for single parts of the multibody simulation (mass matrix, elastic forces, residual, linear solver, Jacobian, etc.)

2.3.5.8 Results

PlotToolDialog	Open the dialog for the plot tool which offers creating, editing, scaling, labeling, and exporting plots from one or several sensor signals of the actual simulation or imported from a solution file. See section 2.7 for details.
Plot Sensor	Plot the output data of a sensor versus time
Plot 2 Sensors XY	Create an XY-Plot from two individually chosen sensor signals
Sensor Watch	Open a small window that shows the actual value of a sensor

Enable Output	Enable output written into the output window. The output can be deactivated in order to reduce the computation time for writing into the edit window. This might be especially advantageous for very long simulations.
Show Static Output	Show the output in a separate window which does not update and can be used to analyze or copy the output during the computation.

2.3.5.9 “?"

About	Shows the “About"-dialog with some basic information about HOTINT
Help	Opens the “Help"-contents

2.4 Creating your model in HOTINT

2.4.1 Introduction

Clearly, when working with multibody simulation tools, the subject of model setup and configuration is of central importance. In HOTINT, there are two possibilities to create a multibody system:

- creating a model file using the HOTINT script language (recommended)
- building a system via the graphical user interface (GUI) (not recommended)

Both options shall be illustrated briefly in the following subsections.

2.4.2 Model setup via the script language

2.4.2.1 Script language

The HOTINT script language is a versatile tool which supports a variety of commands for (automatized) generation of multibody system components, such as bodies, loads, or constraints, along with the definition of initial conditions and material parameters. Moreover, variables and, in future versions, certain programming structures (e.g. loops or conditionals), can be used together with a set of mathematical operations similarly to other programming languages. Furthermore, just like the user-defined variables, also any HOTINT option or parameter may be specified via an input file (cf. 2.5.1). Details on the handling of variables and some general remarks with respect to the syntax of the script language are given below; for further information on the HOTINT file and folder structure see section 2.8.

Parser

The Parser used in HOTINT allows to use basic mathematical operations in the model files. Furthermore it is possible to copy parts of the data structure and work with previously defined variables. More details are provided in the following.

Data structure

All assignments in the model file of the form “left-hand side = right-hand side" where the left-hand side names the variable or object that is assigned a value (identifier), and the right-hand side is a number, vector or an evaluable expression (value). Between identifiers and values there may be as many spaces or tabs as desired by the user. However, line breaks need to be set according to the specification.
A valid right hand side entry - or variable name - may include alphanumeric characters and underscores, but no interpunctuation characters; comments start with the % character.

For example, the syntax for the definition of a floating point variable with the identifier “a" and the value 3.0 ist simply

```
a = 3.0
```

After this definition, “a" can be used and referred to at any point below in the script, for instance in the definition of another variable “b" combined with a basic mathematical operation

```
b = a
```

Optionally, the data entries can be arranged in named tree-structured containers which can be defined using curly braces. Such containers may hold any set of data entries, and, moreover, can be nested, i.e. can contain other containers as well. Access to each level and entry in these data structures is possible using the “."-operator, similar to the access to (nested) class members in Java or C++. See the following example for clarification:
Assume we want to describe a material – let us call it “m1" – using its elastic modulus “E" and Poisson ratio “nu", we could create a container named “m1" via

```
m1
{
  E = 1E11
  nu = 0.45
}
```

and access the parameters then via

```
m1.E
```

or

```
m1.nu
```

at any point in the file. Note that, within “m1", i.e., within one level in a container, the parameters specified there also may be referred to “directly", e.g. m1 E = 1E11 nu = 0.45 temp = 2*E
Now, if we had several materials “m1",“m2",“m3"..., as the one above, we could also define a nested structure “materials" – again a container – holding any of these material containers, for instance

```
materials
{
  m1
  {
      E = 1E11
      nu = 0.45
  }
  m2
  {
      E = 1.5E11
      nu = 0.47
  }
  m3
  {
      E = 2E11
      nu = 0.46
  }
}
```

where the access works analogously, e.g.

```
... = materials.m2.E
```

In summary, the entries on the right-hand side in an assignment can be of the following types, depending on the type of the left-hand side:

```
bool = yes                      % boolean can be 'yes' or 'no'
integer = 1                     % integer number
float = 0.628e1                 % floating point number
string = "text"                 % string variable
vector = [1.,2.,3.,4.]          % vector, with ',' as separator
matrix = [1.1,2.1;1.2,2.2]      % matrix, with ',' and ';' as separators
Container = other_Container     % entire tree
```

Constants and variables

As shown exemplarily above, it is possible to assign existing variables to new names. The variables on the left-hand side can be accessed by their name and/or location in the data structure. The Parser itself also includes intrinsic constants like pi.

```
a = 1
b = 2
SubContainer
{
  b = 12
  c = 13
}
roota = a                       % assign the content of variable a to roota
rootb = b                       % assign the content of variable b to rootb
subb = SubContainer.b           % assign the content of variable b in the
                                % SubContainer to subb
```

Operations

It is also possible to perform simple mathematical operations like adding, multiplying and accessing components on the right-hand side. These features only work on previously assigned variables of the same type.

```
a = 2
b = 3
vec = [1,2,3]
mat = [1,2;3,4]
% VALID OPERATIONS:
c = a+b                         % adding two numbers
d = a*b                         % multiplying the numbers
vec2 = vec + vec                % adding two vectors
two = vec[2]                    % access to component of a vector
three = vec[b]                  % access in succession
four = mat[2,2]                 % access to component of a vector
vec[2] = 7                      % access in right hand side expression

% NOT WORKING:
vec3 = 3*vec                    % type mismatch
scalar = vec*vec                % not implemented as operator
mat_succ = mat[ mat[1,1], mat[1,2]]
                                % not implemented succession with multiple ','
```

Built-in functions

Several mathematical functions are implemented in the Parser and can be used in right-hand side expressions. This feature includes

- power

```
a = sqr(3)                     % square
b = sqrt(a)                    % square root
c = 2\^3                       % power
```

- exponential and logarithm

```
h1 = exp(5)                    % exponential
h2 = ln(h1)                    % logarithm base e
h3 = log(1000)                 % logarithm base e
h4 = log10(1000)               % logarithm base 10
```

- trigonometric

```
e1 = sin(pi/2)                 % sinus function
f1 = cos(pi/2)                 % cosinus function
g1 = tan(pi/2)                 % tangens function
e2 = asin(1)
f2 = acos(1)
g2 = atan(1)
e3 = sinh(pi/2)
f3 = cosh(pi/2)
g3 = tanh(pi/2)
```

- unitarian operators and functions

```
b = -a                         % change sign
c = fact(10)                   % factorial
i1 = abs(-273.15)              % absolute value
i2 = fabs(b)                   % absolute value
d1 = round(1.61803399)         % round to nearest integer
d2 = floor(1.61803399)         % next interger lower or equal
d3 = ceil(1.61803399)          % next integer larger od equal
tam = transpose(mat)           % transpose a matrix
h = heaviside(a)               % heaviside function
```

- vectors

```
v = [1, 2]                     % vector v = [v1, v2, ...]
d4 = vabs(v)                   % sqrt((v1\^2+v2\^2+...))
d5 = varg(v)                   % atan2(v2, v1)
```

2.4.2.2 Model setup

Any consistent file written in the script language (“HOTINT data input file", with “.hid" filename extension; cf. section 2.8) can be loaded and used in HOTINT. In short, it can contain any setting of options for HOTINT itself (see section 2.5 or 3.13 for details), and fully describe the multibody system. On the other hand, if a model is loaded and edited, or created completely via the GUI (see the following subsection 2.4.3), and then saved to a file, the output again will be in terms of the script language. For a detailed description of all supported commands, as well as corresponding example code fragments for illustration, please refer to the reference manual under section 3.12. Sections 3.2–3.10, on the other hand, contain detailed information about all multibody system components available in HOTINT, i.e. various types of elements such as rigid bodies or structural finite elements such as ANCF beam elements, connectors, loads, sensors, and geometrical elements.

Concludingly, it should be pointed out that the best way to get to know how the whole thing works probably is – as already mentioned – to start and experiment with ready-to-use example files (see also 2.2.2), which are located in the folder documentation/examples in your HOTINT directory and for download at the homepage. See also the the minimal examples in the reference manual.

2.4.3 Model setup via the graphical user interface

The generation and setup of a multibody system via the GUI is more or less self-explanatory: Use the main menu entries “Edit" (cf. 2.3.5.4) and “Add Object" (cf. 2.3.5.3) to edit existing or add new components to the system, specify parameters, and define initial conditoins. The model can be saved – as model file in HOTINT script language – at any time. Before an object is added or edited via the GUI, the model is saved automatically. The resulting file is located in the application path and named model_asv.hmc.

However, note that the full functionality and flexibility is only accessible via the direct use of the script language.

For details concerning the settings for parameters of single multibody system components please refer to the HOTINT reference manual, sections 3.2–3.10.

2.5 Options Dialogs

2.5.1 Introduction

Via the Windows user interface a wide range of options can be specified to customize HOTINT, concerning, for instance, the graphics, solver or in- and output. The corresponding option dialogs are documented in the following; for a full and detailed listing of all available options refer to the reference manual, section 3.13.

Note that any of these options can be set just like any variable in a script language model file (cf. also subsection 2.4.2) by using its full data name (category + data name according to the options reference). For example, if you would like to specify a maximum time step of 5 ms within the model file, you would just add the line (cf. “TimeInt" in the SolverOptions 3.13.1)

```
SolverOptions.Timeint.max_step_size = 0.005
```

In case of several settings within SolverOptions – or at any other level (such as “Timeint") for that matter – you may use the syntax as with the nested “data containers" described in the subsection 2.4.2. See the following example for illustration:

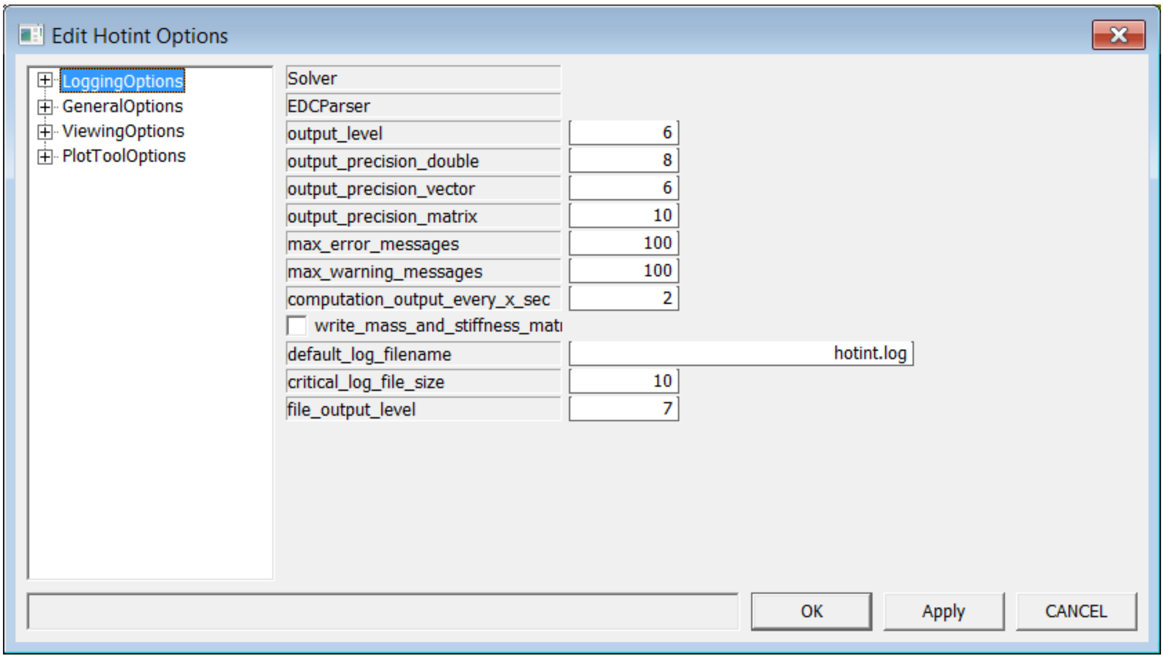
```
SolverOptions
{
  end_time = 1                           %1 second simulated time
  Timeint
  {
    max_step_size = 1e-5                 %max. step size for time integration
    min_step_size = 1e-3*max_step_size   %min. step size for time integration
  }
  Newton.max_modified_newton_steps = 20  %max. number of modified Newton steps
}
```

which would be equivalent to

```
SolverOptions.end_time = 1
SolverOptions.Timeint = max_step_size = 1e-5
SolverOptions.Timeint = min_step_size = 1e-3*max_step_size
SolverOptions.Newton.max_modified_newton_steps = 20
```

2.5.2 Hotint Options

Access: View → Edit Hotint Options



Solver
EDCParser
output_level 6
output_precision_double 8
output_precision_vector 6
output_precision_matrix 10
max_error_messages 100
max_warning_messages 100
computation_output_every_x_sec 2
☐ write_mass_and_stiffness_mat
default_log_filename hotint.log
critical_log_file_size 10
file_output_level 7

2.5.2.1 LoggingOptions

LoggingOptions	Specify which, how detailed, and in what intervals information concerning the model initialization and solution procedure should be written to the Output-Window and Log-File, respectively
Solver	Special configurations for log information concerning the solution procedure
EDCParser	Special configurations for log information concerning during the parsing of the model data file

2.5.2.2 GeneralOptions

Application	A set of general options concerning the application itself. See “Application" under 3.13.3 in the reference manual for details
Paths	Access and set paths of the executable, for the input of input data, and for video/single frame/image/PlotTool image exports
ModelFile	See “ModelFile" under 3.13.3 in the reference manual for details
Measurement	Choose units for angles and the legend and values of the contour plot
OutputWindow	Limit the maximum number of characters in the output window
SavedViewingOptions	Define the file names where viewing options shall be stored to when clicking one of the buttons ’1’, ’2’ or ’3’

2.5.2.3 ViewingOptions

Animation	Settings specifying how the animation via the Data Manager should be performed
Misc	Various settings concerning the redraw frequency during the simulation, and the thickness or size of points and lines
GeomElements	Settings concerning the GeomElements, e.g. line thickness
Origin	Choose if and how the origin of the coordinate system should be displayed
Grid	Specify and show a background coordinate grid
CuttingPlane	Detailed options for the configuration of up to two cutting planes
StandardView	Define standard views of the system via specification of rotation axes and corresponding angles
Bodies	Options specifying how bodies in general, and rigid bodies and particles in particular, should be drawn and tagged; also includes settings for velocity vectors
FiniteElements	Settings concerning the drawing and coloring of the contour plot, and of finite elements and corresponding meshes and nodes
Connectors	Options specifying if and how constraints should be displayed
Loads	Define if and how loads should be displayed
Sensors	Define if and how sensors should be displayed
OpenGL	Settings for lighting, light sources, transparency, shininess, and color intensity.
ApplicationWindow	Size and position of main window of HOTINT
DataManager	Settings concerning the data manager, e.g. how often the solution is stored.
OutputWindow	Settings concerning the output window (left of main window).
View3D	Define the perspective and sensitivity of mouse movements.

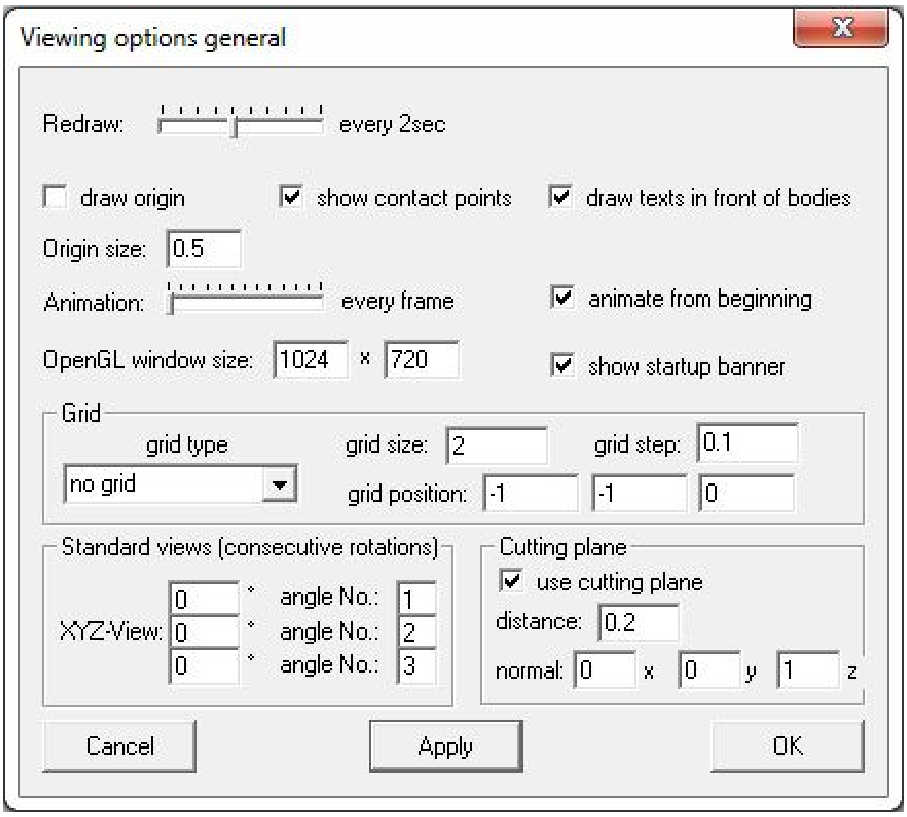
2.5.2.4 PlotToolOptions

PlotToolOptions	General setting for the Plot Tool (cf. also section 2.7), concerning redrawing, scaling, and some size factors for labeling and axis/tick styles
DataPoints	Settings for marking of data points
View	Configuration of size and position of the plot window and the plot itself

Watches	Initial size of sensor watch windows
Axis	Settings for ticks and labels for both x- and y-axis of the plots
Grid	Specification of line types for background coordinate grids in the plots
Legend	Specification if and where a legend should be shown
SavePictures	Options concerning the export of image files from a plot

2.5.3 Viewing Options

Access: View → Viewing Options



Redraw: every 2sec

☐ draw origin ☑ show contact points ☑ draw texts in front of bodies

Origin size: 0.5

Animation: every frame ☑ animate from beginning

OpenGL window size: 1024 x 720 ☑ show startup banner

Grid

grid type: no grid

grid size: 2

grid step: 0.1

grid position: -1 -1 0

Standard views (consecutive rotations)

XYZ-View: 0 ° angle No.: 1

0 ° angle No.: 2

0 ° angle No.: 3

Cutting plane

☑ use cutting plane

distance: 0.2

normal: 0 x 0 y 1 z

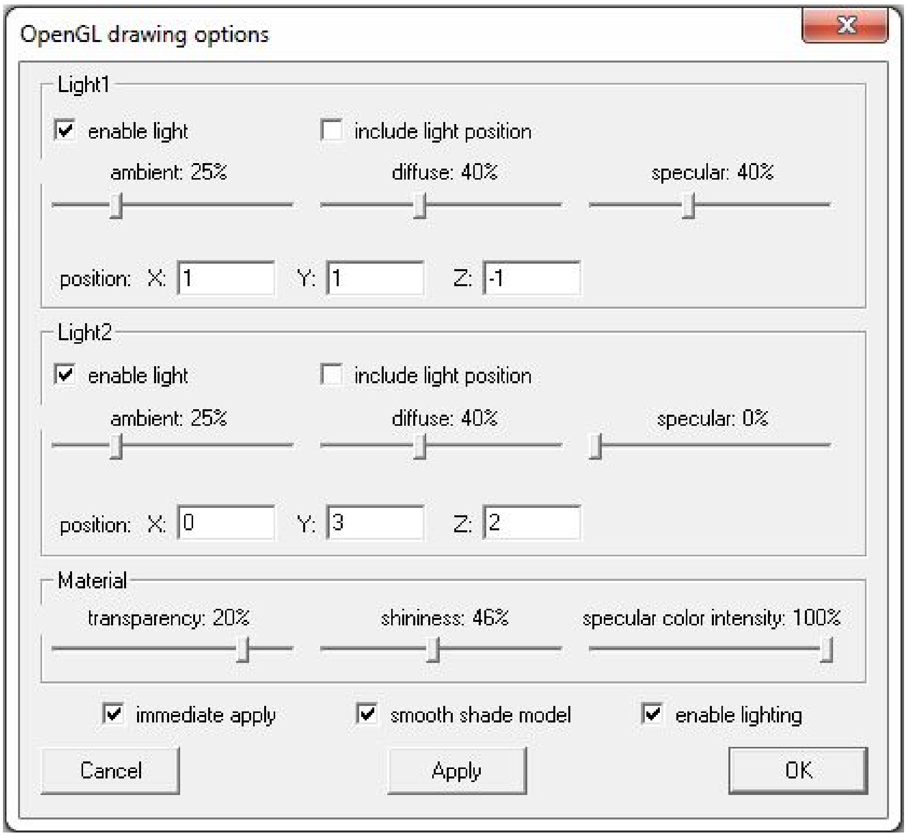
Viewing options allow changing some of the parameters for visualizing the multibody model:

redraw	Change the time between subsequent redraws of the model during the simulation in order to speed up the simulation
draw origin	Draw the origin (0,0,0) and the orientation of the global coordinate system
origin size	Length of the drawn axis of the origin
show contact points	If checked, contact points are shown

draw texts in front of bodies	This option will draw texts much closer to the viewer such that they are visible even if they are hidden in reality by an object. However, due to distortion, the texts might appear at slightly different positions.
animation	Your animation will run faster if you draw e.g. only every 10 or 50 frames of the stored computation steps
animate from beginning	Pressing the animation button will always move to the beginning of the simulation
OpenGL window size	For screen shots and animation, this lets you adjust the size of the visualization screen in pixels. Best results are obtained if you chose standard resolutions such as 640x480, 800x600, etc.
show startup banner	If checked, the startup banner is shown
grid	Chose a grid type (orientation), the grid size (length = width), grid step and a grid reference point in order to show a grid for determining positions of the selected model
standard views	The selection of these parameters allows you to define a standard rotation with respect to the global axis 1, 2 and 3 (= x,y,z) by certain angles. The standard view is x-horizontal and y-vertical, z points out of the x/y plane.
cutting plane	Define a cutting plane by its normal vector and distance from the origin in the direction of the normal; any part of the system lying beyond that plane (in direction of the normal) is cut, i.e. not displayed. A second cutting plane and additional configurations can be defined and accessed via the menu View → Edit Hotint Options → ViewingOptions → CuttingPlane.

2.5.4 OpenGL Drawing Options

Access: View → OpenGL Drawing Options



Light1

☑ enable light ☐ include light position

ambient: 25% diffuse: 40% specular: 40%

position: X: 1 Y: 1 Z: -1

Light2

☑ enable light ☐ include light position

ambient: 25% diffuse: 40% specular: 0%

position: X: 0 Y: 3 Z: 2

Material

transparency: 20% shininess: 46% specular color intensity: 100%

☑ immediate apply ☑ smooth shade model ☑ enable lighting

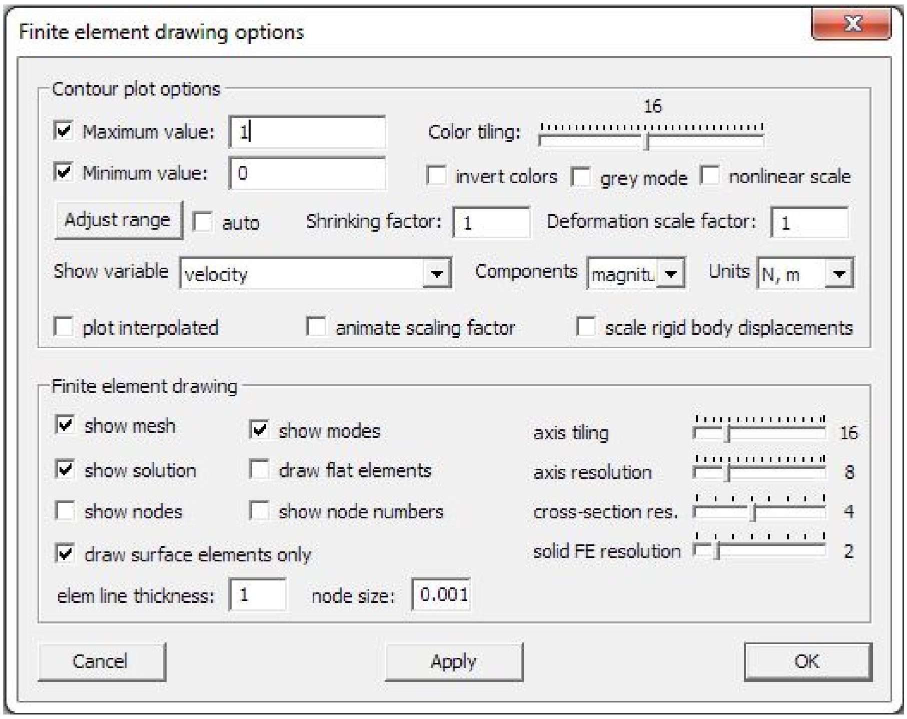
The OpenGL graphics includes some settings in order to customize the drawing. Yet it is not possible to choose the surface property of a single body, but the material is set for all bodies to the same values, like shininess, transparency, specular color. Sometimes a specific lighting model improves the visibility of an object or the understanding of its geometric complexity. Otherwise the default values can be kept.
There are two independent light sources included, it is possible to activate only one or both lights.

enable light	Enable the light source
include light position	Include light position in the computation of the intensity. If not checked, objects that are farther away from the light will have the same lighting conditions as near objects
ambient	Percentage of ambient light, the intensity of the light is independent of the direction of the light
diffuse	Percentage of diffuse light, the brightness is dependent on the position and orientation of the surface with respect to the light source
specular light	Percentage of specular light, creates highlight on surfaces like polished metal or mirror-like surfaces.
position	Position of the light source
transparency	The percentage defines the transparency of the material where 0% is not transparent and 100% is fully transparent. Note that the transparency is dependent on the order of the objects which are currently

	not sorted in HOTINT. This can cause strange transparency effects in meshed objects.
shininess	This factor defines the radius of shininess of the specular light, 100%=small radius, 0%= very large radius
specular color intensity	Defines the amount of specular color reflected by the material
immediate apply	If this is activated, all changes in the dialog are immediately applied to the graphics window
smooth shade model	Use this to activate smooth shading, which improves the drawing of round surfaces. Otherwise, flat shading is activated (piecewise flat polygons)
enable lighting	If not activated, the brightness is not depending on the position of the light with respect to the surface

2.5.5 Finite Element Drawing Options

Access: View → FE Drawing Options



Finite element drawing options

Contour plot options

☑ Maximum value: 1

☑ Minimum value: 0

Color tiling: 16

☐ invert colors ☐ grey mode ☐ nonlinear scale

Adjust range ☐ auto Shrinking factor: 1 Deformation scale factor: 1

Show variable velocity Components magnitu Units N, m

☐ plot interpolated ☐ animate scaling factor ☐ scale rigid body displacements

Finite element drawing

☑ show mesh ☑ show modes axis tiling 16

☑ show solution ☐ draw flat elements axis resolution 8

☐ show nodes ☐ show node numbers cross-section res. 4

☑ draw surface elements only solid FE resolution 2

elem line thickness: 1 node size: 0.001

2.5.5.1 Contour plot options

Maximum value	If activated, the maximum value of the contour plot is limited to the specified value (in the specified units)
Minimum value	If activated, the minimum value of the contour plot is limited to the specified value (in the specified units)
Adjust range	Auto-adjust the range of the contour plot

auto	If activated, the minimum and/or maximum value of the contour plot is chosen automatically, unless it is explicitly specified in the Minimum/Maximum value setting.
color tiling	The number of different colors in the contour iso-plot. The maximum is 32 different colors, a larger value leads to a continuous color
invert colors	The color bar is inverted
grey mode	Only black to white colors are used
nonlinear scale	A nonlinear scale of colors is used. This can be interesting for Mises comparison plots e.g. with edge singularities
Shrinking factor	The size of the finite elements is multiplied with this factor. Use a value of 1 for displaying the original size and e.g. 0.9 in order to display a reduced view of the elements
Deformation scale factor	A factor by which all deformations are magnified in the graphic representation. For better visualization of small deformations you may use a large scale factor
Show variable	The field variable chosen from this list is diplayed in the contour plot.
Components	If a non-scalar field variable has been chosen, here the absolute value (magnitude) or component of the field variable which should be displayed in the contour plot can be chosen.
Units	Select units for the chosen field variable.
plot interpolated	If activated, field variables defined on a finite element mesh are plotted interpolated in the contour plot
animate scaling factor	In order to view eigenmodes or static deformation, the scaling factor can be animated
scale rigid body displacements	If activated, all rigid body displacements are scaled by the factor specified in the field "Deformation scale factor" (in the graphic representation)

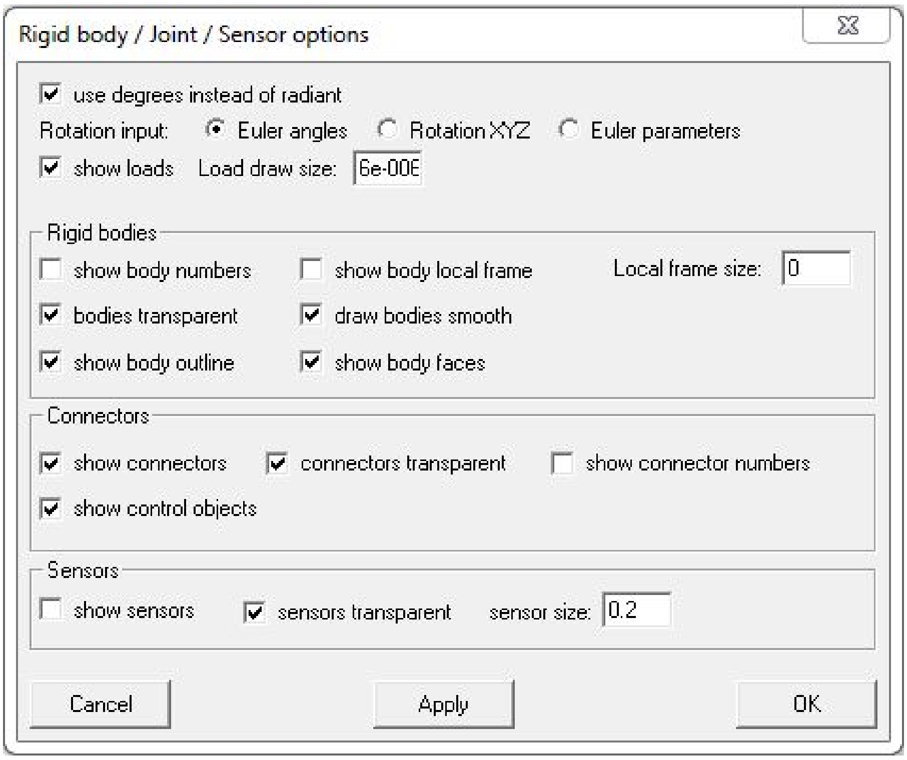
2.5.5.2 Finite element drawing

show mesh	Shows the mesh outlines
show modes	If checked, modes are shown via Chladni isolines
show solution	Shows the mesh surface
draw flat elements	If checked, draw plate elements as flat polygons, otherwise draw plate elements with specified thickness

show nodes	Shows the nodes of the mesh
show node numbers	Displays the numbers corresponding to the nodes
draw surface elements only	If checked, only finite elements on the surface of a mesh are drawn
elem line thickness	Line thickness for element outline
node size	Size of nodes
axis tiling	Tiling specifies the number of quadrangles to draw a curved beam or plate element in axial direction
axis resolution	Resolution specifies the number of quadrangles used to draw the contour solution of a beam or plate element in axial direction
cross-section resolution	Resolution specifies the number of quadrangles used to draw the contour solution of a beam or plate element within the transverse direction (discretization of the cross-section)
solid FE resolution	Resolution (tiling) used to approximate one solid finite element (triangle, quadrangle, hexahedral, tetrahedral, etc.)

2.5.6 Body / Joint Options

Access: View → Body/Joint Options



Rigid body / Joint / Sensor options

☑ use degrees instead of radiant

Rotation input: ● Euler angles ○ Rotation XYZ ○ Euler parameters

☑ show loads Load draw size: 6e-00E

Rigid bodies

☐ show body numbers ☐ show body local frame Local frame size: 0

☑ bodies transparent ☑ draw bodies smooth

☑ show body outline ☑ show body faces

Connectors

☑ show connectors ☑ connectors transparent ☐ show connector numbers

☑ show control objects

Sensors

☐ show sensors ☑ sensors transparent sensor size: 0.2

2.5.6.1 General

use degrees instead of rad.	Checked = use degrees (0° - 360°) instead of radiant (0 - 2π) for the input of angles and angular velocities. The stored values are always in radiant.
rotation input	Select input mode for spatial rotations: Euler angles = rotation about Z-X-Z, RotationXYZ = rotation about X-Y-Z, Euler parameters = direct input of 4 Euler parameters
show loads	If activated, all loads in the multibody system are shown
load draw size	Specification of the size of the displayed loads

2.5.6.2 Rigid Bodies

show body numbers	Checked = display element number of the body
show body local frame	Checked = draw local frame of body
local frame size	Drawing size of local body frame
bodies transparent	Checked = draw bodies transparent with factor defined in OpenGL options
draw bodies smooth	Interpolate GeomElement meshes with increased smoothness
show body outline	Checked = draw the outline (edges) of a body or GeomElement
show body faces	Checked = draw the surface of a body or GeomElement → if "show body outline" and "show body faces" is unchecked, the bodies are not drawn

2.5.6.3 Connectors

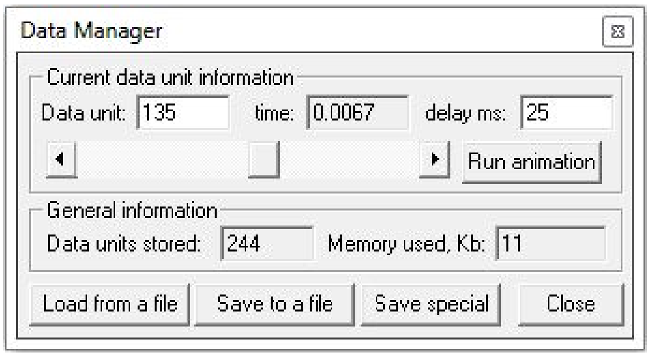
show connectors	Checked = draw connectors
connectors transparent	Checked = draw connectors transparent with a factor defined in OpenGL options
show connector numbers	Checked = display element number of the connectors
show control objects	Checked = control objects are drawn.

2.5.6.4 Sensors

show senors	Checked = show sensors
sensors transparent	Checked = draw sensors transparent with factor defined in OpenGL options
sensor size	Size of sensor local axes

2.5.7 Data Manager

Access: View → Show Data Manager



Data unit: 135
time: 0.0067
delay ms: 25
Data units stored: 244
Memory used, Kb: 11

The Data Manager is used to draw the solution at certain time instants where the data has been stored internally. The data is stored either in internal memory or written to the hard disk in the output directory, depending on what was specified for the option Solver Options → Solution → store_data_to_files. Make sure to activate this option in cases where the simulation data would exceed the available main memory. The sliding bar can be used to view certain stored data units and analyze the solution, which is possible even during computation. It is preferable to set the redraw time of the model view very high (→ Viewing options → Redraw) in order to be able to smoothly animate the solution during a long computation. The analysis of the solution during the computation can help to detect model input or convergence errors at an early stage or allows you to run your simulation infinitely (set end time e.g. to 1e6) and to stop the simulation at the point of your consideration.

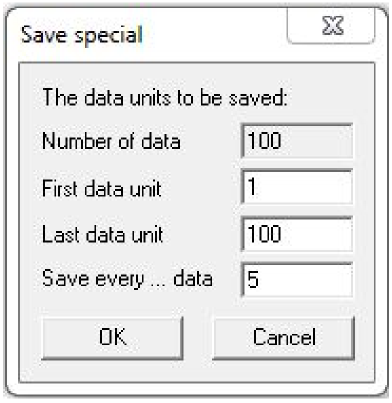
The button "Run animation" starts the animation either from the beginning (data unit 1) if Display Options → Animate from beginning is set, or otherwise from the current position of the slider bar.

There are two data formats: The .txt format which stores data in pure text (space-separated data):

line 1: Version identifier
line 2: checksums, first value = size of data, second value = checksum
line 3: number of available data units
line 4: first line of data unit: time, size of data, number1, number2, ...
The .dat format uses windows serialize functions and can not be edited.

Data unit	Actual data unit drawn
time	Actual time instant drawn
delay	Delay used between frames when running animations
Run animation	Start animation
Load from a file	Load a stored solution for animation. Note that only the stored solution that belongs to the same multibody model can be loaded.
Save to a file	Save the data units into a file. You can choose to save in .txt format which saves the data of each time point in one line (row), or in .dat format. The dat format is considerably faster and smaller in size.

Save special	Save selected data units into a file: specify first data unit, last data unit and the increment between stored data units. The data can be stored in .txt, .dat and also as .sol file. Choosing the same number for the first and last data unit allows to use this solution as a .sol solution which can be used as an initial vector for a further computation.



The data units to be saved:

Number of data 100

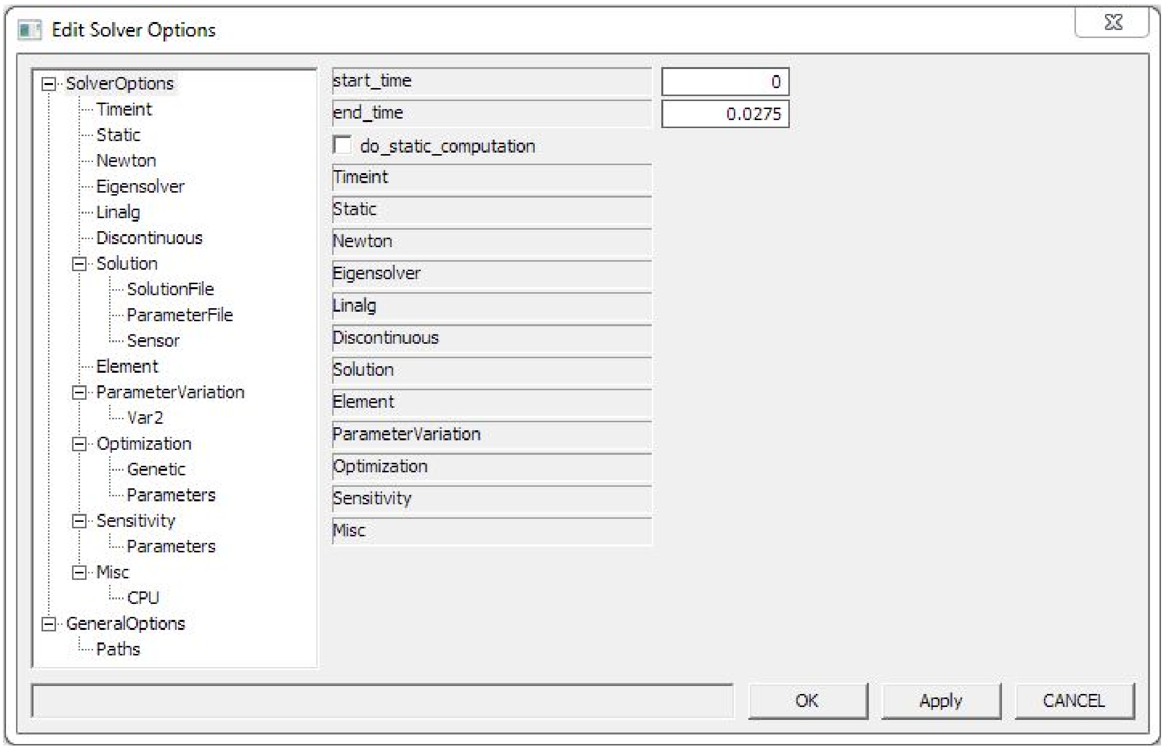
First data unit 1

Last data unit 100

Save every ... data 5

2.5.8 Solver Options

Access: Computation → Edit Solver Options



start_time 0

end_time 0.0275

☐ do_static_computation

2.5.8.1 SolverOptions

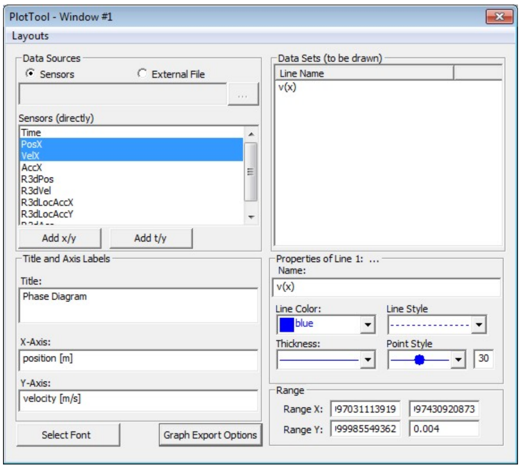
SolverOptions	Set start and end time, and choose between dynamic and static computation
Timeint	Settings concerning the time integration, such as minimum and maximum step size, the maximum index of the differential algebraic equations, or the time integration scheme

Static	Settings of the static solver, e.g. concerning load increments
Newton	Parameters which specify the accuracy goal of the Newton solver, and other options regarding the latter (e.g. settings for numerical differentiation, maximum number of modified or full Newton steps,...)
Eigensolver	Settings concerning the modal (eigensystem) analysis, such as number of eigenvalues and maximum iterations, the accuracy goal, etc.
Linalg	Specify whether to use s sparse solver for the solution of the linear systems in the Newton procedure
Discontinuous	Settings regarding discontinuous systems (e.g. due to friction, contact, etc.)
Solution	A set of options defining how, in which intervals, and where the solution data and data of the parameter variation procedure should be stored
Element	Specify whether to store intermediate finite element matrices, and to compute the Jacobians elementwise
ParameterVariation	Settings concerning the parameter variation procedure: (de)activate, inital and final value, arithmetic or geometric step size, and the path and variable name of the parameter to be varied in the model data input file
Optimization	Settings regarding the optimization procedure: (de)activate, choice of method, settings for the respective parameters
Sensitivity	Specify if and how the sensitivity of sensor values with respect to certain parameters should be analysed
Misc	Various settings regarding, for instance, a default model data file, or multithreading in the computation

2.6 Data visualization and graphics export

2.7 Visualization Tool

In HOTINT it is possible to visualize the simulation data with an integrated tool.
The Visualization Tool consists of two windows, one containing the most important control elements and a separate window for the plot itself. Both Windows can be blinded out if required.
One can display the data directly from the current simulation run or from a file from a previous simulation. The data can be displayed as y(t) using a single data set and also as y(x) when two datasets are combined. The main advantages of using an integrated tool are that we are able to display the data on the fly and create serviceable graphs automatically.
As in most visualization tools each data line can be assigned a color, linestyle and a marker shape. Together with title, labels, positions and other options the graphs' layout information may be stored for later use. As mentioned above a dialog provides access to the frequently used options. To keep the dialog slim, for both windows an additional context menu is implemented and some hardly ever used options are only available via the full options menu.
The tool is intended for visualization only, so we do not intend to include curve fitting routines to it. Still it is possible to create a consistent dataset for mathematical functions and add those to the graph. For a deep analysis of the result like curve fitting an external program must be used.
The model itself can be programmed such that for selected sensor values a visualization window is automatically created when the model is loaded. For simulations with multiple cycles it is possible to generate graphs with identical properties for comparison.



PlotTool - Window #1

Data Sources: ● Sensors ○ External File

Sensors (directly): Time, PosX, VelX, AccX, R3dPos, R3dVel, R3dLocAccX, R3dLocAccY

Data Sets (to be drawn) – Line Name: v(x)

Title: Phase Diagram

X-Axis: position [m]

Y-Axis: velocity [m/s]

Properties of Line 1: ...

Name: v(x)

Line Color: blue

Point Style: 30

Range X: ı97031113919 ı97430920873

Range Y: ı99985549362 0.004

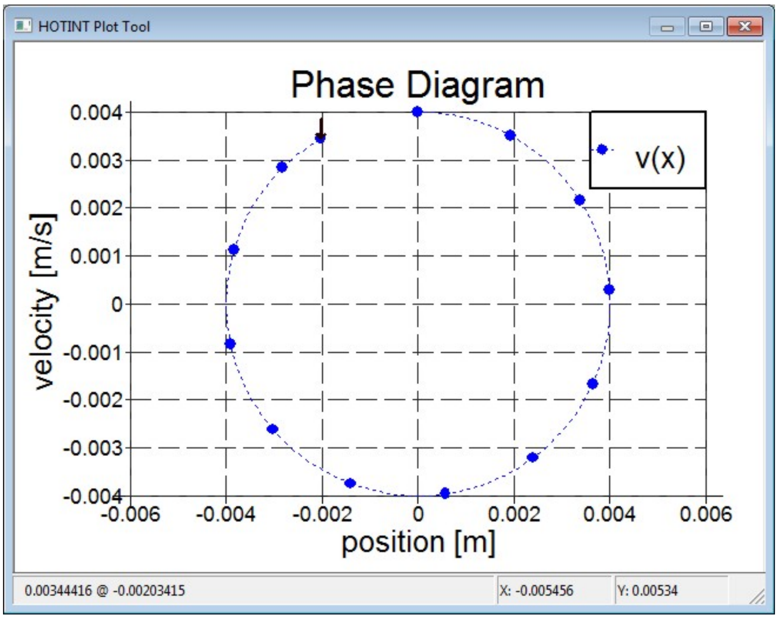
Figure 2.2: PlotToolDialog

2.7.0.2 Data Sources

The top section of the Dialog is dedicated to the selection of the data source. The left side allows to pick either the Sensors of the current model or an external (solution) file, most likely a soulition file from an other computation. The right part displays the available datasets. With the Buttons any highlighted item in the left list can be added to the right list of drawn lines. It is possible to plot a line over time (T/Y), but also combining two sensors for a (X/Y) graph. In this case exactly two lines must be selected in the list.

2.7.0.3 Graph Window

The middle section of the Dialog controls the content of the graph window, on the left side the caption and axis labels as well as the range of the plotted data can be chosen. On the right hand side the proberties of an individual line can be changed. Note: the line style can only be changed for thin lines (restriction from Windows Draw function). The general options control the redraw intervals and whether the range is adapted to the full range during a computation.



0.00344416 @ -0.00203415

X: -0.005456

Y: 0.00534

Figure 2.3: PlotToolGraph. Displays a datasets over time

2.7.0.4 Export

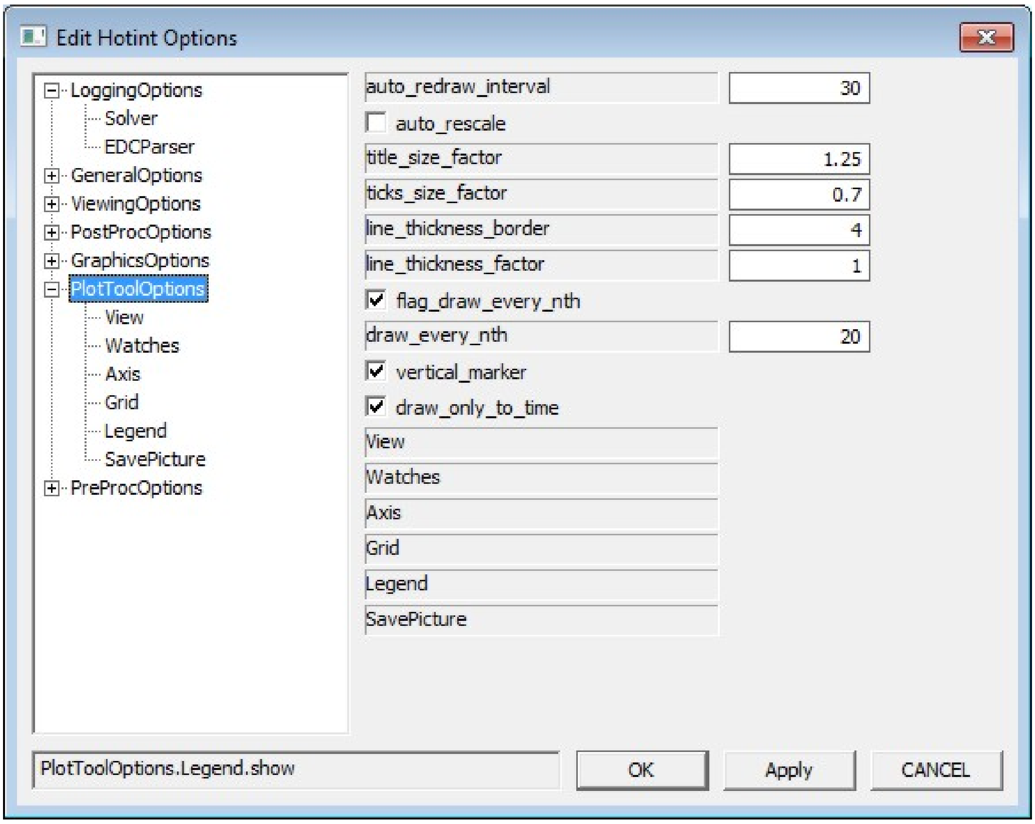
It is possible to export the content of the graph window to a file. Destination folder and filename can be defined in the textboxes. The resolution and all formats for the output can also be chosen.

2.7.0.5 Other Buttons

The remaining individual Buttons in the Dialog have the following effect:

Button	**description**
Show Graph	reactivates the Graph Window
Hide	hides the control dialog (reactivate in the Status Bar of the Graph Window)
Scale Graph	computes the range of the full dataset and rescales the axes accordingly
Redraw	performs a redraw operation manually (considers auto-rescale flag)
Axis Equal	forces equal scaling of both axes (mostly used for X-Y-Plots)
Print Graph	print dialog for the Graph Window
Update Options	Applies changes made in the HOTINT Options Dialog

The options available in control part of the dialog are only a selection the entire set. Many more are available in the HOTINT Options Dialog, in the subtree PlotToolOptions.



auto_redraw_interval	30
☐ auto_rescale	
title_size_factor	1.25
ticks_size_factor	0.7
line_thickness_border	4
line_thickness_factor	1
☑ flag_draw_every_nth	
draw_every_nth	20
☑ vertical_marker	
☑ draw_only_to_time	

PlotToolOptions.Legend.show

Figure 2.4: PlotToolOptions. In this Dialog many settings for the Graph can be done here, e.g. sizes, grid, ...

2.7.1 How to record a video

In order to create a video of your simulation perform the following steps:

- Run the simulation
- Be sure that enough data is stored in data manager
- Create a folder where the image files shall be stored
- Viewing options: set resolution to the desired value (e.g. 1024x768)
- Set all drawing options (with or without mesh, sensors, loads, etc.)
- Remove (drag+drop) all windows (data-manager, options-dialog, etc.) from the main window
- Click on the video–camera button (cf. subsection 2.3.4) to open the “Record frames"-dialog
- Once you have activated the image recorder, images are written at every update of the drawing window, even when the simulation has been stopped and you just resize or move the window
- When you are setting the path where the images will be stored, be sure that the folder already exists and that your path ends with a backslash (e.g. D:\images\ and not D:\images)

- Choose the desired image file format (JPEG, BMP, or PNG)
- Click ”Run animation” in the data manager
- Image files are now stored in the specified folder
- Use VideoMach, VirtualDub or comparable software to create a video from the single video frames

Additional hints:

- For video frames export, it is recommended to turn off any screen-saver, start your simulation (or load it from the database) and do not touch it until it has been finished.
- Usually it is preferable to run the simulation first and then use the stored data for the export of images. The whole procedure normally takes a several minutes, which, of course, depends on the complexity of the scene (e.g. number of elements) and the number of video frames.
- Clicking on the button left from the video–camera button lets you store single images into the directory specified above (see also subsection 2.3.4).
- If you are using Windows 7 you have to switch off “aero-design".

2.8 HOTINT File and Folder Structure

In this chapter, the file structure for saving multibody system models is described. The multibody system can be defined in an editable (".hid") format which allows the editing and creation of such files manually or automatically with external programs. However, one needs to be cautious when creating such files, because errors might lead to unexpected results!

The best way to get to know the file structure is to open an existing example file. Details on the HOTINT script language used in those files are provided in section 2.4.2.1.

2.8.1 Input Files

The new version of a text-file containing script language is called Hotint Input Data file - with file extension (".hid"). The file can be opened via the menu with "Open MBS". The filename is then stored in the variable
"GeneralOptions.ModelFile.hotint_data_filename". Using the button "Reload MBS" it is possible to open this model again, which allows the user to edit the model in an editor and check the correct implementation with just one click. Alternatively, the Hotint Input Data file can be committed to "hotint.exe" by the drag & drop function of the mouse. If the filetype (".hid") is linked with the application "hotint.exe", the Hotint Input Data file can be opened also by doubleclick of the mouse. A third variant to commit the Hotint Input Data file to HOTINT is to commit the Hotint Input Data file in the DOS-command line e.g. "hotint.exe filename.hid". In all three cases, the directory and filename is stored in the previously described Hotint Options[1]. The input file has to contain the variable "HOTINT_data_file_version" before the first command. HOTINT uses this variable to check, if the (old) input file still can be used with the current (new) version of HOTINT.

2.8.2 Folder Structure

The paths are collected in the Options "GeneralOptions.Paths". Most of them are located in the dialog "Edit Hotint Options":

- Application path: path of the application ("hotint.exe").
- Record frames path: path for storage of single frames for creating animations (modify in dialog "Video frames recording/Path to the image").
- Hotint input data path: path of the Hotint Data Input file (".hid").
- Sensor output path: path of the solution files from sensors (in dialog "Edit Solver Options").

---

[1] Note: the Include-command of the script language searches a file with absolute paths and afterwards relative to the previously described path of the Hotint Data Input file.

Chapter 3

HOTINT Reference Manual

3.1 Preface

In this reference manual all available objects and options are described.

3.1.1 Examples

If there is provided a short example for an object, keep in mind that the examples may not have any physical meaning. The examples just show how to add the object to the system.

3.1.2 Data objects

The description of each object contains a table called Data objects. These are the variables, that can be changed in the GUI or set in the script language. Variables marked with **R** are **readonly** and can not be changed by the user.

3.1.3 Observable FieldVariables

If an object provides field variables, they are listed in the documentation of the object. How to measure these variables with a FVElementSensor is described in section 3.9.1.

3.1.4 Observable special values

If an object provides special (internal) values, they are listed in the documentation of the object. How to measure these variables with a ElementSensor is described in section 3.9.2.

3.1.5 Controllable special values

If an object provides special (internal) values, that can be changed during runtime, they are listed in the documentation of the object. How to change these variables with a IOElementDataModifier is described in section 3.4.16.

3.2 Element

These elements are available:

- Mass1D, 3.2.1
- Rotor1D, 3.2.2
- Mass2D, 3.2.3
- Rigid2D, 3.2.4
- Mass3D, 3.2.5
- NodalDiskMass3D, 3.2.6
- Rigid3D, 3.2.7
- Rigid3DKardan, 3.2.8
- Rigid3DMinCoord, 3.2.9
- LinearBeam3D, 3.2.10
- RotorBeamXAxis, 3.2.11
- ANCFBeamShear3DLinear, 3.2.12
- ANCFBeamShear3DQuadratic, 3.2.13
- ANCFBeam3DTorsion, 3.2.14

Note:
In HOTINT several classes are treated as 'elements'. Connectors and control elements are also 'elements', and can therefore be edited and deleted in the GUI with the menu items of the elements.
In the script language the command `AddElement` is just available for the elements in the list above, but not for connectors or control elements.

3.2.1 Mass1D

Short description

A point mass in one dimensions with 1 position coordinate. The computation of the dynamics of the point mass is extremely simple. The Mass1D can be used for a lot of applications which can be represented by the same type of equations. If you interpret the 'mass' to be 'moment of inertia' and the 'position' to be 'angle', then you can realize a 1D rotatory element as well.

Degrees of freedom

1 degree of freedom: the position in x-direction

Geometry

The global position p_{glob} of a local point p is computed as

$$p_{glob} = p_0 + A\left(\begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} + p\right) \tag{3.1}$$

with the reference_position p_0 and the rotation_matrix A.

Equations

$$m\ddot{x} = F \tag{3.2}$$

with the mass m and the force F.

Limitations

The mass has no rotations, thus external moments can not be applied. The transformation of local to global coordinates is based on a translation, e.g. the global mass position is added to the local coordinates.

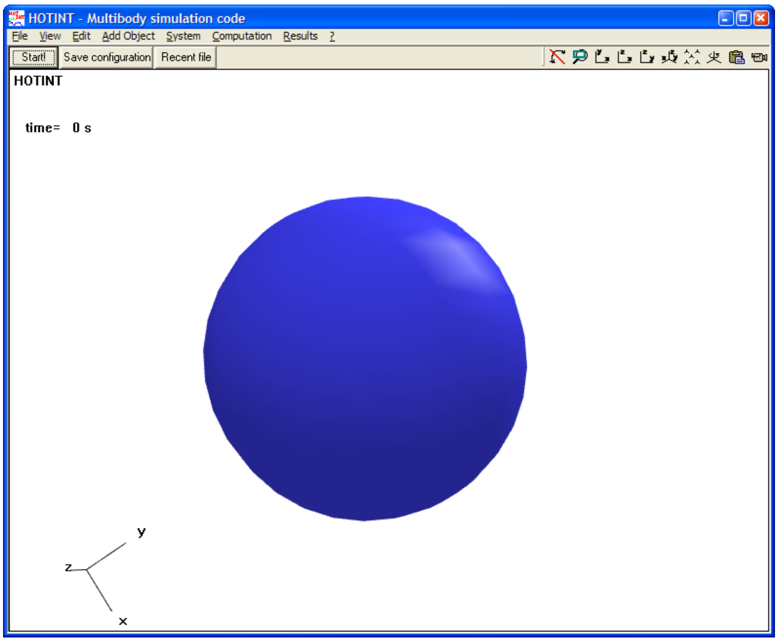

HOTINT - Multibody simulation code
File View Edit Add Object System Computation Results ?
Start! Save configuration Recent file
HOTINT
time= 0 s
y
z
x

Figure 3.1: Mass1D

Data objects of Mass1D:

Data name	type	R	default	description
element_type	string		"Mass1D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Mass1D"	name of the element
element_number	integer	R	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty

Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics.drawing_tiling	integer		8	tiling of circle/sphere to represent Mass1D; the drawing_tiling should be set small in order to improve efficiency, but large for nice graphical represenations
Graphics.radius	double		0.1	drawing radius of mass
Graphics. reference_position	vector		[0, 0, 0]	Reference point for transformation of 1D objects to 3D; p = [X, Y, Z]
Graphics.rotation_matrix	matrix		[1, 0, 0; 0, 1, 0; 0, 0, 1]	Rotation matrix for transformation of 1D objects to 3D
Initialization				
Initialization. initial_position	vector		[0]	initial values for position [x]
Initialization. initial_velocity	vector		[0]	initial values for velocity [v]
Physics				
Physics.mass	double		0	total mass of point mass

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, magnitude
displacement	x, magnitude
velocity	x, magnitude
acceleration	x, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-2
Internal.second_order_variable	second order variables of the element. range: 1-1
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-1

Suitable Connectors:

The following connectors can be used to constrain the element:
CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, FrictionConstraint, 3.3.8, Contact1D, 3.3.9,

Example

see file Mass1D.txt

```
force
{
  load_type = "GCLoad"
  load_value= 1
}
nLoad=AddLoad(force)

Element1
{
  element_type= "Mass1D"
  loads= [nLoad]
  Physics.mass= 1
}
nElement = AddElement(Element1)

senspos
{
  sensor_type= "FVElementSensor"
  element_number= nElement
  field_variable= "position"
  component= "x"
}
AddSensor(senspos)
```

3.2.2 Rotor1D

Short description

A rotor with 1 degree of freedom (the rotation). Mathematically implemented like Mass1D but different geometric representation.

Degrees of freedom

1 degree of freedom: the rotation

Geometry

The global position p_{glob} of a local point p is computed as

$$p_{glob} = p_0 + A_0 A p \tag{3.3}$$

with the reference_position p_0, the constant rotation_matrix A_0 and the non-constant rotation matrix

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \tag{3.4}$$

Equations

$$I\ddot{\varphi} = M \tag{3.5}$$

with the moment of inertia I and the torque M.

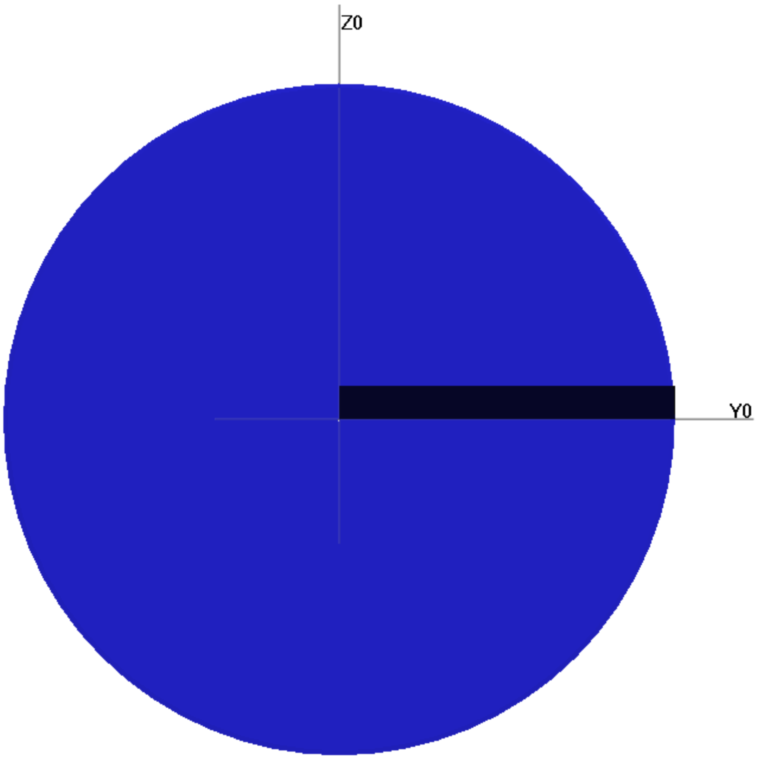


Figure 3.2: Rotor1D is represented as rotating disc.

Data objects of Rotor1D:

Data name	type	R	default	description
element_type	string		"Rotor1D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rotor1D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics.use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element

Graphics.show_element	bool		1	Flag to draw element
Graphics. reference_position	vector		[0, 0, 0]	Reference point for transformation of 1D objects to 3D; p = [X, Y, Z]
Graphics.rotation_matrix	matrix		[1, 0, 0; 0, 1, 0; 0, 0, 1]	Rotation matrix for transformation of 1D objects to 3D
Graphics.radius	double		0.1	radius of rotor
Graphics.length	double		0.2	length of rotor
Initialization				
Initialization. initial_rotation	vector		[0]	initial value for rotation
Initialization. initial_angular_velocity	vector		[0]	initial value for angular velocity
Physics				
Physics. moment_of_inertia	double		0	mass moment of inertia in kg*m*m

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
bryant_angle	x, magnitude
angular_velocity	x, magnitude
angular_acceleration	x, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-2
Internal.second_order_variable	second order variables of the element. range: 1-1
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-1

Suitable Connectors:

The following connectors can be used to constrain the element:
CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, FrictionConstraint, 3.3.8, Contact1D, 3.3.9,

Example

see file Rotor1D.txt

```
force
{
  load_type = "GCLoad"
  load_value= 1
}
nLoad=AddLoad(force)

Element1
{
  element_type= "Rotor1D"
  loads= [nLoad]
  Physics.moment_of_inertia= 1
}
nElement = AddElement(Element1)

senspos
{
  sensor_type= "FVElementSensor"
  element_number= nElement
  field_variable= "bryant_angle"
  component= "x"
}
AddSensor(senspos)
```

3.2.3 Mass2D

Short description

A point mass in two dimensions with 2 position coordinates. The computation of the dynamics of the point mass is extremely simple, thus the Mass2D can be used for many body simulations (e.g. particles).

Degrees of freedom

2 degrees of freedom: the position in 2 coordinates

Equations

$$m\ddot{\boldsymbol{x}} = \boldsymbol{F} \tag{3.6}$$

Limitations

The mass has no rotations, thus external moments can not be applied. The transformation of local to global coordinates is based on a translation, i.e., the global mass position is added to the local coordinates.

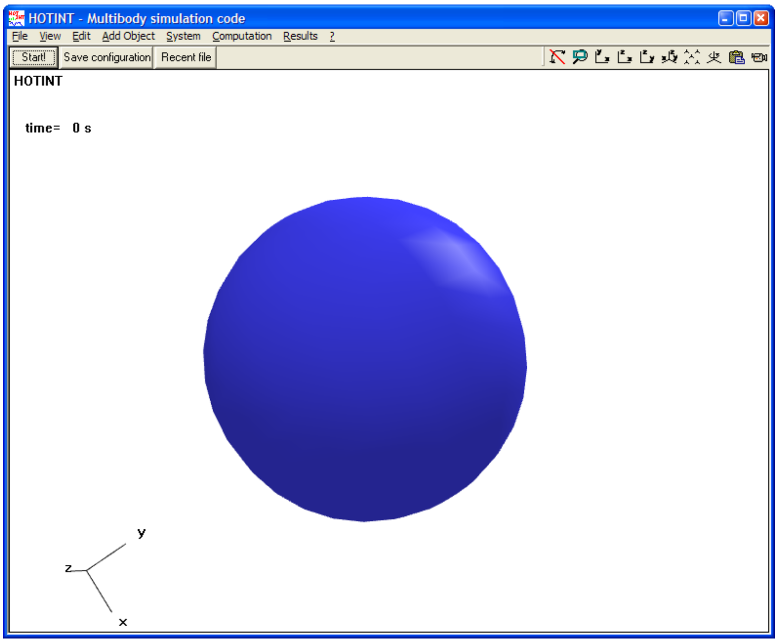

HOTINT - Multibody simulation code
File View Edit Add Object System Computation Results ?
Start! Save configuration Recent file
HOTINT
time= 0 s
y
z
x

Figure 3.3: Mass2D

Data objects of Mass2D:

Data name	**type**	**R**	**default**	**description**
element_type	string		"Mass2D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Mass2D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics.use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics.reference_position	vector		[0, 0, 0]	Reference point for transformation of planar objects to 3D; p = [X, Y, Z]
Graphics.rotation_matrix	matrix		[1, 0, 0; 0, 1, 0; 0, 0, 1]	Rotation matrix for transformation of planar objects to 3D
Graphics.drawing_tiling	integer		8	tiling of circle/sphere to represent Mass2D; the drawing_tiling should be set small in order to improve efficiency, but large for nice graphical representations
Graphics.radius	double		0.1	drawing radius of mass
Initialization				
Initialization.initial_position	vector		[0, 0]	initial values for position [x,y]
Initialization.initial_velocity	vector		[0, 0]	initial values for velocity [vx,vy]

Physics				
Physics.mass	double		0	total mass of point mass

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, magnitude
displacement	x, y, magnitude
velocity	x, y, magnitude
acceleration	x, y, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-4
Internal.second_order_variable	second order variables of the element. range: 1-2
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-2

Suitable Connectors:

The following connectors can be used to constrain the element:
CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, SpringDamperActuator2D, 3.3.19, PointJoint2D, 3.3.20,

Example

see file mass2D.txt

```
Load1
{
  load_type= "GCLoad"           % generalized force (here: actual force)
  generalized_coordinate= 2     % corresponding generalized coordinate
                                % (here: y-direction)
  load_value= -0.02
}
nLoad = AddLoad(Load1)
```

```
Element1
{
  element_type= "Mass2D"
  loads= [nLoad]
  Initialization.initial_position= [0, 1]
  Physics.mass= 1
}
nElement = AddElement(Element1)

Sensor1
{
  name= "global y-position"
  sensor_type= "FVElementSensor"
  element_number= nElement
  field_variable= "position"
  component= "y"
}
AddSensor(Sensor1)
```

3.2.4 Rigid2D

Short description

A rigid body in 2D.

Degrees of freedom

The first 2 degrees of freedom are those describing the position in the xy-plane. The rotation around the local z-axis is parameterized with the third degree of freedom.

Geometry

The center of gravity, S, is defined by the vector initial_position, which is in global coordinates. The rotation of the body-fixed local coordinate system w.r.t. the global coordiante system is defined by the variable initial_rotation.
In order to define the position of a point P of the element, e.g. for connectors or sensors, the local coordinate system is used. The reference point is the center of mass, S, so the values of the local coordinates can be positive or negative.

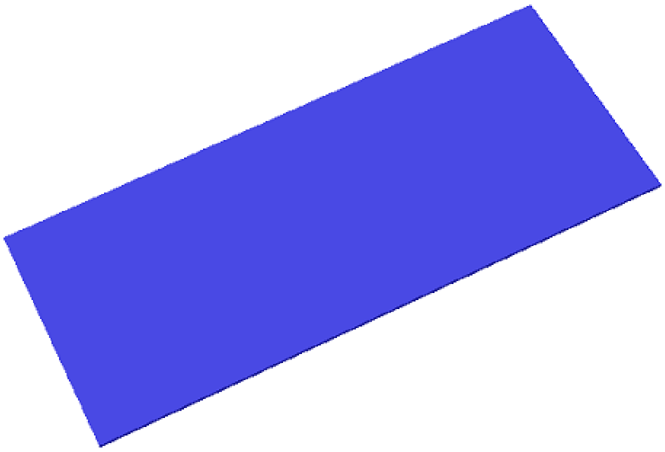


Figure 3.4: Rigid2D

Data objects of Rigid2D:

Data name	type	R	default	description
element_type	string		"Rigid2D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rigid2D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics. reference_position	vector		[0, 0, 0]	Reference point for transformation of planar objects to 3D; p = [X, Y, Z]
Graphics.rotation_matrix	matrix		[1, 0, 0; 0, 1, 0; 0, 0, 1]	Rotation matrix for transformation of planar objects to 3D
Graphics. body_dimensions	vector		[0.1, 0.1, 0.01]	Dimensions of a regular cube [L_x, L_y, (L_z)]
Physics				
Physics. moment_of_inertia	double		1.67e-007	[I_ZZ]
Physics.mass	double		0.0001	mass of the body in kg
Initialization				
Initialization. initial_position	vector		[0, 0]	[X, Y]
Initialization. initial_velocity	vector		[0, 0]	[vX, vY]
Initialization. initial_rotation	vector		[0]	rotation in rad

Initialization. initial_angular_velocity	vector		[0]	Angular velocity in rad/s

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, magnitude
displacement	x, y, magnitude
velocity	x, y, magnitude
bryant_angle	x, magnitude
angular_velocity	x, magnitude
acceleration	x, y, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-6
Internal.second_order_variable	second order variables of the element. range: 1-3
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-3

Suitable Connectors:

The following connectors can be used to constrain the element:
CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, SpringDamperActuator2D, 3.3.19, PointJoint2D, 3.3.20,

Example

see file Rigid2D.txt

```
L_x = 0.10 % length
L_y = 0.20 % width
L_z = 0.01 % height (for drawing and computation of mass)
density= 7850

myRigid2D % add rigid body
```

```
{
  element_type= "Rigid2D"  %specification of element type.
  name= "my first two-dimensional rigid body"  %name of the element
  Graphics.body_dimensions = [L_x, L_y, 0]
  Physics
  {
    mass= density*L_x*L_y*L_z
    moment_of_inertia= 1.0/12.0*mass*(L_x^2+L_y^2)
  }
  Initialization
  {
    initial_position= [0, 0]  %[X, Y]
    initial_rotation= [0.0]  % rot1_Z in rad
    initial_velocity= [0, 0]  %[X, Y]
    initial_angular_velocity= [pi*0.5] %rad/s
  }
}
nElement = AddElement(myRigid2D)
```

3.2.5 Mass3D

Short description

A point mass in three dimensions with 3 position coordinates. The computation of the dynamics of the point mass is extremely simple, thus the Mass3D can be used for many body simulations (e.g. particles).

Degrees of freedom

3 degrees of freedom: the position in 3 coordinates

Limitations

The mass has no rotations, thus external moments can not be applied. The transformation of local to global coordinates is based on a translation, e.g. the global mass position is added to the local coordinates.

Data objects of Mass3D:

Data name	type	R	default	description
element_type	string		"Mass3D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Mass3D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element

Graphics.show_element	bool		1	Flag to draw element
Graphics.drawing_tiling	integer		6	tiling of circle/sphere to represent Sphere
Graphics.radius	double		0.1	drawing radius of mass
Initialization				
Initialization. initial_position	vector		[0, 0, 0]	coordinates for initial position of mass [X Y Z]
Initialization. initial_velocity	vector		[0, 0, 0]	coordinates for initial velocity of mass [X Y Z]
Physics				
Physics.mass	double		0	total mass of point mass

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
acceleration	x, y, z, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-6
Internal.second_order_variable	second order variables of the element. range: 1-3
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-3

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17,

Example

see file AddElement.txt

`emptyMass3D`

```
{
  element_type = "Mass3D"
  Physics.mass= 1
}
nElement = AddElement(emptyMass3D)
```

3.2.6 NodalDiskMass3D

Short description

This is a disk mass for the purpose of rotordynamics applications and should be used together with the RotorBeamXAxis element.

Nodes

The DOF of the disk element are stored in a node. To create a new disk element the user has to define a 'Node3DR123' node. This node type has 6 DOF. The first 3 DOF describe the node displacement (x, y, z) w.r.t local rotor element coordinate system, the last 3 DOF are angles of rotation (ϕ_x, ϕ_y, ϕ_z) w.r.t local rotor element coordinate system. The rotation about the local x-axis is considered as large, the rotations about the local y and z-axes are considered as small (linearized angles).

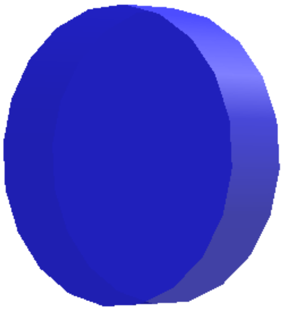


Figure 3.5: NodalDiskMass3D

Data objects of NodalDiskMass3D:

Data name	type	R	default	description
element_type	string		"NodalDiskMass3D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"NodalDiskMass3D"	name of the element
element_number	integer	R	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty

Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics.drawing_tiling	integer		6	tiling of circle/sphere to represent Sphere
Graphics.thickness	double		0.1	drawing thickness of disk mass
Graphics.radius	double		0	drawing radius of disk mass
Physics				
Physics. full_mass_matrix	bool		1	set to 1 if influence of tilted mass should be considered in the mass matrix
Physics. moment_of_inertia	vector		[1, 1, 1]	moments of inertia of the disk
Physics.mass	double		0	total mass of disk
node_number	integer		1	node number to which the mass refers

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
velocity	x, y, z, magnitude
acceleration	x, y, z, magnitude
angular_velocity	x, y, z, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-12
Internal.second_order_variable	second order variables of the element. range: 1-6
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-6

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file NodalDiskMass3D.txt

```
% define a node

node
{
  node_type = "Node3DR123"
  Geometry
  {
    reference_position = [0,0,0]
    reference_rot_angles = [0,0,0]
  }
}
n = AddNode(node)

disk
{
  element_type= "NodalDiskMass3D"
  Graphics.radius= 0.2 %radius
  Physics
  {
    moment_of_inertia= [1, 1, 1]  %moments of inertia
    mass= 1  %total mass
  }
  node_number= n  %node number to which the mass refers
}
nDisk = AddElement(disk)
```

3.2.7 Rigid3D

Short description

A rigid body in 3D.

Degrees of freedom

The first 3 degrees of freedom are those describing the position. The rotation is parameterized with 4 degrees of freedom and one additional algebraic equation.

Geometry

The center of gravity, S, is defined by the vector initial_position, which is in global coordinates, see figure 3.7. The rotation of the body-fixed local coordinate system w.r.t. the global coordiante system is defined by the Matrix initial_rotation.
In order to define the position of a point P of the element, e.g. for connectors or sensors, the local coordinate system is used. The reference point is the center of mass, S, so the values of the local coordinates can be positive or negative.

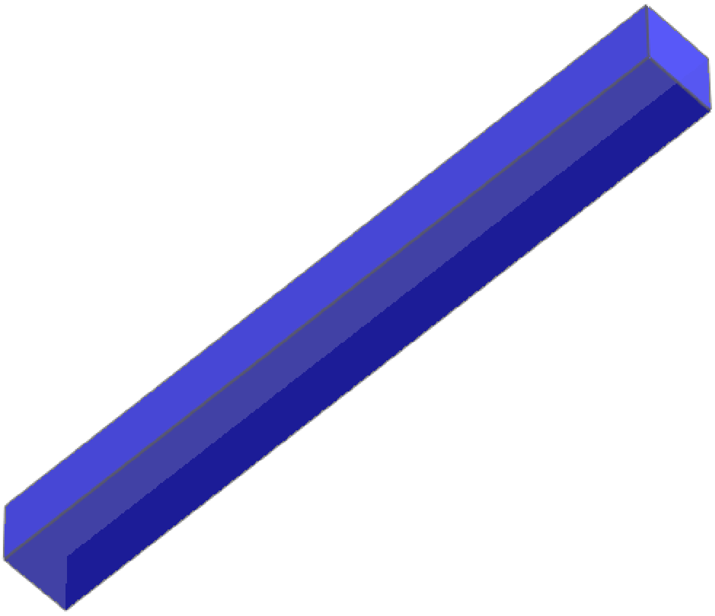


Figure 3.6: Rigid3D

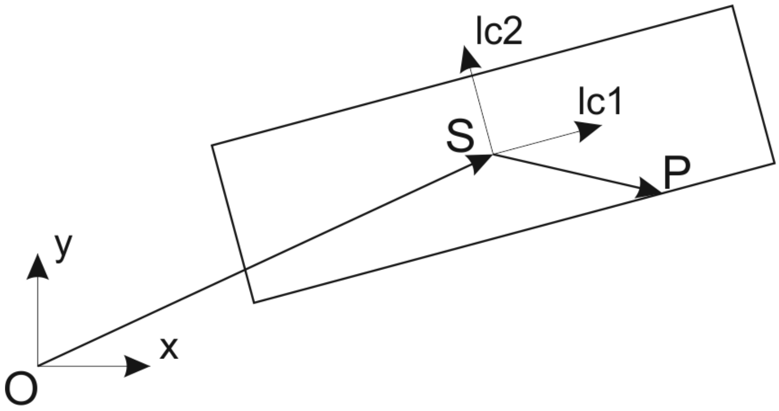


Figure 3.7: local and global coordinate system for a Rigid3D

Data objects of Rigid3D:

Data name	type	R	default	description
element_type	string		"Rigid3D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rigid3D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty

Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics. body_dimensions	vector		[1, 1, 1]	Dimensions of a regular cube [L_x, L_y, L_z] in m
Physics				
Physics. moment_of_inertia	matrix		[0.167, 0, 0; 0, 0.167, 0; 0, 0, 0.167]	[I_XX,I_XY,I_XZ; ...]
Physics.volume	double		1	volume of the body in m*m*m
Physics.mass	double		1	mass of the body in kg
Initialization				
Initialization. initial_position	vector		[0, 0, 0]	[X, Y, Z]
Initialization. initial_velocity	vector		[0, 0, 0]	[X, Y, Z]
Initialization. initial_rotation	vector		[0, 0, 0]	3 consecutive rotations (global rotation axes): [rot3_X, rot2_Y, rot1_Z] in rad
Initialization. initial_angular_velocity	vector		[0, 0, 0]	Angular velocity vector in global coordinates: [ang_X, ang_Y, ang_Z] in rad/s

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
angular_velocity_local_basis	x, y, z, magnitude
acceleration	x, y, z, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-15
Internal.second_order_variable	second order variables of the element. range: 1-7
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-7
Internal.algebraic_variable	algebraic variables of the element. range: 1-1

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file Rigid3D.txt

```
dimension = [1, 0.1, 0.1] %Dimensions of a regular cube [L_x, L_y, L_z] in m

my_data % compute inertia values
{
  density = 7850
  Cube.body_dimensions = dimension
}
inertia_values = ComputeInertia(my_data)

myRigid % add rigid body
{
  element_type= "Rigid3D"  %specification of element type.
  name= "my first rigid"  %name of the element
  Graphics.body_dimensions= dimension
  Physics
  {
    moment_of_inertia= inertia_values.moment_of_inertia
    volume= inertia_values.volume
    mass= inertia_values.mass
  }
  Initialization
  {
    initial_position= [0, 0, 0]  %[X, Y, Z]
  initial_rotation= [0, pi/2, 0]  %[rot3_X, rot2_Y, rot1_Z] in rad
  }
}
nElement = AddElement(myRigid)
```

3.2.8 Rigid3DKardan

Short description

A rigid body in 3D, implemented with bryant angles (also called Tait Bryan or Cardan angles).

Degrees of freedom

The first 3 degrees of freedom are those describing the position. The rotation is parameterized with 3 bryant angles with the sequence x-y-z. If you use this element for dynamic simulation of a fast rotating rigid body, it is adviced to use the global x-axis as rotation axis.

Geometry

The center of gravity, S, is defined by the vector initial_position, which is in global coordinates, see figure 3.7. The rotation of the body-fixed local coordinate system w.r.t. the global coordiante system is defined by the Matrix initial_rotation.
In order to define the position of a point P of the element, e.g. for connectors or sensors, the local coordinate system is used. The reference point is the center of mass, S, so the values of the local coordinates can be positive or negative.

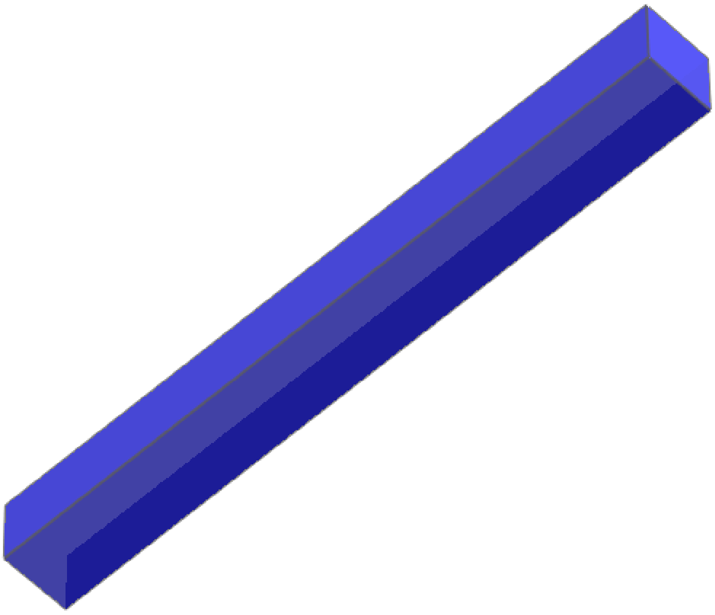


Figure 3.8: Rigid3DKardan

Data objects of Rigid3DKardan:

Data name	type	R	default	description
element_type	string		"Rigid3DKardan"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rigid3DKardan"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics. body_dimensions	vector		[1, 1, 1]	Dimensions of a regular cube [L_x, L_y, L_z] in m
Physics				
Physics. moment_of_inertia	matrix		[0.167, 0, 0; 0, 0.167, 0; 0, 0, 0.167]	[I_XX,I_XY,I_XZ; ...]

Physics.volume	double		1	volume of the body in m*m*m
Physics.mass	double		1	mass of the body in kg
Physics.rotations_sequence	string		"xyz"	rotations sequence, can be xyz, zxy or zxz
Initialization				
Initialization.initial_position	vector		[0, 0, 0]	[X, Y, Z]
Initialization.initial_velocity	vector		[0, 0, 0]	[X, Y, Z]
Initialization.initial_rotation	vector		[0, 0, 0]	3 consecutive rotations (global rotation axes): [rot3_X, rot2_Y, rot1_Z] in rad
Initialization.initial_angular_velocity	vector		[0, 0, 0]	Angular velocity vector in global coordinates: [ang_X, ang_Y, ang_Z] in rad/s

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
angular_velocity_local_basis	x, y, z, magnitude
acceleration	x, y, z, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-12
Internal.second_order_variable	second order variables of the element. range: 1-6
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-6

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, Pris-

maticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file Rigid3DKardan.txt

```
dimension = [1, 0.1, 0.1] %Dimensions of a regular cube [L_x, L_y, L_z] in m

my_data % compute inertia values
{
  density = 7850
  Cube.body_dimensions = dimension
}
inertia_values = ComputeInertia(my_data)

myRigid % add rigid body
{
  element_type= "Rigid3DKardan"  %specification of element type.
  name= "my first rigid with kardan angles"  %name of the element
  Graphics.body_dimensions= dimension
  Physics
  {
    moment_of_inertia= inertia_values.moment_of_inertia
    volume= inertia_values.volume
    mass= inertia_values.mass
  }
  Initialization
  {
    initial_position= [0, 0, 0]  %[X, Y, Z]
  initial_rotation= [0, pi/2, 0]  %[rot3_X, rot2_Y, rot1_Z] in rad
  }
}
nElement = AddElement(myRigid)
```

3.2.9 Rigid3DMinCoord

Short description

A rigid body with just one degree of freedom. Efficient formulation for robotic applications are possible with this body.

Degrees of freedom

The body just has 1 (own) degree of freedom (d.o.f.). Depending on the type of joint it is a translational or rotational one, see figure 3.9 and figure 3.10, which rotates or translates with respect to the i-th coordinate system around or along the Z-axis (additionally to the initial parameters θ and respectively d in figure 3.11). If you look at the i-th body in a chain of such bodies, then the i-th body seems to have i d.o.f.s. In fact it also just adds 1 d.o.f. to the system. If you are using connectors or loads which use a d.o.f. directly (e.g. GCLoad or CoordinateConstraint) you have to be carefull with the settings. In these cases the i-th d.o.f. of the i-th body is the correct one.

Geometry

The reference frame of the body is defined with Denavit-Hartenberg parameters, see figure 3.11, and an (optional) additional rotation matrix. The local reference frame is shown with the following colors: x in green, y in blue and z in red. See figure 3.9 and figure 3.10.

Equations

The implementation is based on the so-called 'Projection Equation' by Bremer. For details see [13].

Limitations

The first body of a chain of such Rigid3DMinCoord bodies has to be fixed to ground. It is not possible yet to connect a robot built up with these elements to e.g. a Rigid3D.
The implementation of the translational degree of freedom is up to now just tested for the case, that there is just one transl. d.o.f. and that this joint is the first one in the kinematic chain (=fixed to ground).

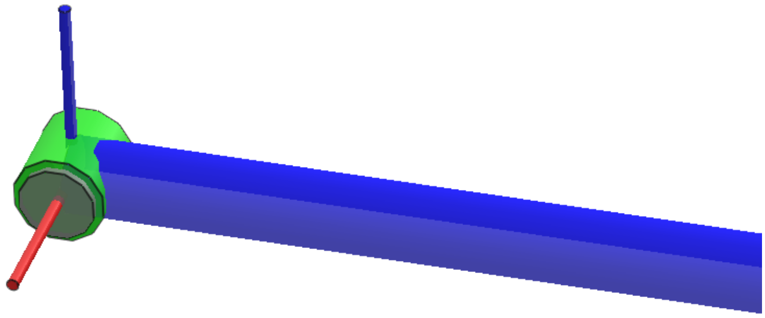


Figure 3.9: Rigid3DMinCoord with rotational degree of freedom

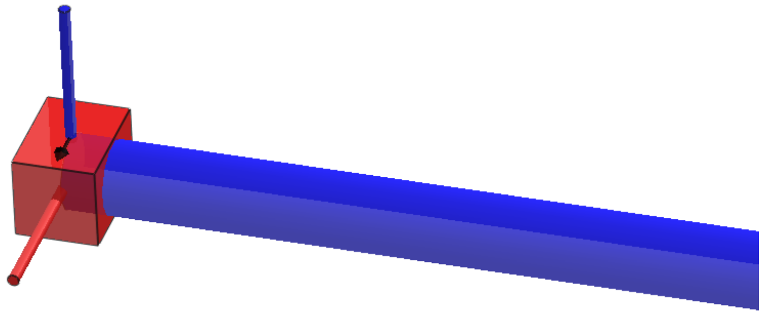


Figure 3.10: Rigid3DMinCoord with translational degree of freedom

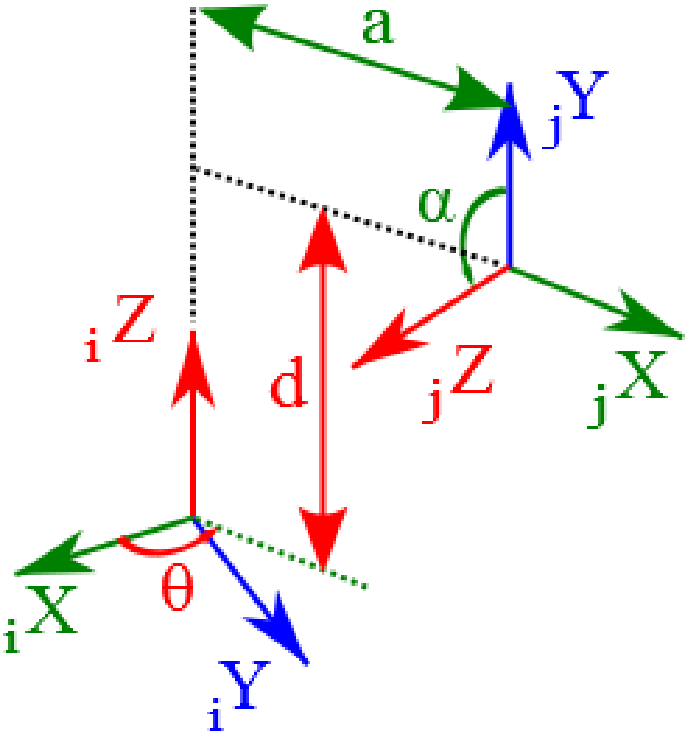


Figure 3.11: Definition of the geometry with DenavitŨHartenberg parameters [14]

Data objects of Rigid3DMinCoord:

Data name	**type**	**R**	**default**	**description**
element_type	string		"Rigid3DMinCoord"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rigid3DMinCoord"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Graphics.position_offset	vector		[0, 0, 0]	reference position, global vector to reference frame of first body. Only different from zero for first body!
Geometry				
Geometry.prev_body	integer		0	element number of previous body in chain
Geometry.link_type	integer		1	1...rotation of body i around origin of local body-fixed frame (joint), 2...sliding joint

Geometry. next_link_position	vector	**R**	[1, 0, 0]	1r12 vector from origin of local frame (= joint) to end of body. [X Y Z] in first body fixed coordinate system.
Geometry. joint_local_frame	vector		[0, 0, 0]	Euler angles between global coordinate system to first body or between next_link_rotation and local coordinate system: 3 consecutive rotations (local rotation axes): [rot3_X, rot2_Y, rot1_Z] in rad
Geometry. DH_parameters	vector		[0, 0, 1, 0]	Denavit-Hartenberg Parameters: [theta (rad), d (m), a (m), alpha (rad)]
Physics				
Physics.mass	double		1	mass of the body in kg
Physics. center_of_gravity	vector		[0.5, 0, 0]	vector from link to center of gravity in local frame. Measured in first body fixed coordinate system.
Physics. moment_of_inertia	matrix		[0, 0, 0; 0, 0.0833, 0; 0, 0, 0.0833]	[I XX,I XY,I XZ; ...] w.r.t. center of gravity, defined in body fixed coordinate system.
Physics. moment_of_inertia_add	double		0	additional relative inertia moment (e.g. inertia of drive at link side)
Initialization				
Initialization. initial_position	vector		[0]	in m or rad, depending on link_type
Initialization. initial_velocity	vector		[0]	in m/s or rad/s, depending on link_type

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
velocity	x, y, z, magnitude

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-2
Internal.second_order_variable	second order variables of the element. range: 1-1
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-1

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17,

Example

see file Rigid3DMinCoordDoublePendulum.txt

```
grav.load_type= "Gravity"
grav.direction= 2  %global direction of the gravity
grav.gravity_constant= -9.81  %use negative sign if necessary
nLoad = AddLoad(grav)

pendulum
{
  element_type= "Rigid3DMinCoord"
  loads= [nLoad]
  Geometry
  {
    prev_body= 0  % 0 is constraint to ground = first body in chain
    link_type= 1  % 1...rotational degree of freedom
    DH_parameters= [0, 0, 1, 0]  % Denavit-Hartenberg Parameters: [theta (rad), d (m), a (m), a
  }
  Physics
  {
    mass= 0.1  % mass of the body in kg
    center_of_gravity= [0.5, 0, 0]  %vector to center of gravity
    moment_of_inertia= [0, 0, 0
        0, 0.008354166666666666, 0
        0, 0, 0.008354166666666666]  %[I XX,I XY,I XZ; ...]
  }
}
AddElement(pendulum)

% add second pendulum with same geometry and orientation to end of first pendulum
pendulum.Geometry.prev_body= 1  % element number of previous body in chain
AddElement(pendulum)
```

3.2.10 LinearBeam3D

Short description

The Beam3D element is a three dimensional elastic beam element which is aligned along the local x axis. It provides a decoupled calculation of bending in y and z direction, axial deformation in x direction and torsion about the x axis. Shear deformation is not considered. The decoupled calculation is a simplification of the real, nonlinear problem, but for small deformations the results coincidence highly with the exact solution.

Degrees of freedom

Bending is described by 4 DOF, the number of DOF for axial deformation as well as torsion is 2. These 12 DOF are stored in two nodes i and j. The DOF vector of the LinearBeam3D read as follows

$$\mathbf{q}^{(i)} = [\mathbf{q}^{(i)}, \mathbf{q}^{(j)}] = [x^{(i)}, y^{(i)}, z^{(i)}, \phi_x^{(i)}, \phi_y^{(i)}, \phi_z^{(i)}, x^{(j)}, y^{(j)}, z^{(j)}, \phi_x^{(j)}, \phi_y^{(j)}, \phi_z^{(j)}]^T. \tag{3.7}$$

Nodes

To create a new beam element the user has to define two 'Node3DRxyz' nodes i and j. Every node of this type has 6 DOF. The first 3 DOF describe the node displacement (x, y, z) w.r.t global coordinate system, the last 3 DOF are angles of rotation (ϕ_x, ϕ_y, ϕ_z) w.r.t global coordinate system. All angles are considered as small (linearized angles). The reference positions of the nodes define the beam ends at initial configuration and so the length of the beam. The beam orientation is defined due to reference rot angles of node i. The advantage of using nodes with global DOF is the possibility to discretize a beam element into small beams easily without needing complicated constraint conditions. The beam elements do not even have to be aligned along a straight line. If using the same node number for the boundary point of the adjoint beams, beam elements are constrained automatically, see figure 3.13.

Geometry

The beam geometry is fully defined by 2 'Node3DRxyz' nodes and a 'Beam3DProperties' material element. Beam length and orientation is specified due to node positions and the orientation of the first node. The beam cross section size is defined due to the material. See figure 3.12 for more details. In order to define the position of point P of the element, e.g. for connectors or sensors, the local coordinate system is used. The origin of the local coordinate system is the center of gravity of the beam, p_0 is the vector to the center of gravity.

Limitations

Shear deformation is not considered. The decoupled calculation is a simplification of the real, nonlinear problem, but for small deformations the results coincidence highly with the exact solution.

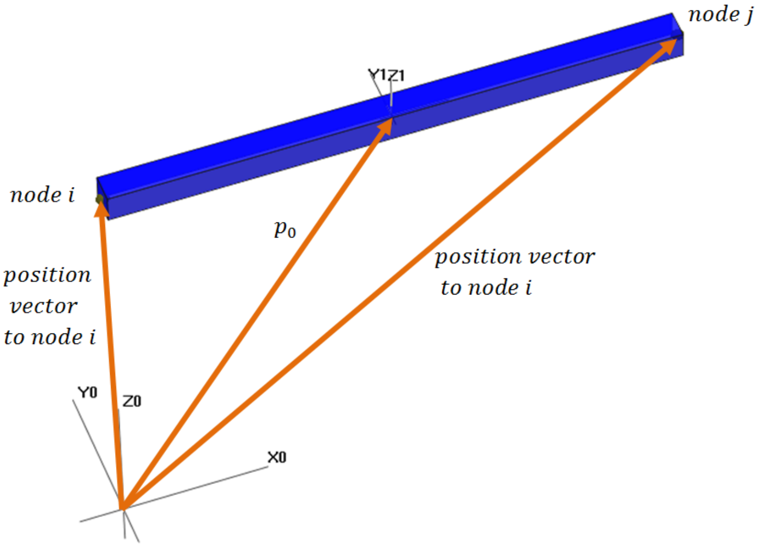


Figure 3.12: LinearBeam3D - Geometry

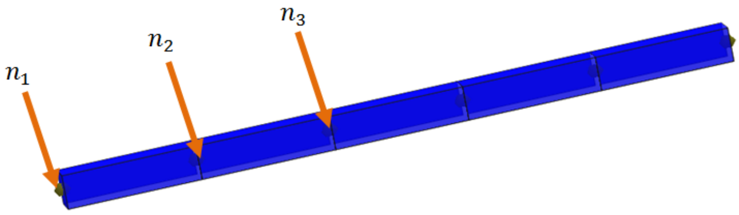


Figure 3.13: LinearBeam3D - Nodes

Data objects of LinearBeam3D:

Data name	type	R	default	description
element_type	string		"LinearBeam3D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"LinearBeam3D"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Geometry				
Geometry.node_1	integer		1	number of Node 1
Geometry.node_2	integer		2	number of Node 2
Physics				

Physics. axial_deformation	bool		1	include effect of axial deformation
Physics.material_number	integer		1	material number which contains the main material properties of the beam

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
beam_torsion	
beam_force_axial	
beam_force_transversal	y, z
beam_moment_torsional	
beam_moment_bending	y, z
acceleration	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
angular_velocity_local_basis	x, y, z, magnitude
stress	xx, xy, xz, yy, yz, zz, magnitude
stress_mises	

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-24
Internal.second_order_variable	second order variables of the element. range: 1-12
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-12
Internal.acceleration	accelerations of the element. range: 1-12

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, Spring-

DamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file LinearBeam3D.txt

```
%=====================================================================
% define a material
beam_material
{
  material_type = "Beam3DProperties"
  cross_section_type = 1 % rectangular cross section
  cross_section_size = [0.1,0.1]
  density = 1
  EA = 1
  EIy = 1
  EIz = 1
  GAky = 1
  GAkz = 1
  GJkx = 1
  RhoA = 1
  RhoIx = 1
  RhoIy = 1
  RhoIz = 1
}
nBeamMaterial = AddBeamProperties(beam_material)

%=====================================================================
% define two nodes

node1
{
  node_type = "Node3DRxyz"
  Geometry
  {
    reference_position = [0,0,0]
    reference_rot_angles = [0,0,0]
  }
}
n1 = AddNode(node1)

node2
{
  node_type = "Node3DRxyz"
  Geometry
  {
    reference_position = [1,0,0]
    reference_rot_angles = [0,0,0]
  }
}
n2 = AddNode(node2)
```

```
beam
{
  element_type= "LinearBeam3D"
  Physics
  {
  material_number = nBeamMaterial
  }
  Geometry.node_1 = n1
  Geometry.node_2 = n2
}
nBeam = AddElement(beam)
```

3.2.11 RotorBeamXAxis

Short description

The RotorBeamXAxis element is a three dimensional elastic rotor beam element. It has exact the same characteristics and properties as the LinearBeam3D element except two differences. The first difference is that for a rotor element it is necessary to enable big rotation about the rotor axis instead of the small rotation of the LinearBeam3D. The second difference is that all element DOF are stored w.r.t. local beam coordinate system.

Degrees of freedom

Bending is described by 4 DOF, the number of DOF for axial deformation as well as torsion is 2. These 12 DOF are stored in two nodes i and j. The DOF vector of the LinearBeam3D read as follows

$$\mathbf{q}^{(i)} = [\mathbf{q}^{(i)}, \mathbf{q}^{(j)}] = [x^{(i)}, y^{(i)}, z^{(i)}, \phi_x^{(i)}, \phi_y^{(i)}, \phi_z^{(i)}, x^{(j)}, y^{(j)}, z^{(j)}, \phi_x^{(j)}, \phi_y^{(j)}, \phi_z^{(j)}]^T. \tag{3.8}$$

Nodes

To create a new rotor beam element the user has to define two 'Node3DR123' nodes i and j. Every node of this type has 6 DOF. The first 3 DOF describe the node displacement (x, y, z) w.r.t local rotor element coordinate system, the last 3 DOF are angles of rotation (ϕ_x, ϕ_y, ϕ_z) w.r.t local rotor element coordinate system. The rotation about the local x-axis is considered as large, the rotations about the local y and z-axes are considered as small (linearized angles). The reference positions of the nodes define the beam ends at initial configuration and so the length of the beam. The beam orientation is defined due to reference rot angles of node i.

Geometry

The rotor beam geometry is fully defined by 2 'Node3DR123' nodes and a 'Beam3DProperties' material element. Beam length and orientation is specified due to node positions and the beam cross section size due to the material. The rotor beam has a circular cross section.

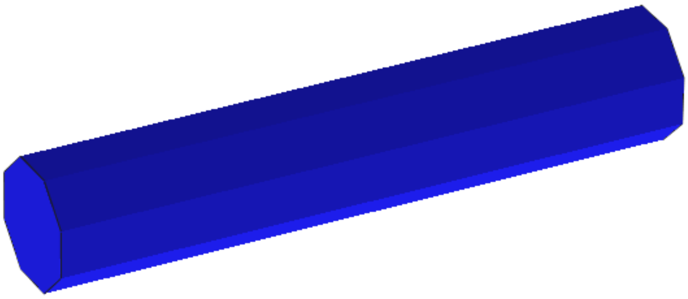


Figure 3.14: RotorBeamXAxis

Data objects of RotorBeamXAxis:

Data name	type	R	default	description
element_type	string		"RotorBeamXAxis"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"RotorBeamXAxis"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
Geometry				
Geometry.node_1	integer		1	number of Node 1
Geometry.node_2	integer		2	number of Node 2
Physics				
Physics. axial_deformation	bool		1	include effect of axial deformation
Physics.material_number	integer		1	material number which contains the main material properties of the beam

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	possible components

position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
beam_torsion	
beam_force_axial	
beam_force_transversal	y, z
beam_moment_torsional	
beam_moment_bending	y, z
acceleration	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
angular_velocity_local_basis	x, y, z, magnitude
stress	xx, xy, xz, yy, yz, zz, magnitude
stress_mises	

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-24
Internal.second_order_variable	second order variables of the element. range: 1-12
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-12
Internal.acceleration	accelerations of the element. range: 1-12

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file RotorBeamXAxis.txt

```
%===================================================================
% define a material
beam_material
{
  material_type = "Beam3DProperties"
  cross_section_type = 2 % circular cross section
  cross_section_size = [0.1]
  density = 1
  EA = 1
  EIy = 1
  EIz = 1
  GAky = 1
  GAkz = 1
  GJkx = 1
  RhoA = 1
  RhoIx = 1
  RhoIy = 1
  RhoIz = 1
}
nBeamMaterial = AddBeamProperties(beam_material)

%======================================================================
% define two nodes

node1
{
  node_type = "Node3DR123"
  Geometry
  {
    reference_position = [0,0,0]
    reference_rot_angles = [0,0,0]
  }
}
n1 = AddNode(node1)

node2
{
  node_type = "Node3DR123"
  Geometry
  {
    reference_position = [1,0,0]
    reference_rot_angles = [0,0,0]
  }
}
n2 = AddNode(node2)


rotor_beam
{
  element_type= "RotorBeamXAxis"
  Physics
  {
  material_number = nBeamMaterial
  }
  Geometry.node_1 = n1
  Geometry.node_2 = n2
}
nRotorBeam = AddElement(rotor_beam)
```

3.2.12 ANCFBeamShear3DLinear

Short description

ANCFBeamShear3DLinear is an ANCF beam finite element for multibody dynamics systems which is capable of large deformations and can be used for static as well as dynamic investigations. The beam finite element can reproduce axial, bending, shear and torsional deformation. A linear interpolation for the geometry and the displacement along the beam axis is chosen. The definition of the beam finite element is based on the absolute nodal coordinate formulation (ANCF), which uses slope vectors for the parameterization of the orientation of the cross section instead of rotational parameters. Two different formulations for the elastic forces of the beam elements are presented:
(1) A structural mechanics based formulation of the elastic forces based on Reissner's nonlinear rod theory including generalized strain measures. A term accounting for thickness and cross section deformation is included and shear locking is prevented.
(2) A continuum mechanics based formulation of the elastic forces for a St.Venant Kirchhoff material which avoids the Poisson and shear locking phenomenon.

Degrees of freedom

The degrees of freedom of the i-th node are the nodal displacements and change of slope vectors and read as follows

$$\mathbf{q}^{(i)} = [\mathbf{u}^{(i)\,T} \quad \mathbf{u}_{,\eta}^{(i)\,T} \quad \mathbf{u}_{,\zeta}^{(i)\,T}]^T. \tag{3.9}$$

Hence, nine degrees of freedom are specified in each node, therefore the two-noded linear beam element has 18 degrees of freedom.

Nodes

The element needs 2 nodes of type 'Node3DS2S3'. The element is described by two nodes at the end points of the beam (node 1 = left node, node 2 = right node). See Fig. 3.15 for a sketch of the two-noded linear beam element and the degrees of freedom per node.

Geometry

The deformed geometry of the ANCF beam finite elements is defined by position and two slope vectors in each node, see Fig. 3.15. The slope vectors $\mathbf{r}_{,\eta}^{(i)}$ and $\mathbf{r}_{,\zeta}^{(i)}$ are no unit vectors, therefor a cross section deformation is not prohibited. The displacement along the beam axis is interpolated with linear shape functions, while the orientation of the cross section is interpolated linearly. The slope vectors are the derivative vectors with respect to the coordinate system of the scaled straight reference element, see Fig. 3.16.

Description of the different modi

CMF	The definition of the elastic forces is based on a continuum mechanics based formulation for a St.Venant Kirchhoff material using the relation between the nonlinear Green-Lagrange strain tensor and the second Piola-Kirchhoff stress tensor. The beam is defined as any other solid finite element and the volume integration can be chosen via the variables order_axial and order_crosssectional in this modus. Using the parameter perform_reduced_integration, the standard integration order is reduced, in order to reduce stiffening effects or computation time.
SMF	The definition of the elastic forces is based on a structural mechanics based formulation based on Reissner's nonlinear rod theory including generalized strain measures, namely the axial strain, the shear strains, the torsional strain, and the bending strains. The integration along the beam axis is performed as follows: two Lobatto integration points are used for the integration of the elastic forces covering cross section deformation and one Gauss point is used for the integration of the terms accounting for axial deformation, bending, shear and torsion.

Additional notes

In general: For further details on the definition of the elastic forces, the strain measures or the cross section deformation see reference [15].
Cross section deformation: In order to penalize a possible cross section deformation of the beam, an additional term is added to the classical strain energy and can be varied by the penalization factors named penalty. See reference [15] for more details.Examples: Find static and linearized dynamic applications of the beam element as well as nonlinear dynamic examples and buckling tests in reference [16].

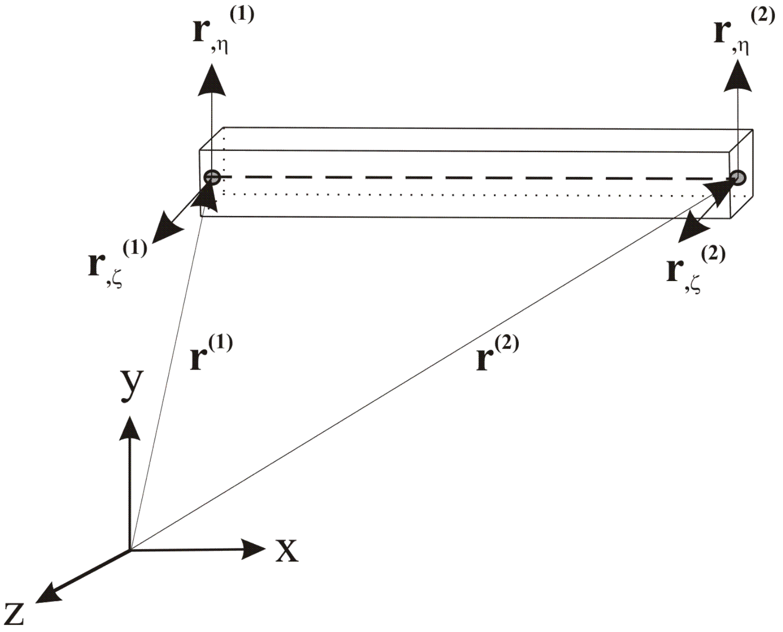
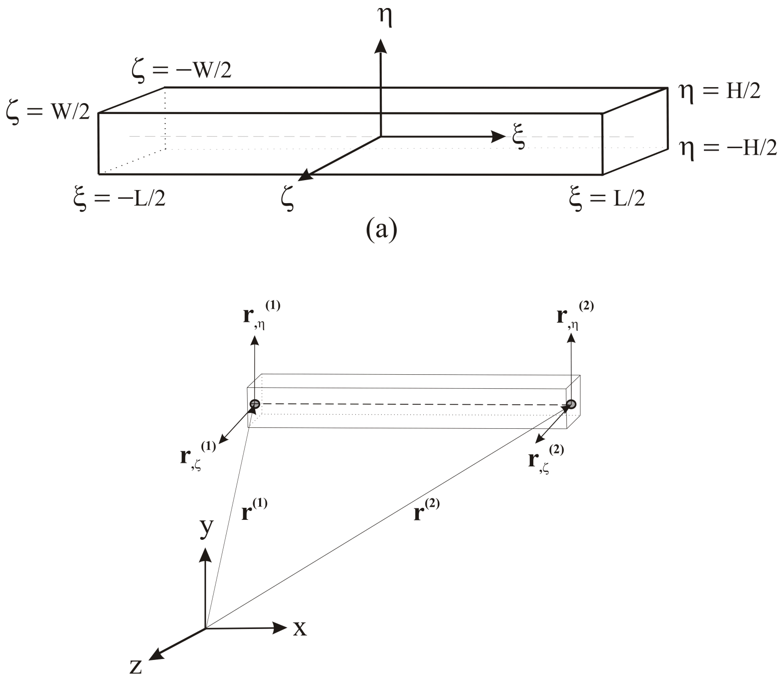


Figure 3.15: The geometric description of the elements is based on a position vector $\mathbf{r}^{(i)}$ and two slope vectors $\mathbf{r}_{,\eta}^{(i)}$ and $\mathbf{r}_{,\zeta}^{(i)}$ in the i-th node. These vectors are defined on a scaled and straight reference element, given in coordinates (ξ, η, ζ).



(a)

(b)

Figure 3.16: Different configurations of the finite beam element: (a) scaled straight reference element and (b) the reference element depicted in the global coordinate system.

Data objects of ANCFBeamShear3DLinear:

Data name	type	R	default	description

element_type	string		"ANCFBeamShear3DLinear"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Element"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty

Graphics

Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics.use_alternative_shape	bool		0	Graphical representation of element with geomobjects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element

ShearBeam

ShearBeam.straight_beam	bool		0	is straight beam in reference configuration
ShearBeam.reduced_integration	bool		0	reduced integration in cont. mech. formulation (CMF)
ShearBeam.beamformulation	integer		4	2 = CMF, 4 = SMF
ShearBeam.calc_linear	bool		0	linearized strain computation in cont. mech. formulation (CMF)
ShearBeam.nnodes	integer	**R**	2	number of nodes
ShearBeam.integration_order_axial	integer		4	axial integration order
ShearBeam.integration_order_cross_section	integer		2	cross section integration order, taks effect only in cont. mech. formulation (CMF)
ShearBeam.penalty_kappa	vector		[1, 1, 1]	penalty term for kappa [kappa1,kappa2,kappa3]
ShearBeam.penalty_gamma	vector		[1, 1, 1]	penalty term for gamma [gamma1,gamma2,gamma3]
ShearBeam.penalty_E	vector		[1, 1, 1]	penalty term for green lagrange strains (E) [Eyy,Ezz,Eyz]

Geometry

Geometry.body_dimensions	vector		[1, 0.1, 0.1]	dimensions of the beam. [L_x (length), L_y (width), L_z (height)]
Geometry.node_number1	integer		1	global number of node 1 (left), node must already exist
Geometry.node_number2	integer		2	global number of node 2 (right), node must already exist

Physics

Physics.material_number	integer		1	material number which contains the main material properties of the beam

Initialization

Initialization.node1_reference_position	vector		[0, 0, 0]	position of node 1 (left) in reference configuration.
Initialization.node1_reference_slope_2	vector		[0, 0, 0]	slope vector 2 of node 1 (left) in reference configuration.
Initialization.node1_reference_slope_3	vector		[0, 0, 0]	slope vector 3 of node 1 (left) in reference configuration.
Initialization.node2_reference_position	vector		[0, 0, 0]	position of node 2 (right) in reference configuration.
Initialization.node2_reference_slope_2	vector		[0, 0, 0]	slope vector 2 of node 2 (right) in reference configuration.

Initialization. node2_reference_slope_3	vector		[0, 0, 0]	slope vector 3 of node 2 (right) in reference configuration.

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
rotations_312	x, y, z, magnitude
velocity	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
beam_curvature	x, y, z, magnitude
beam_torsion	
beam_moment_bending	x, y, z, magnitude
beam_moment_torsional	
beam_shear	x, y, z, magnitude
beam_axial_extension	
beam_force_transversal	x, y, z, magnitude
beam_force_axial	

Observable special values:

For more information see section 3.1

value name	**description**
Internal.kinetic_energy	force in the rope
Internal.potential_energy	length of the rope

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file ANCFBeamShear3DLinear.txt

```
my_material
{
  material_type= "Beam3DProperties"
  cross_section_type= 1
  cross_section_size= [0.1, 0.1]
  EA= 20000
  EIy= 16.66666666666667
  EIz= 16.66666666666667
  GAky= 7692.307692307694
  GAkz= 7692.307692307694
  GJkx= 10.81538461538462
  RhoA= 72
  RhoIx= 0.12
  RhoIy= 0.06
  RhoIz= 0.06
  density= 7200
}
nMaterial = AddBeamProperties(my_material)

node
{
  node_type = "Node3DS2S3"
  Geometry.reference_position = [0,0,0]
  Geometry.reference_slope2 = [0,1,0]
  Geometry.reference_slope3 = [0,0,1]
}
nNode1 = AddNode(node)

node.Geometry.reference_position = [1,0,0]
nNode2 = AddNode(node)

my_beam
{
  element_type = "ANCFBeamShear3DLinear"
  Physics.material_number = nMaterial
  ShearBeam.beamformulation = 4
  Geometry.node_number1 = nNode1
  Geometry.node_number2 = nNode2
}
nElement = AddElement(my_beam)

ViewingOptions.FiniteElements.Nodes.show = 1
ViewingOptions.FiniteElements.Nodes.node_size = 0.05
```

3.2.13 ANCFBeamShear3DQuadratic

Short description

ANCFBeamShear3DQuadratic is an ANCF beam finite element for multibody dynamics systems which is capable of large deformations and can be used for static as well as dynamic investigations. The beam finite element can reproduce axial, bending, shear and torsional deformation. A quadratic interpolation for the geometry and the displacement along the beam axis is chosen.

The definition of the beam finite element is based on the absolute nodal coordinate formulation (ANCF), which uses slope vectors for the parameterization of the orientation of the cross section instead of rotational parameters. Two different formulations for the elastic forces of the beam elements are presented:

(1) A structural mechanics based formulation of the elastic forces based on Reissner's nonlinear rod theory including generalized strain measures. A term accounting for thickness and cross section deformation is included and shear locking is prevented.

(2) A continuum mechanics based formulation of the elastic forces for a St.Venant Kirchhoff material which avoids the Poisson and shear locking phenomenon.

Degrees of freedom

The degrees of freedom of the i-th node are the nodal displacements and change of slope vectors and read as follows

$$\mathbf{q}^{(i)} = [\mathbf{u}^{(i)T} \quad \mathbf{u}_{,\eta}^{(i)T} \quad \mathbf{u}_{,\zeta}^{(i)T}]^T. \tag{3.10}$$

Hence, nine degrees of freedom are specified in each node, therefore the three-noded quadratic beam element has 27 degrees of freedom.

Nodes

The element needs 3 nodes of type 'Node3DS2S3'. The element is described by three nodes: at the end points and the mid point of the beam (node 1 = left node, node 2 = right node, node 3 = mid point). See Fig. 3.15 for a sketch of the three-noded quadratic beam element and the degrees of freedom per node.

Geometry

The deformed geometry of the ANCF beam finite elements is defined by position and two slope vectors in each node, see Fig. 3.15. The slope vectors $\mathbf{r}_{,\eta}^{(i)}$ and $\mathbf{r}_{,\zeta}^{(i)}$ are no unit vectors, therefor a cross section deformation is not prohibited. The displacement along the beam axis is interpolated with quadratic shape functions, while the orientation of the cross section is interpolated linearly. The slope vectors are the derivative vectors with respect to the coordinate system of the scaled straight reference element, see Fig. 3.16.

Description of the different modi

CMF	The definition of the elastic forces is based on a continuum mechanics based formulation for a St.Venant Kirchhoff material using the relation between the nonlinear Green-Lagrange strain tensor and the second Piola-Kirchhoff stress tensor. The beam is defined as any other solid finite element and the volume integration can be chosen via the variables order_axial and order_crosssectional in this modus. Using the parameter perform_reduced_integration, the standard integration order is reduced, in order to reduce stiffening effects or computation time.

SMF	The definition of the elastic forces is based on a structural mechanics based formulation based on Reissner's nonlinear rod theory including generalized strain measures, namely the axial strain, the shear strains, the torsional strain, and the bending strains. The integration along the beam axis is performed as follows: two Lobatto integration points are used for the integration of the elastic forces covering cross section deformation and one Gauss point is used for the integration of the terms accounting for axial deformation, bending, shear and torsion.

Additional notes

In general: For further details on the definition of the elastic forces, the strain measures or the cross section deformation see reference [15].
Cross section deformation: In order to penalize a possible cross section deformation of the beam, an additional term is added to the classical strain energy and can be varied by the penalization factors named penalty. See reference [15] for more details.
Examples: Find static and linearized dynamic applications of the beam element as well as nonlinear dynamic examples and buckling tests in reference [16].

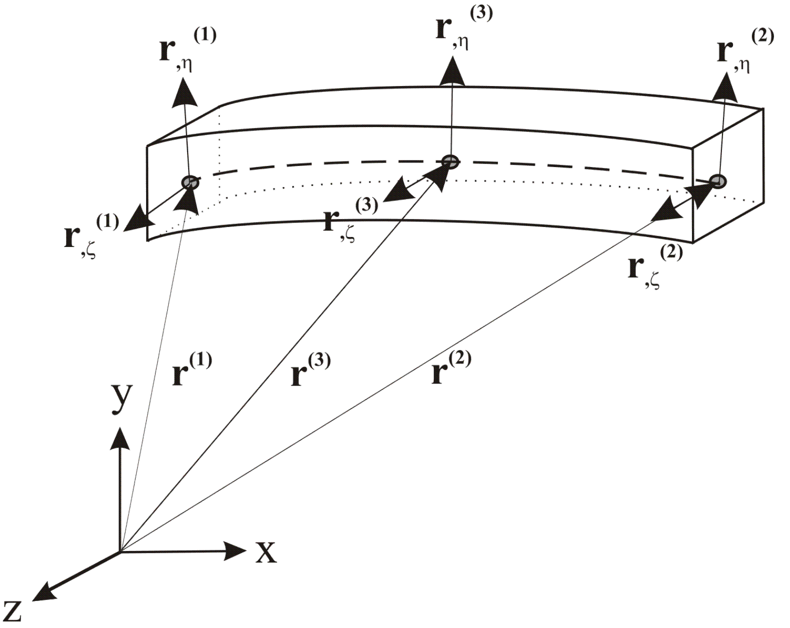

$\mathbf{r}_{,\eta}^{(1)}$
$\mathbf{r}_{,\eta}^{(3)}$
$\mathbf{r}_{,\eta}^{(2)}$
$\mathbf{r}_{,\zeta}^{(1)}$
$\mathbf{r}_{,\zeta}^{(3)}$
$\mathbf{r}_{,\zeta}^{(2)}$
$\mathbf{r}^{(1)}$
$\mathbf{r}^{(3)}$
$\mathbf{r}^{(2)}$
y
x
z

Figure 3.17: The geometric description of the elements is based on a position vector $\mathbf{r}^{(i)}$ and two slope vectors $\mathbf{r}_{,\eta}^{(i)}$ and $\mathbf{r}_{,\zeta}^{(i)}$ in the i-th node. These vectors are defined on a scaled and straight reference element, given in coordinates (ξ, η, ζ).

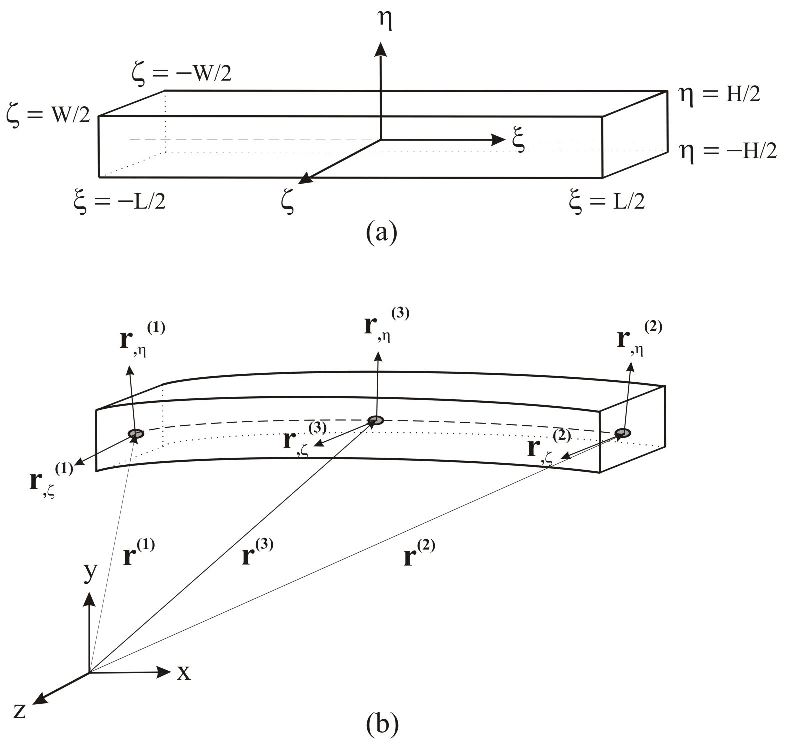

η
ζ = −W/2
ζ = W/2
η = H/2
ξ
η = −H/2
ξ = −L/2
ζ
ξ = L/2
(a)
$\mathbf{r}_{,\eta}^{(1)}$
$\mathbf{r}_{,\eta}^{(3)}$
$\mathbf{r}_{,\eta}^{(2)}$
$\mathbf{r}_{,\zeta}^{(3)}$
$\mathbf{r}_{,\zeta}^{(2)}$
$\mathbf{r}_{,\zeta}^{(1)}$
y
$\mathbf{r}^{(1)}$
$\mathbf{r}^{(3)}$
$\mathbf{r}^{(2)}$
x
z
(b)

Figure 3.18: Different configurations of the finite beam element: (a) scaled straight reference element and (b) the reference element depicted in the global coordinate system.

Data objects of ANCFBeamShear3DQuadratic:

Data name	**type**	**R**	**default**	**description**
element_type	string		"ANCFBeamShear3DQuadratic"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Element"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element
ShearBeam				
ShearBeam. straight_beam	bool		0	is straight beam in reference configuration
ShearBeam. reduced_integration	bool		0	reduced integration in cont. mech. formulation (CMF)
ShearBeam. beamformulation	integer		4	2 = CMF, 4 = SMF
ShearBeam.calc_linear	bool		0	linearized strain computation in cont. mech. formulation (CMF)
ShearBeam.nnodes	integer	**R**	3	number of nodes

ShearBeam. integration_order_axial	integer		4	axial integration order
ShearBeam.integration_order_cross_section	integer		2	cross section integration order, taks effect only in cont. mech. formulation (CMF)
ShearBeam. penalty_kappa	vector		[1, 1, 1]	penalty term for kappa [kappa1,kappa2,kappa3]
ShearBeam. penalty_gamma	vector		[1, 1, 1]	penalty term for gamma [gamma1,gamma2,gamma3]
ShearBeam.penalty_E	vector		[1, 1, 1]	penalty term for green lagrange strains (E) [Eyy,Ezz,Eyz]
Geometry				
Geometry. body_dimensions	vector		[1, 0.1, 0.1]	dimensions of the beam. [L_x (length), L_y (width), L_z (height)]
Geometry.node_number1	integer		1	global number of node 1 (left), node must already exist
Geometry.node_number2	integer		2	global number of node 2 (right), node must already exist
Geometry.node_number3	integer		3	global number of node 3 (middle), node must already exist
Physics				
Physics.material_number	integer		1	material number which contains the main material properties of the beam
Initialization				
Initialization. node1_reference_position	vector		[0, 0, 0]	position of node 1 (left) in reference configuration.
Initialization. node1_reference_slope_2	vector		[0, 0, 0]	slope vector 2 of node 1 (left) in reference configuration.
Initialization. node1_reference_slope_3	vector		[0, 0, 0]	slope vector 3 of node 1 (left) in reference configuration.
Initialization. node2_reference_position	vector		[0, 0, 0]	position of node 2 (right) in reference configuration.
Initialization. node2_reference_slope_2	vector		[0, 0, 0]	slope vector 2 of node 2 (right) in reference configuration.
Initialization. node2_reference_slope_3	vector		[0, 0, 0]	slope vector 3 of node 2 (right) in reference configuration.
Initialization. node3_reference_position	vector		[0, 0, 0]	position of node 3 (middle) in reference configuration.
Initialization. node3_reference_slope_2	vector		[0, 0, 0]	slope vector 2 of node 3 (middle) in reference configuration.
Initialization. node3_reference_slope_3	vector		[0, 0, 0]	slope vector 3 of node 3 (middle) in reference configuration.

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
bryant_angle	x, y, z, magnitude
rotations_312	x, y, z, magnitude

velocity	x, y, z, magnitude
angular_velocity	x, y, z, magnitude
beam_curvature	x, y, z, magnitude
beam_torsion	
beam_moment_bending	x, y, z, magnitude
beam_moment_torsional	
beam_shear	x, y, z, magnitude
beam_axial_extension	
beam_force_transversal	x, y, z, magnitude
beam_force_axial	

Observable special values:

For more information see section 3.1

value name	**description**
Internal.kinetic_energy	force in the rope
Internal.potential_energy	length of the rope

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, SlidingPointJoint, 3.3.5, SlidingPrismaticJoint, 3.3.6, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17, RotatorySpringDamperActuator, 3.3.18,

Example

see file ANCFBeamShear3DQuadratic.txt

```
my_material
{
  material_type= "Beam3DProperties"
  cross_section_type= 1
  cross_section_size= [0.1, 0.1]
  EA= 20000
  EIy= 16.66666666666667
  EIz= 16.66666666666667
  GAky= 7692.307692307694
  GAkz= 7692.307692307694
  GJkx= 10.81538461538462
  RhoA= 72
  RhoIx= 0.12
  RhoIy= 0.06
  RhoIz= 0.06
  density= 7200
}
```

```
nMaterial = AddBeamProperties(my_material)

node
{
  node_type = "Node3DS2S3"
  Geometry.reference_position = [0,0,0]
  Geometry.reference_slope2 = [0,1,0]
  Geometry.reference_slope3 = [0,0,1]
}
nNode1 = AddNode(node)

node.Geometry.reference_position = [1,0,0]
nNode2 = AddNode(node)

node.Geometry.reference_position = [0.5,0,0]
nNode3 = AddNode(node)

my_beam
{
  element_type = "ANCFBeamShear3DQuadratic"
  Physics.material_number = nMaterial
  ShearBeam.beamformulation = 4
  Geometry.node_number1 = nNode1
  Geometry.node_number2 = nNode2
  Geometry.node_number3 = nNode3
}
nElement = AddElement(my_beam)

ViewingOptions.FiniteElements.Nodes.show = 1
ViewingOptions.FiniteElements.Nodes.node_size = 0.05
```

3.2.14 ANCFBeam3DTorsion

Short description

ANCFBeam3DTorsion is a Bernoulli-Euler beam finite element in ANCF (Absolute Nodal Coordinate Formulation) capable of large axial, bending, and torsional deformations.

Degrees of freedom

The element affects 14 degrees of freedom (generalized coordinates) in total, which are 7 degrees of freedom per node, i.e., at each node we have: the axial displacement $\mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the change of the axial slope $\mathbf{u}' = \mathbf{r}' - \mathbf{r}_0'$, and the change of the torsional angle $\theta - \theta_0$. Each quantity with index 0 here confers to the reference configuration. The element wise ordering of the degrees of freedom is displayed in Fig. 3.20.

Nodes

The element operates with two Nodes of type `Node3DS1rot1`, each of which are located at either tip of the beam element. The integer values `Geometry.node_number1` and `Geometry.node_number2` refer to the index of the nodes in the multibody system. Each of these Nodes is instantiated by the user with a position and a rotation (kardan angles), and provides a frame $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ (which is measured in the global frame of the multibody system) for the instantiation of the beam elemtent: At each node, the slope of the beam axis $\mathbf{r}'$ is identical with $\mathbf{e}_1$, and the director is defined as $\mathbf{e}_3$.

Geometry

The geometry of the element is defined by the nodal values for axial position $\mathbf{r}$, the axial slope vector $\mathbf{r}'$, and the torsional angle θ of the cross section around the beam axis, see Fig. 3.20. This angle is measured with respect to a reference direction in the global frame (director). Between the nodal values, the axial position is interpolated cubically, the axial slope is interpolated quadratically, and the torsional angle of the cross section (around the beam axis) as well as the director are interpolated linearly.

Equations

Variation of the strain energy:

$$\delta\Pi = \int_{-L/2}^{L/2} (EA\varepsilon\,\delta\varepsilon + GJ\kappa_1\,\delta\kappa_1 + EI_y\kappa_2\,\delta\kappa_2 + EI_z\kappa_3\,\delta\kappa_3)\,d\xi. \tag{3.11}$$

Considering viscous material damping, we replace $\varepsilon \to \varepsilon^E$ and $\kappa_i \to \kappa_i^E$ for $i \in \{1,2,3\}$ with

$$\varepsilon^E = \varepsilon + \varepsilon^D, \qquad \varepsilon^D = c_K\,\dot{\varepsilon} = c_K\left(\frac{\partial\varepsilon}{\partial\mathbf{q}}\right)^T\dot{\mathbf{q}}, \tag{3.12}$$

$$\kappa_i^E = \kappa_i + \kappa_i^D, \qquad \kappa_i^D = c_K\,\dot{\kappa}_i = c_K\left(\frac{\partial\kappa_i}{\partial\mathbf{q}}\right)^T\dot{\mathbf{q}}, \tag{3.13}$$

resulting in

$$\begin{aligned}\delta\Pi = \int_{-L/2}^{L/2} &(EA\,(\varepsilon + c_K\,\dot{\varepsilon})\,\delta\varepsilon + GJ\,(\kappa_1 + c_K\,\dot{\kappa}_1)\,\delta\kappa_1 + \\ &+EI_y\,(\kappa_2 + c_K\,\dot{\kappa}_2)\,\delta\kappa_2 + EI_z\,(\kappa_3 + c_K\,\dot{\kappa}_3)\,\delta\kappa_3)\,d\xi.\end{aligned} \tag{3.14}$$

Additional notes

For details on the element, such as the definition of the elastic forces and the kinetic terms, see [17, 18].

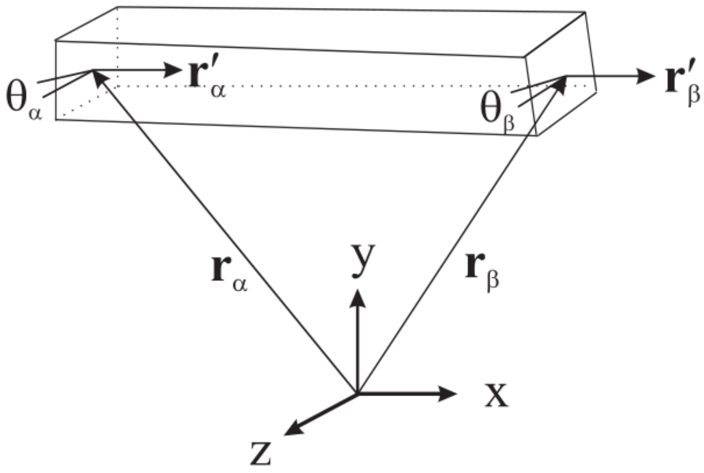


Figure 3.19: The geometry of the element is defined by nodal values for (a) the axial position, (b) the axial slope vector, and (c) the torsional angle of the cross section around the beam axis. This angle is measured with respect to a reference direction in the global frame (director). Between the nodal values, the axial position is interpolated cubically, the axial slope is interpolated quadratically, and the torsional angle of the cross section around the beam axis) as well as the director are interpolated linearly.

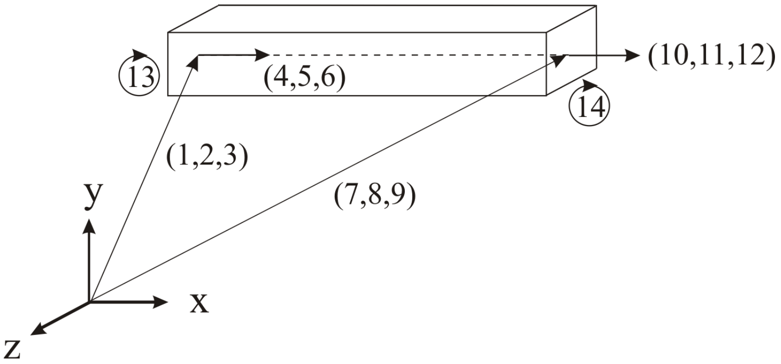


Figure 3.20: Ordering of the generalized coordinates.

Data objects of ANCFBeam3DTorsion:

Data name	type	R	default	description
element_type	string		"ANCFBeam3DTorsion"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"ANCFBeam3DTorsion"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
loads	vector		[]	Set loads attached to this element: 'nr_load1, nr_load2, ...' or empty
Graphics				
Graphics.RGB_color	vector		[0.1, 0.1, 0.8]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]

Graphics.geom_elements	vector		[]	Set Geometric elements to represent body 'geomelem1, geomelem2, ...' or empty
Graphics. use_alternative_shape	bool		0	Graphical representation of element with geom-objects that are attached to the element
Graphics.show_element	bool		1	Flag to draw element

Geometry

Geometry.node_number1	integer		1	global number of node 1 (left), node must already exist
Geometry.node_number2	integer		2	global number of node 2 (right), node must already exist
Geometry. update_directors	bool		0	update directors during calculation

Computation

Computation.kinematic_computation_mode	integer		0	0 .. exact kinematic terms + 5th order gaussian integration (slow), 1 .. exact terms + 1st order lobatto integration (fast), 2 .. constant mass matrix approximation (fastest)

Computation.IntegrationOrder

Computation. IntegrationOrder.mass	integer		4	integration order for mass terms
Computation. IntegrationOrder. axial_strain	integer		9	integration order for work of axial strain
Computation. IntegrationOrder. curvature	integer		5	integration order for work of curvature

Physics

Physics.material_number	integer		1	material number which contains the main material properties of the beam

Observable FieldVariables:

The following values can be measured with a FieldVariableElementSensor, 3.9.1. The sensor needs 2 informations: the field_variable itself and the component. For more information see section 3.1

field_variable	**possible components**
position	x, y, z, magnitude
displacement	x, y, z, magnitude
velocity	x, y, z, magnitude
beam_axial_extension	
beam_force_axial	
beam_curvature	x, y, z, magnitude
beam_moment	x, y, z, magnitude
beam_torsion	
beam_moment_torsional	

Observable special values:

For more information see section 3.1

value name	description
Internal.kinetic_energy	force in the rope
Internal.potential_energy	length of the rope

Suitable Connectors:

The following connectors can be used to constrain the element:
PointJoint, 3.3.1, CoordinateConstraint, 3.3.2, VelocityCoordinateConstraint, 3.3.3, MultiCoordConstraint, 3.3.4, Rope3D, 3.3.7, FrictionConstraint, 3.3.8, Contact1D, 3.3.9, GenericBodyJoint, 3.3.10, RevoluteJoint, 3.3.11, PrismaticJoint, 3.3.12, UniversalJoint, 3.3.13, RigidJoint, 3.3.14, CylindricalJoint, 3.3.15, SpringDamperActuator, 3.3.16, RigidLink, 3.3.17,

Example

see file ANCFBeam3DTorsion.txt

```
node
{
  node_type= "Node3DS1rot1"
  Geometry
  {
    reference_position= [0, 0, 0]
    reference_rot_angles= [0, 0, 0]
  }
}
nNode1 = AddNode(node)

node.Geometry.reference_position = [1,0,0]
nNode2 = AddNode(node)

beamproperties
{
  material_type= "Beam3DProperties"
  cross_section_type= 1
  cross_section_size= [0.1, 0.1]
  EA= 2100000000
  EIy= 1750000
  EIz= 1750000
  GJkx= 2692307.692307693
}
nBeamProperties = AddBeamProperties(beamproperties)

element
{
  element_type= "ANCFBeam3DTorsion"
  loads= [1]
  Physics
  {
    material_number= nBeamProperties
  }
  Geometry
  {
    node_number1= nNode1
    node_number2= nNode2
  }
}
AddElement(element)
```

3.3 Connector

These connectors are available:

- PointJoint, 3.3.1
- CoordinateConstraint, 3.3.2
- VelocityCoordinateConstraint, 3.3.3
- MultiCoordConstraint, 3.3.4
- SlidingPointJoint, 3.3.5
- SlidingPrismaticJoint, 3.3.6
- Rope3D, 3.3.7
- FrictionConstraint, 3.3.8
- Contact1D, 3.3.9
- GenericBodyJoint, 3.3.10
- RevoluteJoint, 3.3.11
- PrismaticJoint, 3.3.12
- UniversalJoint, 3.3.13
- RigidJoint, 3.3.14
- CylindricalJoint, 3.3.15
- SpringDamperActuator, 3.3.16
- RigidLink, 3.3.17
- RotatorySpringDamperActuator, 3.3.18
- SpringDamperActuator2D, 3.3.19
- PointJoint2D, 3.3.20

Note:
In HOTINT several classes are treated as 'elements'. Connectors and control elements are also 'elements', and can therefore be edited and deleted in the GUI with the menu items of the elements.
In the script language the command `AddConnector` has to be used for the connectors in the list above and also for control elements.

3.3.1 PointJoint

Short description

The PointJoint constrains two elements at a local position or node each. If only one element is specified (second element 0), a ground PointJoint is realized. It is possible to constrain just some of the directions. If the first body is a rigid body then the constraint forces are applied as follows:

Connecting element to element:
The constraint forces are applied on both bodies at the position of the connection point of the second body.

Connecting element to ground:
The constraint forces are applied on the body

- at the position of the connection point on ground if the formulation is penalty and use_local_coordinate_ and
- at the position of the connection point of the element otherwise.

If the first element is a flexible body or a point mass the forces are applied differently. See *Limitations.*

Equations

Lagrange formulation:
The constraint equations are

$$\mathbf{C} = \mathbf{A}^T (\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{0}$$

or on velocity level

$$\mathbf{C} = \dot{\mathbf{A}}^T (\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{A}^T (\mathbf{v}_1 - \mathbf{v}_2) = \mathbf{0}$$

where each equation corresponds to a constrained direction.
The meaning of the variables is as follows:

$\mathbf{x}_1$	position of connection point on body 1 in global coordinates
$\mathbf{x}_2$	position of connection point on body 2 in global coordinates, or if constraint connects element to ground then connection point of ground in global coordinates
$\mathbf{v}_1$	time derivative of $\mathbf{x}_1$
$\mathbf{v}_2$	time derivative of $\mathbf{x}_2$
$\mathbf{A}$	rotation matrix from local joint coordinates to global coordinates. If Geometry.use_local_coordinate_system = 1 and Geometry.use_joint_local_frame = 1, then $\mathbf{A} = \mathbf{QJ}$. If Geometry.use_local_coordinate_system = 1 and Geometry.use_joint_local_frame = 0, then $\mathbf{A} = \mathbf{Q}$. If Geometry.use_local_coordinate_system = 0 and Geometry.use_joint_local_frame = 1, then $\mathbf{A} = \mathbf{J}$. If Geometry.use_local_coordinate_system = 0 and Geometry.use_joint_local_frame = 0, then $\mathbf{A} = \mathbf{I}$.
$\mathbf{Q}$	rotation matrix from local coordinate system of body 1 to global coordinates
$\mathbf{J}$	joint local frame

Penalty formulation:
The spring force is given by

$$\mathbf{f}_s = \mathbf{A} \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{pmatrix} \mathbf{A}^T \cdot (\mathbf{x}_2 - \mathbf{x}_1)$$

and the damper force by

$$\mathbf{f}_d = d \cdot (\mathbf{v}_2 - \mathbf{v}_1)\,.$$

The resulting constraint force is then given by

$$\mathbf{f} = \mathbf{f}_s + \mathbf{f}_d.$$

Where

$\mathbf{f}_s$	constraint force due to stiffness
$\mathbf{f}_d$	constraint force due to damping,
$\mathbf{f}$	constraint force
k_x	stiffness in (local or global) x-direction
k_y	stiffness in (local or global) y-direction
k_z	stiffness in (local or global) z-direction
d	damping coefficient

Limitations

In general the constraint forces act on the first body not at the position of the connection point of the first body.
If the first body is a rigid body, then the force acting on body 1 is shifted to the connection point of the first body. The moment induced by shifting the force is compensated by a moment in the opposite direction.
If the first body is a point mass, we cannot apply a force outside it's position.
If the first body is a flexible body, applying a force outside the connection point is at least very questionable.
Hence the PointJoint has the following limitations:

- It is not possible to use the PointJoint in Lagrangian formulation if not all directions are constrained and the first body is a point mass or a flexible body.

- When using the PointJoint in penalty formulation with body 1 being a point joint or a flexible body, the constraint force acting on body 1 is applied at the position of the connection point of body 1. If the constraint forces are not collinear to connection points, e.g. if the stiffness is unisotropic or if a damping is set, then the shifting the force induces a moment. This moment is not compensated and thus the law of angular momentum is broken. Therefore a PointJoint in penalty formulation should only be used with a point masses or a flexible bodies being the first element, if you expect small displacements or the constraint forces to be (nearly) collinear to the connection points.

Another limitations, which applies for all kinds of bodies, is that it is not possible in Lagrange formulation to constraint just some global directions.

Description of the different modi

element to ground	Position2.element_number AND Position2.node_number have to be equal to 0
element to element	Position2.element_number and/or Position2.node_number must not be equal to 0
Lagrange	If Physics.use_penalty_formulation = 0, then no stiffness and no damping parameters are used.

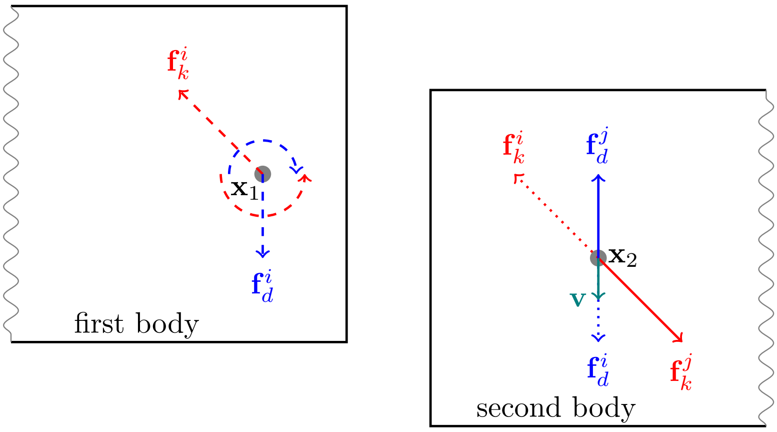


Figure 3.21: The penalty forces acting on the first body (dotted) act on the position of $\mathbf{x}_2$. The forces acting on the first body are shifted to $\mathbf{x}_1$ (dashed) and a moment (dashed) is applied to compensate the induced moment.

Data objects of PointJoint:

Data name	**type**	**R**	**default**	**description**
element_type	string		"PointJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"PointJoint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Graphics. draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Geometry				
Geometry. use_joint_local_frame	bool		0	Use a special joint local frame
Geometry. joint_local_frame	matrix		[0, 0, 0; 0, 0, 0; 0, 0, 0]	Rotates the local or global coordinate system. Just used if use_joint_local_frame == 1
Geometry. use_local_coordinate_system	bool		0	0=use global coordinates, 1=use local coordinate system of Body 1
Physics				

Physics. use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty. spring_stiffness	double		0	general or penalty stiffness parameter
Physics.Penalty. spring_stiffness_vector	vector		[0, 0, 0]	penalty stiffness parameter [kx,ky,kz]. Just used if scalar spring_stiffness == 0, otherwise kx=ky=kz=spring_stiffness
Physics.Penalty.damping	double		0	damping coefficient for viscous damping (F = d*v), applied in all constrained directions
Physics.Lagrange				
Physics.Lagrange. constrained_directions	vector		[1, 1, 1]	[x,y,z]...(1 = constrained, 0 = free), can be defined as local or global directions (see Geometry)
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position1.node_number	integer		0	local or global (if element_number == 0) node number.
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!
Position2.node_number	integer		0	local or global (if element_number == 0) node number.

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-3
Internal.algebraic_variable	algebraic variables of the element. range: 1-3
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction

Example

see file PointJointShort.txt

```
l = 1 % m
g = 9.81 % m/s^2

gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = g
}
nLoad = AddLoad(gravLoad)

rigidBody
{
  element_type= "Rigid3D"
  loads= [nLoad]
  Graphics.body_dimensions= [1, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

pointJoint
{
  element_type= "PointJoint"
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-1/2, 0, 0]  %local position
  }
  Position2.position= [-1/2, 0, 0]
}
AddConnector(pointJoint)
```

3.3.2 CoordinateConstraint

Short description

The CoordinateConstraint constrains two elements by constraining a single coordinate of each element, e.g. the x-displacement of two different elements. If the second element number is zero, a groundjoint can be realized. The CoordinateConstraint uses the lagrange multiplier formulation by default, which means that there is no constraint violation at all. For static problems, the lagrange multiplier constraint formulation is applied directly, by adding the kinematical conditions to the nonlinear system equations. In dynamic (time dependent) simulations, the constraint is solved on the position (displacement) levelwith index 3 solvers and on the velocity level with index 2 solvers. Alternatively, the penalty formulation can be used, which means that a certain (very high) spring stiffness is used instead of lagrange multipliers. Thus, no additional equation is added, however, the systemequations may become unsolvable stiff (ill conditioned) in case of static problems; for dynamical problems, the very high stiffness might lead to high-frequency oscillations, inaccurate solutions or no convergence.

Equations

Lagrange formulation:

position constraint (index 3 solver)
2 elements (coordinate to coordinate): $C = k\left(q_i^{el1} - q_{i,0}^{el1}\right) - \left(q_j^{el2} - q_{j,0}^{el2}\right) - d = 0$
1 element (coordinate to ground): $C = k\left(q_i^{el1} - q_{i,0}^{el1}\right) - d = 0$

velocity constraint - index reduction (index 2 solver)
2 elements (coordinate to coordinate): $C = k\,\dot{q}_i^{el1} - \dot{q}_j^{el2} = 0,$

1 element (coordinate to ground): $C = \dot{q}_i^{el1} = 0$

Langrange multiplier
$\frac{\partial C}{\partial \mathbf{q}^{el1}}^T = [0 \ldots\ldots\ldots 0\,, k\,, 0 \ldots 0]$ ... with k at index i
$\frac{\partial C}{\partial \mathbf{q}^{el2}}^T = [0 \ldots 0\,, -1\,, 0 \ldots\ldots\ldots 0]$ ... with -1 at index j

Penalty formulation:

2 elements (coordinate to coordinate): $f = S_P \left(k \left(q_i^{el1} - q_{i,0}^{el1} \right) - \left(q_j^{el2} - q_{j,0}^{el2} \right) - d \right) + D_P \left(k\, \dot{q}_i^{el1} - \dot{q}_j^{el2} \right)$
1 element (coordinate to ground): $f = S_P \left(k \left(q_i^{el1} - q_{i,0}^{el1} \right) \right) - d \right) + D_P\, k\, \dot{q}_i^{el1}$

Description:
k ... coordinate gain factor
d ... coordinate offset (for index 2 solvers not used)
q_i^{el1} ... i^{th} coordinate of element 1
$q_{i,0}^{el1} = q_i^{el1}\,(t=0)$ ... i^{th} coordinate of element 1 at initialization
q_j^{el2} ... j^{th} coordinate of element 2
$q_{j,0}^{el2} = q_j^{el2}\,(t=0)$ ... j^{th} coordinate of element 2 at initialization
S_P ... spring stiffness
D_P ... damping
C ... Lagrange equation
f ... force vector (penalty formulation)

Description of the different modi

coordinate to ground	Coordinate2.element_number AND Coordinate2.local_coordinate have to be equal to 0
coordinate to coordinate	Coordinate2.element_number and/or Coordinate2.local_coordinate must not be equal to 0
Lagrange	For Physics.use_penalty_formulation = 0 no stiffness parameter is used.
relative or absolute to initial values	Only important for max index 3 solvers. If relative_to_inital_values is set to 1: Equation above is used. If set to 0: Simplified equation is used ($q_{i,0}^{el1} = q_{j,0}^{el2} = 0$).

Data objects of CoordinateConstraint:

Data name	**type**	**R**	**default**	**description**
element_type	string		"CoordinateConstraint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"CoordinateConstraint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				

Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		0.1	General drawing size of constraint
Physics				
Physics. use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.damping	double		0	damping coefficient Dp for viscous damping
Physics.Penalty. spring_stiffness	double		0	general or penalty stiffness parameter Sp
coord_offset	double		0	coordinate offset d, see documentation section equation
coord_gain_factor	double		1	coordinate gain factor k, see documentation section equation
relative_to_inital_values	bool		1	flag == 1: full equation is used, see documentation; flag == 0: the init state values qi0 and qj0 are neglected.
Coordinate1				
Coordinate1. element_number	integer		0	element number for coordinate 1
Coordinate1. local_coordinate	integer		0	Local coordinate of element 1 to be constrained
Coordinate2				
Coordinate2. element_number	integer		0	element number for coordinate 2; for ground joint, set element number to zero
Coordinate2. local_coordinate	integer		0	Local coordinate of element 2 to be constrained

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-1
Connector.CoordinateConstraint.generalized_force	force acting on the generalized coordinates
Connector.CoordinateConstraint.coordinate_difference	difference between the coordinates
Connector.CoordinateConstraint.coordinate_offset	coordiante offset for CoordinateConstraint (w.r.t. ground or between two element coordinates); offset is ignored for Index 2 (setting of time integration) velocity level constraint
Connector.CoordinateConstraint.gain_factor	coordiante gain factor for CoordinateConstraint

Controllable special values:

For more information see section 3.1

value name	**description**

Connector.CoordinateConstraint.coordinate_offset	coordiante offset for CoordinateConstraint (w.r.t. ground or between two element coordinates); offset is ignored for Index 2 (setting of time integration) velocity level constraint
Connector.CoordinateConstraint.gain_factor	coordiante gain factor for CoordinateConstraint

Example

see file CoordinateConstraint.txt

```
l = 1 % m

rigidBody
{
  element_type= "Rigid3D"
  Graphics.body_dimensions= [l, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

coordinateConstraint
{
  element_type= "CoordinateConstraint"
  Coordinate1
  {
    element_number= nRigid  %element number for coordinate 1
    local_coordinate= 1  %local coordinate of element 1
  }
}
AddConnector(coordinateConstraint)
```

3.3.3 VelocityCoordinateConstraint

Short description

Similar to CoordinateConstraint. Lagrangian constraint implemented for index 3 and index 2 solvers. A penalty formulation is also implemented.

Equations

Lagrange formulation:

velocity constraint (index 2 and 3 solvers)
2 elements (coordinate to coordinate): $C = k\left(\dot{q}_i^{el1} - \dot{q}_{i,0}^{el1}\right) - \left(\dot{q}_j^{el2} - \dot{q}_{j,0}^{el2}\right) - d = 0$,
1 element (coordinate to ground): $C = k\left(\dot{q}_i^{el1} - \dot{q}_{i,0}^{el1}\right) - d = 0$

Langrange multiplier
$\frac{\partial C}{\partial \dot{\mathbf{q}}^{el1}}^T = [0 \ldots\ldots\ldots 0\,, k\,, 0 \ldots 0]$ ... with k at index i
$\frac{\partial C}{\partial \dot{\mathbf{q}}^{el2}}^T = [0 \ldots 0\,, -1\,, 0 \ldots\ldots\ldots 0]$ ... with -1 at index j

Penalty formulation:

2 elements (coordinate to coordinate): $f = S_P \left(k \left(\dot{q}_i^{el1} - \dot{q}_{i,0}^{el1}\right) - \left(\dot{q}_j^{el2} - \dot{q}_{j,0}^{el2}\right) - d\right)$
1 element (coordinate to ground): $f = S_P \big(k \big(\dot{q}_i^{el1} - \dot{q}_{i,0}^{el1}\big)\big) - d\big)$

Description:
k ... coordinate velocity gain factor
d ... coordinate velocity offset
$\dot{q}_i^{el1}$ ... i^{th} coordinate velocity of element 1
$\dot{q}_{i,0}^{el1} = \dot{q}_i^{el1}(t = 0)$ ... i^{th} coordinate velocity of element 1 at initialization
$\dot{q}_j^{el2}$ ... j^{th} coordinate velocity of element 2
$\dot{q}_{j,0}^{el2} = \dot{q}_j^{el2}(t = 0)$ ... j^{th} coordinate velocity of element 2 at initialization
S_P ... spring stiffness
C ... Lagrange equation
f ... force vector (penalty formulation)

Description of the different modi

coordinate to ground	Coordinate2.element_number AND Coordinate2.local_coordinate have to be equal to 0
coordinate to coordinate	Coordinate2.element_number and/or Coordinate2.local_coordinate must not be equal to 0
relative or absolute to initial values	If relative_to_inital_values is set to 1: Equation above is used. If set to 0: Simplified equation is used ($\dot{q}_{i,0}^{el1} = \dot{q}_{j,0}^{el2} = 0$).

Data objects of VelocityCoordinateConstraint:

Data name	type	R	default	description
element_type	string		"VelocityCoordinateConstraint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"VelocityCoordinateConstraint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		0.1	General drawing size of constraint
Physics				
Physics. use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty. spring_stiffness	double		0	general or penalty stiffness parameter Sp
coord_offset	double		0	coordinate offset d, see documentation section equation
coord_gain_factor	double		1	coordinate gain factor k, see documentation section equation

relative_to_inital_values	bool		1	flag == 1: full equation is used, see documentation; flag == 0: the init state derivatives d(qi0)/dt and d(qj0)/dt are neglected.
Coordinate1				
Coordinate1. element_number	integer		0	element number for coordinate 1
Coordinate1. local_coordinate	integer		0	Local coordinate of element 1 to be constrained
Coordinate2				
Coordinate2. element_number	integer		0	element number for coordinate 2; for ground joint, set element number to zero
Coordinate2. local_coordinate	integer		0	Local coordinate of element 2 to be constrained

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-1
Connector.CoordinateConstraint.generalized_force	force acting on the generalized coordinates
Connector.CoordinateConstraint.coordinate_difference	difference between the coordinates
Connector.CoordinateConstraint.coordinate_offset	coordiante offset for CoordinateConstraint (w.r.t. ground or between two element coordinates); offset is ignored for Index 2 (setting of time integration) velocity level constraint
Connector.CoordinateConstraint.gain_factor	coordiante gain factor for CoordinateConstraint

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.CoordinateConstraint.coordinate_offset	coordiante offset for CoordinateConstraint (w.r.t. ground or between two element coordinates); offset is ignored for Index 2 (setting of time integration) velocity level constraint
Connector.CoordinateConstraint.gain_factor	coordiante gain factor for CoordinateConstraint

Example

see file VelocityCoordinateConstraint.txt

```
l = 1 % m

rigidBody
{
  element_type= "Rigid3D"
  Graphics.body_dimensions= [1, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

coordinateConstraint
{
  element_type= "VelocityCoordinateConstraint"
  Coordinate1
  {
    element_number= nRigid  %element number for coordinate 1
    local_coordinate= 1  %local coordinate of element 1
  }
}
AddConnector(coordinateConstraint)
```

3.3.4 MultiCoordConstraint

Short description

The MultiCoordConstraint is an extension of CoordinateConstraint and constrains more than two elements.Only the lagrange multiplier formulation is implemented and no penalty formulation.

Equations

position constraint (index 3 solver)
$C = k_1\left(q^{el1}_{c1} - q^{el1}_{c1,0}\right) - k_2\left(q^{el2}_{c2} - q^{el2}_{c2,0}\right) - k_3... - d = 0$

velocity constraint - index reduction (index 2 solver)
$C = k_1\,\dot{q}^{el1}_{c1} - k_2\dot{q}^{el2}_{c2} - k_3... = 0$

Langrange multiplier
first element:
$\frac{\partial C}{\partial \mathbf{q}^{el1}}^T = [0\,.........\,0,\ k_1\,,0\,...\,0]$ ... with k_1 at index c1
i-th element:
$\frac{\partial C}{\partial \mathbf{q}^{eli}}^T = [0\,...\,0\,, -k_i\,, 0\,.........\,0]$ ... with $-k_i$ at index ci

Description:
$k_1, k_2, ..., k_i$ ... coordinate gain factors
d ... coordinate offset (for index 2 solvers not used)
q^{elj}_{ci} ... ci^{th} coordinate of element j
$q^{elj}_{ci,0} = q^{elj}_{ci}\,(t = 0)$ ... ci^{th} coordinate of element j at initialization
C ... Lagrange equation

Description of the different modi

relative or absolute to initial values	Only important for max index 3 solvers. If relative_to_inital_values is set to 1: Equation above is used. If set to 0: Simplified equation is used ($q^{el1}_{c1,0} = q^{el2}_{c2,0} = \ldots = 0$).

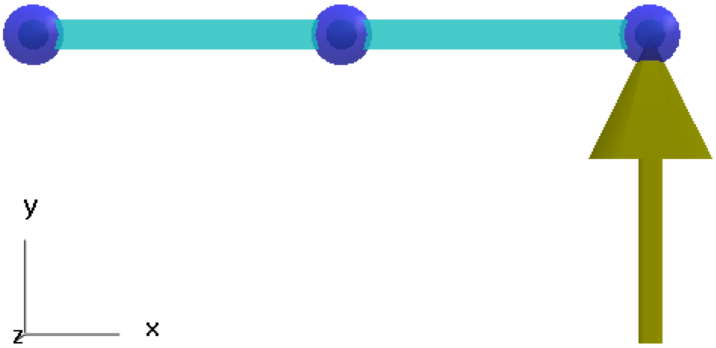


Figure 3.22: 3 point masses are constrained to each other with MultiCoordConstraints to obtain the same behaviour as a rigid body, see the provided example code.

Data objects of MultiCoordConstraint:

Data name	**type**	**R**	**default**	**description**
element_type	string		"MultiCoordConstraint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"MultiCoordConstraint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		0.1	General drawing size of constraint
element_numbers	vector		[]	element numbers to constrain
local_coordinates	vector		[]	local coordinates of elements to be constrained
coord_gain_factors	vector		[]	coordinate gain factor k for each element, see documentation section equation
coord_offset	double		0	coordinate offset d, see documentation section equation
relative_to_inital_values	bool		1	flag == 1: full equation is used, see documentation; flag == 0: the init state values qi0 and qj0 are neglected.

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-1

Example

see file MultiCoordinateConstraint.txt

```
r = 0.1        % [m] distance between point masses
m_center = 2    % [kg] mass of point mass in the center
m_outer = 0.5   % [kg] mass of point mass at the outer edges

Force.load_type= "ForceVector2D"
Force.force_vector= [0,1]
Force.position= [0, 0]
nLoad = AddLoad(Force)

PointMass.element_type= "Mass2D"
PointMass.loads= []          % no load at left mass
PointMass.Graphics.radius= r/10
PointMass.Initialization.initial_position= [0, 0]
PointMass.Physics.mass= m_outer
nE_mLeft=AddElement(PointMass)

PointMass.loads= [nLoad]        % force vector acting on right mass
PointMass.Initialization.initial_position= [2*r, 0]
nE_mRight=AddElement(PointMass)

PointMass.loads= []          % no load at the center mass
PointMass.Initialization.initial_position= [r, 0]
PointMass.Physics.mass= m_center
nE_mCenter=AddElement(PointMass)

MultiCC
{
  element_type= "MultiCoordConstraint"
  Graphics.draw_size = r/10
  element_numbers= [nE_mCenter,nE_mLeft,nE_mRight]
  local_coordinates= [1,1,1]      % constrain x-directions
  coord_gain_factors= [1,0.5,0.5]   % x of center mass is average of outer masses
}
AddConnector(MultiCC)

MultiCC.local_coordinates= [2,2,2]  % constrain y-directions
AddConnector(MultiCC)

ConstLength
{
  element_type= "SpringDamperActuator2D"
  Graphics.show_connector = 0
  Physics.spring_length= r          % keep the distance constant
  Physics.Linear.spring_stiffness= 1000   % high stiffness
  Physics.Linear.damping= 10          % sufficient damping
  Position1.element_number= nE_mCenter    % center - left
  Position2.element_number= nE_mLeft
}
AddConnector(ConstLength)

ConstLength.Position2.element_number= nE_mRight  % center - right
AddConnector(ConstLength)

% compare with rigid body formulation (reference solution) ================
Force.name= "Load for Rigid"
Force.position= [r, 0]            % local position of load is different
nLoadRigid = AddLoad(Force)

RigidBody.element_type= "Rigid2D"
RigidBody.loads= [nLoadRigid]
RigidBody.Graphics.body_dimensions= [2*r, r/10, r/10]
RigidBody.Graphics.RGB_color = [0.1,0.8,0.8]
RigidBody.Physics.moment_of_inertia= 2*m_outer*r*r  % [I_ZZ]
RigidBody.Physics.mass= 2*m_outer+m_center      % total mass of the body in kg
RigidBody.Initialization.initial_position= [r, 0]
AddElement(RigidBody)
```

3.3.5 SlidingPointJoint

Short description

This joint enables sliding of a fixed point of a body i along the x - axis of another body j. Both body i and body j can be flexible or rigid. Body j can contain more than one elements. No rotations are constrained at all. Only a Lagrangian formulation is implemented, the penalty formulation is not implemented yet. A MaxIndex 2 and 3 formulation exists.

Degrees of freedom

The vector of the DOF contains the sliding parameter s, its time derivative $\dot{s}$ and the vector of the Lagrangian parameters $\lambda = [\lambda_1 \lambda_2 \lambda_3]^{\mathbf{T}}$. The Lagrange parameters λ_1 to λ_3 are representing the sliding forces in the global coordinate system.

$$\mathbf{q} \;=\; \begin{bmatrix} s & \dot{s} & \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix}^{\mathbf{T}} \tag{3.15}$$

Equations

positions:

$$\mathbf{x}^i \;=\; \begin{bmatrix} x_1^i & x_2^i & x_3^i \end{bmatrix}^{\mathbf{T}} \tag{3.16}$$

$$\mathbf{x}^j \;=\; \begin{bmatrix} x_1^j = s & x_2^j & x_3^j \end{bmatrix}^{\mathbf{T}} \tag{3.17}$$

constraint equation - position level

$$\mathbf{C} \;=\; \begin{bmatrix} \mathbf{r}^i\left(\mathbf{x}^i\right) - \mathbf{r}^j\left(\mathbf{x}^j\right) \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \end{bmatrix} = \mathbf{0} \tag{3.18}$$

The first three constraints restrict the motion of the sliding point on body i and j. The fourth constraint equation ensures, that there is no force in the sliding direction.

constraint equation - velocity level:

$$\mathbf{C} \;=\; \begin{bmatrix} \frac{\partial \mathbf{r}^i\left(\mathbf{x}^i\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\dot{s} \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \end{bmatrix} = \mathbf{0} \tag{3.19}$$

To obtain the constraints for velocity level, the first three equations are differentiated with respect to time. The sliding parameter s is also a function of time. The fourth constraint equation is equal to the position level equation.

Description of the different modi

sliding along a single body	The vector Geometry.element_numbers is equal to $[en1, en2]$. Index Geometry.elemind must be 1.
sliding along more than one body	Geometry.element_numbers has to be set to $[en1, en2_1, en2_2, ..., en2_n]$. Geometry.elemind is the body j index of the element in inital configuration, e.g. for $en2_2$ the elemind is 2.

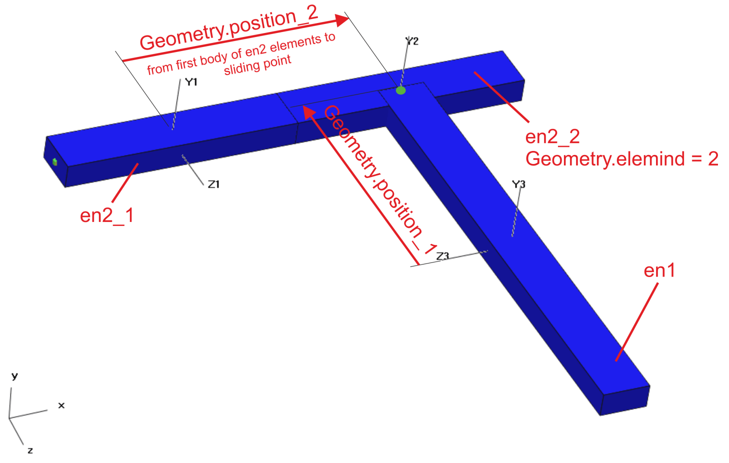


Figure 3.23: SlidingPointJoint

Data objects of SlidingPointJoint:

Data name	type	R	default	description

element_type	string		"SlidingPointJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"SlidingPointJoint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs

Graphics

Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	Drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.

Geometry

Geometry.elemind	integer		1	Index of the initial sliding body.
Geometry.position_1	vector		[0, 0, 0]	Vector from the center of body number 1 (en1) to the sliding point in the local body 1 coordinate system.
Geometry.position_2	vector		[0, 0, 0]	Vector from the center of the first body of en2 array to the sliding point in the local body 2 coordinate system.
Geometry. element_numbers	vector		[1, 2]	Element numbers: [en1,en2_1,en2_2,...,en2_n].

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-5
Internal.first_order_variable	first order variables of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-4
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3 corresponds to force in global x-y-z direction

Example

see file SlidingPointJoint.txt

```
l = 1 %m
k = 0.02 %nominal value k=1; decreased stiffness for demonstration!

load % define the load
{
  load_type = "ForceVector3D"
  position = [l/2,0,0]
  force_vector = [0,0,1000] % magnitude and direction
}
nLoad=AddLoad(load)
```

```
material
{
  material_type = "Beam3DProperties"
  cross_section_type = 1 % rectangular cross section
  cross_section_size = [0.05,0.1]
  density = 7850 %kg/m^3
  EA = 2100000000*k %N
  EIy = 1750000*k %Nm^2
  EIz = 1750000*k %Nm^2
  GAky = 800000000*k %N
  GAkz = 800000000*k %N
  GJkx = 500000000*k %N*m^2
  RhoA = 78.5 %kg/m^2
  RhoIx = 0.1 %kg*m
  RhoIy = 0.1 %kg*m
  RhoIz = 0.1 %kg*m
}
nMaterial = AddBeamProperties(material)

node
{
  node_type = "Node3DRxyz"
}
n1 = AddNode(node)

node.Geometry.reference_position = [l/2,0,0]
n2 = AddNode(node)

node.Geometry.reference_position = [l,0,0]
n3 = AddNode(node)

node.Geometry.reference_position = [3*l/4,0,0]
node.Geometry.reference_rot_angles = [0,-Pi/2,0] %bryant angles
n4 = AddNode(node)

node.Geometry.reference_position = [3*l/4,0,l]
n5 = AddNode(node)

beam
{
  element_type= "LinearBeam3D"
  Physics.material_number = nMaterial
  Geometry.node_1 = n1
  Geometry.node_2 = n2
}
nBeam12 = AddElement(beam)

beam.Geometry.node_1 = n2
beam.Geometry.node_2 = n3
nBeam23 = AddElement(beam)
```

```
beam.loads = [nLoad]
beam.Geometry.node_1 = n4
beam.Geometry.node_2 = n5
nBeam45 = AddElement(beam)

slidingJoint
{
  element_type = "SlidingPointJoint"
  Geometry
  {
    elemind = 2  %number of the initial sliding body (2nd body).
    position_1 = [-1/2, 0, 0]
    %vector from the center of body number 1 (en1) to the sliding point
    %in the local body 1 coordinate system.
    position_2 = [1/2, 0, 0]  %vector from the center of the first body
    %of en2 array to the sliding point in the local body 2 coordinate system.
    element_numbers = [nBeam45,  nBeam12,nBeam23]
    %Element numbers: [en1,  en2_1,en2_2].
  }
}
AddConnector(slidingJoint)

rigidJoint
{
  element_type= "RigidJoint"
  Position1
  {
    element_number= nBeam12  %constrained element
    position= [-1/4, 0, 0]  %local position.
  }
}
AddConnector(rigidJoint)
```

3.3.6 SlidingPrismaticJoint

Short description

This joint enables sliding of a fixed point of a body i along the x - axis of another body j. Both body i and body j can be flexible or rigid. Body j can contain more than one element. The difference to the SlidingPointJoint is that the relative rotation between the bodies is also constrained. A Lagrangian formulation is used for both stiff and springy constrained rotation. For the position constraint only a stiff formulation exists. A penalty formulation is not implemented yet. There is a MaxIndex 2 and 3 formulation implemented.

Degrees of freedom

The vector of the DOF contains the sliding parameter s, its time derivative $\dot{s}$ and the vector of the Lagrangian parameters $\lambda = [\lambda_1 \lambda_2 \lambda_3]^{\mathbf{T}}$. The Lagrange parameters λ_1 to λ_3 are representing the sliding forces in the global coordinate system. The three Lagrangian parameters λ_4 to λ_6 are the sliding moments about the global coordinate system axes.

$$\mathbf{q} \;=\; \begin{bmatrix} s & \dot{s} & \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 \end{bmatrix}^{\mathbf{T}} \tag{3.20}$$

Equations

At initialization the unit vectors of the global coordinate system are transformed to the local coordinate system of each body and the vectors $\mathbf{v}_1^i$, $\mathbf{v}_2^i$ and $\mathbf{v}_3^i$ for body i and $\mathbf{v}_1^j$, $\mathbf{v}_2^j$ and $\mathbf{v}_3^j$ for body j are obtained. The vectors are fixed in the body coordinate system. The position vectors are the same as for the SlidingPointJoint.

constraint equation - position level (stiff connection)

$$\mathbf{C} \quad = \quad \begin{bmatrix} \mathbf{r}^i\left(\mathbf{x}^i\right) - \mathbf{r}^j\left(\mathbf{x}^j\right) \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \\ \mathbf{v}_2^j\mathbf{v}_3^i \\ \mathbf{v}_3^j\mathbf{v}_1^i \\ \mathbf{v}_2^j\mathbf{v}_1^i \end{bmatrix} = \mathbf{0} \tag{3.21}$$

constraint equation - position level (springy connection)

$$\mathbf{C} \quad = \quad \begin{bmatrix} \mathbf{r}^i\left(\mathbf{x}^i\right) - \mathbf{r}^j\left(\mathbf{x}^j\right) \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \\ \mathbf{v}_2^j\mathbf{v}_3^i k_1 + \left(\dot{\mathbf{v}}_2^j\mathbf{v}_3^i + \mathbf{v}_2^j\dot{\mathbf{v}}_3^i\right) d_1 + \lambda_4 \\ \mathbf{v}_3^j\mathbf{v}_1^i k_1 + \left(\dot{\mathbf{v}}_3^j\mathbf{v}_1^i + \mathbf{v}_3^j\dot{\mathbf{v}}_1^i\right) d_2 + \lambda_5 \\ -\mathbf{v}_2^j\mathbf{v}_1^i k_1 - \left(\dot{\mathbf{v}}_2^j\mathbf{v}_1^i + \mathbf{v}_2^j\dot{\mathbf{v}}_1^i\right) d_3 + \lambda_6 \end{bmatrix} = \mathbf{0} \quad , \tag{3.22}$$

constraint equation - velocity level (stiff connection)

$$\mathbf{C} \quad = \quad \begin{bmatrix} \frac{\partial \mathbf{r}^i\left(\mathbf{x}^i\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\dot{s} \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \\ \dot{\mathbf{v}}_2^j\mathbf{v}_3^i + \mathbf{v}_2^j\dot{\mathbf{v}}_3^i \\ \dot{\mathbf{v}}_3^j\mathbf{v}_1^i + \mathbf{v}_3^j\dot{\mathbf{v}}_1^i \\ \dot{\mathbf{v}}_2^j\mathbf{v}_1^i + \mathbf{v}_2^j\dot{\mathbf{v}}_1^i \end{bmatrix} = \mathbf{0} \tag{3.23}$$

constraint equation - velocity level (springy connection)

$$\mathbf{C} \quad = \quad \begin{bmatrix} \frac{\partial \mathbf{r}^i\left(\mathbf{x}^i\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial t} - \frac{\mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\dot{s} \\ \frac{\partial \mathbf{r}^j\left(\mathbf{x}^j\right)}{\partial x_1^j}\lambda \\ \mathbf{v}_2^j\mathbf{v}_3^i k_1 + \left(\dot{\mathbf{v}}_2^j\mathbf{v}_3^i + \mathbf{v}_2^j\dot{\mathbf{v}}_3^i\right) d_1 + \lambda_4 \\ \mathbf{v}_3^j\mathbf{v}_1^i k_1 + \left(\dot{\mathbf{v}}_3^j\mathbf{v}_1^i + \mathbf{v}_3^j\dot{\mathbf{v}}_1^i\right) d_2 + \lambda_5 \\ -\mathbf{v}_2^j\mathbf{v}_1^i k_1 - \left(\dot{\mathbf{v}}_2^j\mathbf{v}_1^i + \mathbf{v}_2^j\dot{\mathbf{v}}_1^i\right) d_3 + \lambda_6 \end{bmatrix} = \mathbf{0} \tag{3.24}$$

Description of the different modi

sliding along a single body	The vector Geomety.element_numbers is equal to $[en1, en2]$. Index Geomety.elemind must be 1.
sliding along more than one body	Geomety.element_numbers has to be set to $[en1, en2_1, en2_2, ..., en2_n]$. Geomety.elemind is the body j index of the element in inital configuration, e.g. for $en2_2$ the elemind is 2.
stiff constrained rotation	Physics.use_penalty_formulation is set to 0.
springy constrained rotation	Physics.use_penalty_formulation is set to 1. The values for stiffness and damping must be set in Physics.Penalty folder.

Data objects of SlidingPrismaticJoint:

Data name	**type**	**R**	**default**	**description**
element_type	string		"SlidingPrismaticJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"SlidingPrismaticJoint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	Drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Geometry				
Geometry.position_1	vector		[0, 0, 0]	Vector from the center of body number 1 (en1) to the sliding point in the local body 1 coordinate system.
Geometry.position_2	vector		[0, 0, 0]	Vector from the center of the first body of en2 array to the sliding point in the local body 2 coordinate system.
Geometry. element_numbers	vector		[1, 2]	Element numbers: [en1,en2_1,en2_2,...,en2_n].
Geometry.elemind	integer		1	Index of the initial sliding body.
Physics				
Physics. use_penalty_formulation	bool		1	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.k1	double		1e+005	Stiffness for rotation about global x - axis.
Physics.Penalty.k2	double		1e+005	Stiffness for rotation about global y - axis.
Physics.Penalty.k3	double		1e+005	Stiffness for rotation about global z - axis.
Physics.Penalty.d1	double		100	Damping of rotation about global x - axis.
Physics.Penalty.d2	double		100	Damping of rotation about global x - axis.
Physics.Penalty.d3	double		100	Damping of rotation about global x - axis.

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-8
Internal.first_order_variable	first order variables of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-7
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.SlidingPrismaticJoint.sliding_parameter	internal sliding parameter s
Connector.SlidingPrismaticJoint.sliding_parameter_p	internal time derivative of sliding parameter s

Example

see file SlidingPrismaticJoint.txt

```
... copy this part from "SlidingPointJoint" example

slidingJoint
{
  element_type = "SlidingPrismaticJoint"
  Geometry
  {
    elemind = 2  %number of the initial sliding body (2nd body).
    position_1 = [-1/2, 0, 0]
    %vector from the center of body number 1 (en1) to the sliding point
    %in the local body 1 coordinate system.
    position_2 = [1/2, 0, 0]  %vector from the center of the first body
    %of en2 array to the sliding point in the local body 2 coordinate system.
    element_numbers = [nBeam45,  nBeam12,nBeam23]
    %Element numbers: [en1,  en2_1,en2_2].
  }
}
AddConnector(slidingJoint)

... copy this part from "SlidingPointJoint" example
```

3.3.7 Rope3D

Short description

Elastic rope that is always under tension and can be fixed to multiple bodies and ground. There are 2 different kinds of suspensions points. Suspension points fixed on the ground are defined with the element number 0 and the global position. Suspension points on bodies are defined with the element number and the corresponding local position.

Limitations

The rope is assumed to be straight between 2 suspension points. No negative forces can be transmitted by a rope. The computation of the time derivative of the length of the rope is just an approximation. Therefore the damping of the rope may be represented slightly incorrect.

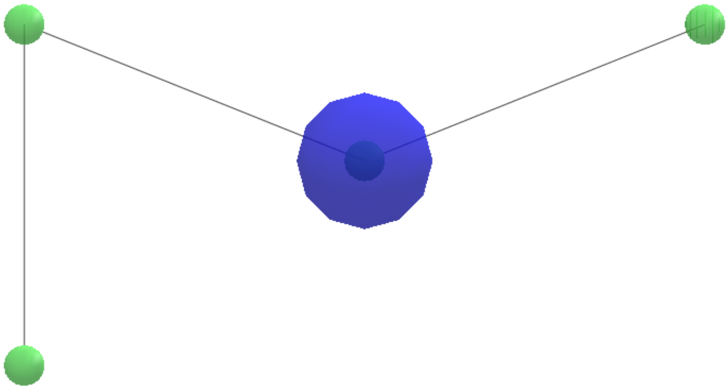


Figure 3.24: Point mass with rope

Data objects of Rope3D:

Data name	type	R	default	description
element_type	string		"Rope3D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Rope3D"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Geometry				
Geometry.rope_length	double	**R**	1	initial length l0 of rope (computed automatically)
Geometry. element_numbers	vector		[0, 0]	element numbers of the suspension points
Geometry.positions	matrix		[0, 0, 0; 1, 0, 0]	(local) positions of the suspension points
Physics				
Physics.Penalty				
Physics.Penalty.damping	double		0	damping coefficient for viscous damping (F = d*v), applied in all constrained directions
Physics.Penalty. rope_stiffness	double		0	[N] stiffness parameter c of the rope, F = c * (l-l0)/l0
Physics.Penalty. spring_stiffness	double	**R**	0	total stiffness c1 of the rope F = c1 * (l-l0)

Observable special values:

For more information see section 3.1

value name	description
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-3

Connector.Rope.coiled_length	(additional) length of the rope that is provided by a coil. length = rope_length + coiled_length
Connector.Rope.force	force in the rope
Connector.Rope.rope_length	length of the rope

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.Rope.coiled_length	(additional) length of the rope that is provided by a coil. length = rope_length + coiled_length

Example

see file Rope3D.txt

```
Mass
{
  element_type= "Mass3D"
  Physics.mass= 1
  Initialization.initial_position= [0.5, 0.8, 0]
}
nMass = AddElement(Mass)

rope
{
  element_type= "Rope3D"
  name= "Rope3D"  %name of the element
  Graphics.draw_size = 0.03
  Physics
  {
  Penalty
  {
    rope_stiffness= 1e3
    damping= 10
  }
  }
  Geometry
  {
  element_numbers= [0, 0, nMass, 0]  %element numbers of the suspension points
  positions= [0, 0.5, 0; 0, 1, 0; 0,0,0; 1,1,0]
  }
}
nRope = AddConnector(rope)
```

3.3.8 FrictionConstraint

Short description

The FrictionConstraint is acting on an arbitraty coordinate, including rotations. It can be used to connect two elements to each other or one element to ground. Up to a specified threshold of the force, the constraint is sticking, which is either realized by a spring-damper formulation (penalty formulation) or with an algebraic equation (lagrange formulation). Above this threshold, a constant friction force is applied during the sliding phase. Alternatively sticking can be switched off and a coulomb friction force, with a transition region for very small velocities, can be applied.

Equations

Lagrange formulation:

sticking:
position constraint (index 3 solver)
2 elements (coordinate to coordinate): $C = q_i^{el1} - q_j^{el2} - x_0 = 0$
1 element (coordinate to ground): $C = q_i^{el1} - x_0 = 0$
velocity constraint - index reduction (index 2 solver)
2 elements (coordinate to coordinate): $C = \dot{q}_i^{el1} - \dot{q}_j^{el2} = 0$,
1 element (coordinate to ground): $C = \dot{q}_i^{el1} = 0$

sliding: $C = \lambda - \mu_{kin} F_n = 0$

Penalty formulation:

sticking:
2 elements (coordinate to coordinate):$F_{st} = c\left(q_i^{el1} - q_j^{el2} - x_0\right) + d\left(\dot{q}_i^{el1} - \dot{q}_j^{el2}\right)$
1 element (coordinate to ground):$F_{st} = c\left(q_i^{el1} - x_0\right) + d\dot{q}_i^{el1}$

sliding: $F_{sl} = \mu_{kin} F_n$

Description:
q_i^{el1} ... i^{th} coordinate of element 1
q_j^{el2} ... j^{th} coordinate of element 2
x_0 ... last sticking position (updated at every slide-stick transition)

Description of the different modi

sticking	During sticking phase, the constraint is implemented as spring-damper, with the force F_{st}, the spring stiffness c and the damping coefficient d or alternatively with one algebraic equation in lagrange mode.
sliding	During sliding phase, a constant friction force F_{sl} is applied. F_{sl} depends on the normal force F_n.If the flag keep_sliding is active, then a transition region for small velocities is used.

Additional notes

The switching from sticking phase to sliding phase is done automatically, as soon as $F_{st} > \mu_{st} F_n$. The switching to sticking phase is performed when the absolute value of the velocity v is smaller than the specified velocity_tolerance.
If the solver does not converge close to the switching points, set the solver option SolverOptions.Discontinuous.ignore_max_iterations = 1.
If you are using index 2 solver it is advised to use RadauIIA and not LobattoIIIA. LobattoIIIA may lead to oscillations of the friction force and therefore unwanted stick-slip transistions.
If you are using the FrictionConstraint in order to constrain rotations, problems may occur when the change of the angle is discontinous, e.g. if it exceeds pi/2.

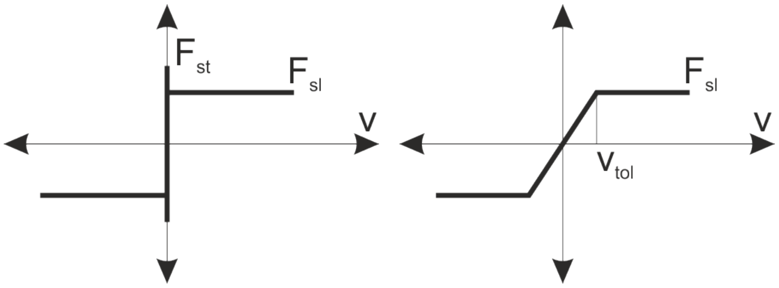


Figure 3.25: FrictionConstraint with friction forces F_{st} and F_{sl}, with sticking (left figure, keep_sliding = 0) and without sticking (right figure, keep_sliding = 1).

Data objects of FrictionConstraint:

Data name	type	R	default	description
element_type	string		"FrictionConstraint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"FrictionConstraint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	Drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Physics				
Physics.use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.normal_force	double		0	constant normal force Fn
Physics.velocity_tolerance	double		1e-005	If velocity is below this value, sticking starts, or if 'keep sliding' is active, the transition region is used.
Physics.fr_coeff_st	double		0.15	static friction coefficient, used to determine the threshold when sliding starts.
Physics.fr_coeff_kin	double		0.1	kinematic friction coefficient, used to calculate the constant force during sliding phase.
Physics.keep_sliding	bool		0	The constraint will never go to modus 'stick'.
Physics.Penalty				

Physics.Penalty. spring_stiffness	double		0	spring stiffness c, only used during sticking phase!
Physics.Penalty.damping	double		0	damping coefficient d for viscous damping, only used during sticking phase!
Initialization				
Initialization. initial_sliding_velocity	double		0	Initial (relative) sliding velocity between the two kinematic pairs. If absolute value is smaller than 'velocity tolerance' then the constraint starts with 'sticking'.
Coordinate1				
Coordinate1. element_number	integer		0	element number for coordinate 1
Coordinate1. local_coordinate	integer		1	Local coordinate of element 1 to be constrained
Coordinate2				
Coordinate2. element_number	integer		0	element number for coordinate 2; for ground joint, set element number to zero
Coordinate2. local_coordinate	integer		1	Local coordinate of element 2 to be constrained

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-1
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-4
Connector.FrictionConstraint.sticking	1 if sticking, 0 if sliding
Connector.FrictionConstraint.force_forward	force, applied to the kinematic pairs due to the constraint
Connector.FrictionConstraint.force_forward_abs	absolute value of the force, applied to the kinematic pairs due to the constraint
Connector.FrictionConstraint.force_normal	force, applied to the kinematic pairs due to the constraint

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.FrictionConstraint.force_normal	force, applied to the kinematic pairs due to the constraint

Example

see file FrictionConstraint.txt

```
force
{
  load_type = "ForceVector3D"
  force_vector= [1, 0, 0]
  load_function_type= 1  %time dependency of the load: 1..MathFunction
  MathFunction
  {
    piecewise_mode= 1  %modus for piecewise interpolation: 1=linear
    piecewise_points= [0,0.08,0.081,0.2]  %supporting points
    piecewise_values= [0,50,-50,0]  %values at supporting points
  }
}
nLoad=AddLoad(force)

test_mass
{
  element_type = "Mass3D"
  Physics.mass = 1
  loads=[nLoad]
}
nMass = AddElement(test_mass)

friction
{
  element_type= "FrictionConstraint"
  name= "FrictionConstraint"
  Physics
  {
    normal_force= 10
    fr_coeff_st= 0.15
    fr_coeff_kin= 0.1
  }
  Coordinate1
  {
    element_number= nMass  %element number for coordinate 1
    local_coordinate= 1  %Local coordinate of element 1 to be constrained
  }
}
nFriction=AddConnector(friction)

sensfriction
{
  name= "sticking"
  sensor_type= "ElementSensor"
  element_number= nFriction
  value= "Connector.FrictionConstraint.sticking"
}
AddSensor(sensfriction)
sensfriction.name="friction_force"
sensfriction.value= "Connector.FrictionConstraint.force_forward"
nSensFriction = AddSensor(sensfriction)

SolverOptions
{
  end_time = 0.2
  TimeInt.max_step_size = 1e-5
  Newton.relative_accuracy = 1
  Newton.use_modified_newton= 1
  Linalg.use_sparse_solver = 1
  Discontinuous.ignore_max_iterations = 1
}
ViewingOptions.Loads.show_loads=1
```

3.3.9 Contact1D

Short description

Contact1D realizes a contact formulation between two elements or one element and ground. Only one coordinate (direction) is considered per element.

Geometry

Figure 3.26 shows the meaning of the values local coordinate and position in the case of a ground constraint. The only direction which is considered is that defined by Coordinate1.local coordinate. Figure 3.26 shows the case for 2 elements. The value Physics.direction, dir in the following equations, is used to define how the elements are located w.r.t. each other.
ATTENTION: Be carefull when using coordinates which do not represent a position!

Equations

Some general definitions:

$$pos = coordinate + localposition \tag{3.25}$$

$$u = dir(pos_1 - pos_2) \tag{3.26}$$

$$v = dir(vel_1 - vel_2) \tag{3.27}$$

Mode 1:
if $u \geq 0$:

$$F = 0 \tag{3.28}$$

else:

$$F = dir(cu + dv) \tag{3.29}$$

Description of the different modi

Mode 1	Penalty Formulation with spring and damper. The bodies will penetrate slightly according to the spring stiffness. Results may depend on chosen step size!
Mode 2	Lagrange Formulation (not implemented yet)

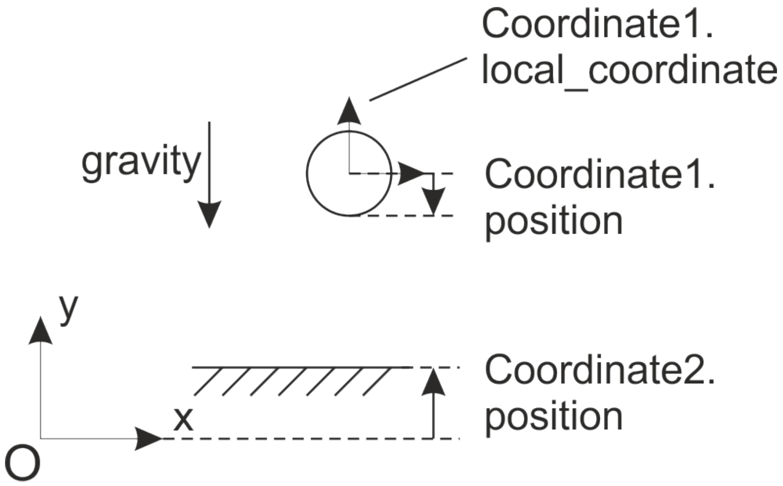


Figure 3.26: Description of the geometry options in the case of a ground constraint.

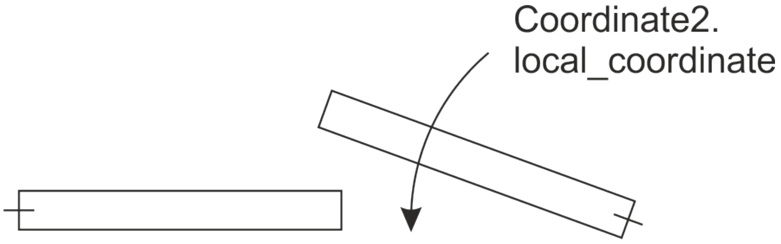


Figure 3.27: Description of the geometry options in the case of 2 elements.

Data objects of Contact1D:

Data name	type	R	default	description
element_type	string		"Contact1D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"Contact1D"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size	double		-1	Drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Physics				
Physics.direction	double		1	Direction of the contact: +1 if the first body is on top, or else -1
Physics.mode	integer		1	mode of computation
Physics.Mode1				
Physics.Mode1. spring_stiffness	double		0	spring stiffness c
Physics.Mode1.damping	double		0	damping coefficient d for viscous damping
Coordinate1				
Coordinate1. local_coordinate	integer		1	Local coordinate of element 1 to be constrained
Coordinate1.position	double		0	Local position at which contact occurs
Coordinate1. element_number	integer		1	element number for coordinate 1; set to zero if you use nodal coordinates!

Coordinate1. node_number	integer		0	(just used if element number = 0) node number for coordinate 1
Coordinate2				
Coordinate2. local_coordinate	integer		1	Local coordinate of element 2 to be constrained (not used if ground constraint)
Coordinate2.position	double		0	Local (or global if ground) position at which contact occurs
Coordinate2. element_number	integer		0	element number for coordinate 2; for ground joint or nodal coordinates, set element number to zero
Coordinate2. node_number	integer		0	(just used if element number = 0) node number for coordinate 2; for ground joint, set node number to zero

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-2
Internal.second_order_variable	second order variables of the element. range: 1-1
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-1
Connector.Contact.force	the force applied to the coordinates due to the contact

Example

see file Contact1D.txt

```
load.load_type= "Gravity"
load.gravity_constant= -9.81
nLoad = AddLoad(load)

r = 0.1
mass  % define point mass
{
  element_type= "Mass2D"
  loads= [nLoad]
  Initialization.initial_position= [1,0]
  Physics.mass= 1
  Graphics.radius = r
}
nElem1 = AddElement(mass)

contact % add contact
{
  element_type= "Contact1D"
  Graphics.draw_size = 0.01
  Physics
  {
    mode= 1           % mode of computation
    Mode1.spring_stiffness= 1e6 % spring stiffness c
    Mode1.damping= 5e2      % damping coefficient d for viscous damping
  }
  Coordinate1
  {
    local_coordinate= 1     % coord 1 of element 1 is x-direction!
    position= -r           % offset in x-direction
    element_number= nElem1    % element number for coordinate 1
  }  % ground constraint without offset: no entries for Coordinate 2 needed
}
AddConnector(contact)

SolverOptions.Discontinuous.absolute_accuracy = 0.001
SolverOptions.end_time = 2
```

3.3.10 GenericBodyJoint

Short description

The GenericBodyJoint constrains two elements at a local position each. If only one element is specified (second element 0), a ground GenericBodyJoint is realized. A penalty and Lagrange formulation is available.
The constraint forces and moments are applied as follows:
Connecting element to element:
The constraint forces and moments are applied on both elements at the position of the connection point of the second element.
Connecting element to ground:
The constraint forces are applied at the position of the connection point of the element.

Equations

Lagrange equations:
The constraint equations for translation are

$$\mathbf{C}_{trans} = \mathbf{A}^T \left(\mathbf{x}_1 - \mathbf{x}_2\right) = \mathbf{0}$$

Each equation in $\mathbf{C}_{trans}$ corresponds to a constrained direction. Hence only those equations corresponding to the constrained directions are considered. If all directions are constrained, $\mathbf{C}_{trans}$ simplifies to

$$\mathbf{C}_{trans} = \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0}$$

since $\mathbf{A}^T \left(\mathbf{x}_1 - \mathbf{x}_2\right) = \mathbf{0} \iff \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0}$.
If all rotations are constrained, then the constraint equations for rotation are

$$\mathbf{C}_{rot} = \begin{pmatrix} \mathbf{e}_y^j \cdot \mathbf{e}_z^i \\ \mathbf{e}_x^j \cdot \mathbf{e}_z^i \\ \mathbf{e}_x^j \cdot \mathbf{e}_y^i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

If the rotation about the x-axis is not constrained, then

$$\mathbf{C}_{rot} = \begin{pmatrix} \mathbf{e}_x^j \cdot \mathbf{e}_z^i \\ \mathbf{e}_x^j \cdot \mathbf{e}_y^i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

If the rotation about the y-axis is not constrained, then

$$\mathbf{C}_{rot} = \begin{pmatrix} \mathbf{e}_y^j \cdot \mathbf{e}_z^i \\ \mathbf{e}_y^j \cdot \mathbf{e}_x^i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

If the rotation about the z-axis is not constrained, then

$$\mathbf{C}_{rot} = \begin{pmatrix} \mathbf{e}_z^j \cdot \mathbf{e}_y^i \\ \mathbf{e}_z^j \cdot \mathbf{e}_x^i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Where

$\mathbf{x}_1$	position of connection point on body 1 in global coordinates
$\mathbf{x}_2$	position of connection point on body 2 in global coordinates, or if constraint connects element to ground then connection point of ground in global coordinates
$\mathbf{v}_1$	time derivative of $\mathbf{x}_1$
$\mathbf{v}_2$	time derivative of $\mathbf{x}_2$
$\mathbf{A}$	rotation matrix from local joint coordinates to global coordinates. $\mathbf{A} = \mathbf{Q}_i\mathbf{J}$.
$\mathbf{B}$	$\mathbf{B} = \mathbf{Q}_j\mathbf{J}^\star$
$\mathbf{Q}_i$	rotation matrix from local coordinate system of body 1 to global coordinates
$\mathbf{Q}_j$	rotation matrix from local coordinate system of body 2 to global coordinates
$\mathbf{J}$	joint local frame
$\mathbf{J}^\star$	$\mathbf{J}^\star = \mathbf{Q}_j^T \mid_{t=0} \mathbf{Q}_i \mid_{t=0} \mathbf{J}$
$\mathbf{e}_x^i, \mathbf{e}_y^i, \mathbf{e}_z^i$	$\begin{pmatrix} \mathbf{e}_x^i & \mathbf{e}_y^i & \mathbf{e}_z^i \end{pmatrix} = \mathbf{A}$
$\mathbf{e}_x^j, \mathbf{e}_y^j, \mathbf{e}_z^j$	$\begin{pmatrix} \mathbf{e}_x^j & \mathbf{e}_y^j & \mathbf{e}_z^j \end{pmatrix} = \mathbf{B}$

Penalty equations:
The stiffness and damping force is given by

$$\mathbf{f} = \mathbf{A}\mathbf{K}_{trans}\mathbf{A}^T\mathbf{u} + \mathbf{A}\mathbf{D}_{trans}\mathbf{A}^T\mathbf{v}$$

The stiffness and damping moment is given by

$$\mathbf{m} = \mathbf{K}_{rot}\varphi + \mathbf{D}_{rot}\omega$$

Where

$\mathbf{f}$	constraint force due to stiffness and damping
$\mathbf{m}$	constraint moment due to stiffness and damping
$\mathbf{K}_{trans}$	stiffness matrix for translation
$\mathbf{D}_{trans}$	damping matrix for translation
$\mathbf{K}_{rot}$	stiffness matrix for rotation
$\mathbf{D}_{rot}$	damping matrix for rotation
φ	relative angles between body 1 and body 2 or absolute angles of body 1 if body 1 is connected to ground
ω	relative angular velocities between body 1 and body 2 or absolute angular velocities of body 1 if body 1 is connected to ground.

If all rotations are constrained, linearized angles

$$\varphi = \begin{pmatrix} \varphi_x \\ \varphi_y \\ \varphi_z \end{pmatrix} = \begin{pmatrix} -\mathbf{e}_y^j \cdot \mathbf{e}_z^i \\ \mathbf{e}_x^j \cdot \mathbf{e}_z^i \\ -\mathbf{e}_x^j \cdot \mathbf{e}_y^i \end{pmatrix}.$$

and linearized angular velocities are used

$$\omega = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} -\dot{\mathbf{e}}_y^j \cdot \mathbf{e}_z^i - \mathbf{e}_y^j \cdot \dot{\mathbf{e}}_z^i \\ \dot{\mathbf{e}}_x^j \cdot \mathbf{e}_z^i + \mathbf{e}_x^j \cdot \dot{\mathbf{e}}_z^i \\ -\dot{\mathbf{e}}_x^j \cdot \mathbf{e}_y^i - \mathbf{e}_x^j \cdot \dot{\mathbf{e}}_y^i \end{pmatrix}$$

Limitations

It is strongly recommended to prefer the Lagrangian method for free rotation instead of penalty formulation to avoid simulation problems.

The constraint forces have to act for both bodies at the same position. This means, that if the constraint is in penalty mode, or if not all directions are constrained, the constraint forces need to be applied outside the connection point of at least one body. In case of the GenericBodyJoint, the constraint forces are applied at the connection position of the second element if two elements are constrained, or if one element is constrained, the constraint forces are applied at the connection position of the element.

So if connecting two elements with a GenericBodyJoint, the constraint forces have to be applied outside the first bodies connection point. For rigid bodies this is equivalent to applying the force at the connection point and applying a moment which compensates the moment induced by the shifting of the force. Applying forces on flexible bodies outside the connection point gives various problems, like what happens if the force is outside the body, etc.

Therefore flexible bodies are treated like rigid bodies and the force is applied to the connection point and a moment is applied, also on the position of the connection point, which compensates the moment induced by shifting the force.

If you need a constraint which allows the sliding of an element on a flexible body please use a SlidingPointJoint (3.3.5) or a SlidingPrismaticJoint (3.3.6).

Description of the different modi

element to ground	Position2.element_number AND Position2.node_number have to be equal to 0
element to element	Position2.element_number and/or Position2.node_number must not be equal to 0
Lagrange	Physics.use_penalty_formulation must be set to 0. Set the vector of constrained directions in Physics.Lagrange.constrained_directions ($[x, y, z]$, 1 = constrained, 0 = free). The directions are w.r.t the local body 1 joint coordinate system. Set the vector of constrained rotations in Physics.Lagrange.constrained_rotations ($[\phi_x, \phi_y, \phi_z]$, 1 = constrained, 0 = free). The rotations are about the axes of local body 1 joint coordinate system.

Penalty	Physics.use_penalty_formulation must be set to 1. In Physics.Penalty.stiffness_matrix and Physics.Penalty.damping_matrix all parameters for translational stiffness and damping w.r.t. local body 1 coordinate system can be set. In Physics.Penalty.stiffness_matrix_rotation and Physics.Penalty.damping_matrix_rotation all parameters for rotational stiffness and damping w.r.t. local body 1 coordinate system can be set.

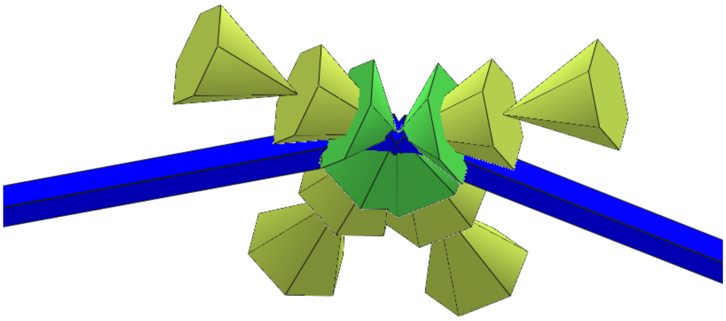


Figure 3.28: GenericBodyJoint

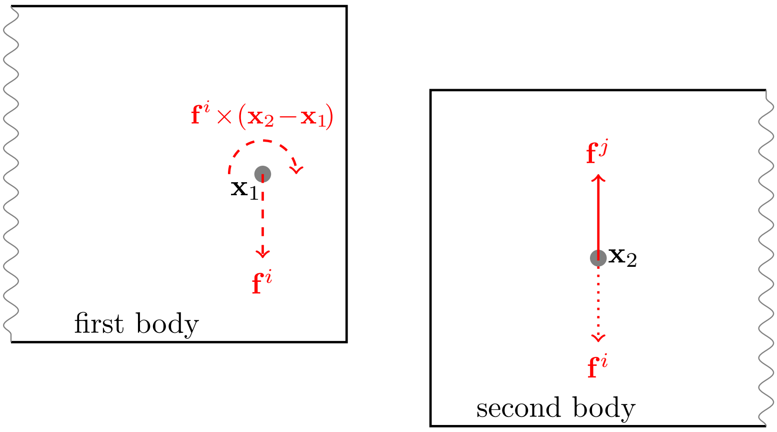


Figure 3.29: The constraint forces act for both bodies on the position of $\mathbf{x}_2$. The force acting on the first body is shifted to $\mathbf{x}_1$ (dashed) and a moment is applied to compensate the induced moment through shifting.

Data objects of GenericBodyJoint:

Data name	type	R	default	description
element_type	string		"GenericBodyJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!

name	string		"GenericBodyJoint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs

Graphics

Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Graphics.draw_size	double		-1	cone size for standard joint drawing
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]

Geometry

Geometry.joint_local_frame	matrix	**R**	[1, 0, 0; 0, 1, 0; 0, 0, 1]	
Geometry.joint_local_frame_in_bryant_angles	vector		[0, 0, 0]	Prerotate joint coordinate system w.r.t. local coordinate system of body 1 [phi x, phi y, phi z]. Rot. sequence: JA0i=A(phi z)A(phi y)A(phi x)

Physics

Physics.use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)

Physics.Penalty

Physics.Penalty.stiffness_matrix	matrix		[0, 0, 0; 0, 0, 0; 0, 0, 0]	3x3 matrix with stiffness parameters
Physics.Penalty.damping_matrix	matrix		[0, 0, 0; 0, 0, 0; 0, 0, 0]	3x3 matrix with damping parameters
Physics.Penalty.stiffness_matrix_rotation	matrix		[0, 0, 0; 0, 0, 0; 0, 0, 0]	3x3 matrix with stiffness parameters for rotation
Physics.Penalty.damping_matrix_rotation	matrix		[0, 0, 0; 0, 0, 0; 0, 0, 0]	3x3 matrix with damping parameters for rotation

Physics.Lagrange

Physics.Lagrange.constrained_directions	vector		[1, 1, 1]	[x,y,z]...(1 = constrained, 0 = free), can be defined as local or global directions (see Geometry)
Physics.Lagrange.constrained_rotations	vector		[1, 1, 1]	[angle about x axis,angle about y axis,angle about z axis]...(1 = constrained, 0 = free), can be defined as local or global directions (see Geometry)

Position1

Position1.element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!

Position2

Position2.element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-6

Internal.algebraic_variable	algebraic variables of the element. range: 1-6
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi
Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.joint_bryant_angle	prescribe the angles of the joint coordinate system (for actuation, penalty formulation ONLY!)

Example

see file GenericBodyJointShort.txt

```
l = 1 % m

rigidBody
{
  element_type= "Rigid3D"
  Graphics.body_dimensions= [l, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

genericBodyJoint
{
  element_type= "GenericBodyJoint"
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-l/2, 0, 0]  %local position
  }
  Position2.position= [-l/2, 0, 0]  %local position
}
AddConnector(genericBodyJoint)
```

3.3.11 RevoluteJoint

Short description

The RevoluteJoint constrains all relative degrees of freedom between two bodies except the rotation about a local rotation axis. A penalty formulation exists, which replaces the exact lagrange constraint by a approximation with joint stiffness and damping. This constraint can be used together with a RotatorySpringDamperActuator (3.3.18).
The RevoluteJoint is equivalent to a GenericBodyJoint (3.3.10) with all directions and rotations constrained except the rotation about the local x axis. The joint local frame is chosen such that the local x axis is the rotation axis. Please read also the documentation of GenericBodyJoint for details and limitations.
The constraint forces and moments are applied as follows:
Connecting element to element:
The constraint forces and moments are applied on both elements at the position of the connection point of the second element.
Connecting element to ground:
The constraint forces are applied on the position of the connection point of the element.

Limitations

"In penalty formulation the constraints damps the relative velocity of the two connection points in global coordinates, hence if the penalty stiffness is low and the forces high, then a damping of the rotation is possible.

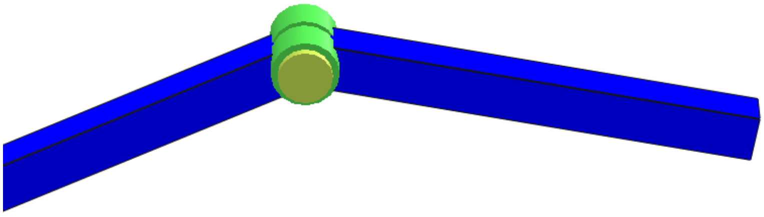


Figure 3.30: RevoluteJoint

Data objects of RevoluteJoint:

Data name	type	R	default	description
element_type	string		"RevoluteJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"RevoluteJoint"	name of the element
element_number	integer	R	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.

Graphics.draw_size	double		-1	cone size for standard joint drawing
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.standard_joint_drawing	bool		1	flag for drawing mode; 1 == draw constraint element; 0 == show constrained directions and rotations;
Graphics.diameter	double		-1	diameter of the revolute joint (for drawing)
Graphics.axis_length	double		-1	axis length of the revolute joint (for drawing)

Physics

Physics.use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)

Physics.Penalty

Physics.Penalty.damping	double		100	damping parameter used for translation and rotation
Physics.Penalty.stiffness	double		1e+006	stiffness parameter used for translation and rotation

Physics.Lagrange

Physics.Lagrange.constrained_directions	vector	**R**	[1, 1, 1]	constrained directions cannot be changed
Physics.Lagrange.constrained_rotations	vector	**R**	[0, 1, 1]	constrained rotations cannot be changed
Physics.rotation_axis	vector		[1, 0, 0]	local rotation axis w.r.t body 1 coordinate system

Position1

Position1.element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!

Position2

Position2.element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-5
Internal.algebraic_variable	algebraic variables of the element. range: 1-5
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi
Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.

Example

see file RevoluteJointShort.txt

```
l = 1 % m
g = 9.81 % m/s^2

gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = g
}
nLoad = AddLoad(gravLoad)

rigidBody
{
  element_type= "Rigid3D"
  loads= [nLoad]
  Graphics.body_dimensions= [l, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

revoluteJoint
{
  element_type= "RevoluteJoint"
  Physics.rotation_axis= [0, 1, 0]  %local rotation axis
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-l/2, 0, 0]  %local position
  }
}
AddConnector(revoluteJoint)
```

3.3.12 PrismaticJoint

Short description

The PrismaticJoint constrains all relative degrees of freedom between two bodies except the translation along a local sliding axis. A penalty formulation exists, which replaces the exact Lagrange constraint by a approximation with joint stiffness and damping.
The PrismaticJoint is equivalent to a GenericBodyJoint (3.3.10) with all directions and rotations constrained except the translation about the local x axis. The joint local frame is chosen such that the local x axis is the sliding axis. Please read also the documentation of GenericBodyJoint for details and limitations.
If the first body is a flexible body, then you might consider using the SlidingPrismaticJoint (3.3.6).
The constraint forces and moments are applied as follows:
Connecting element to element:

The constraint forces and moments are applied on both elements at the position of the connection point of the second element.

Connecting element to ground:

The constraint forces are applied on the position of the connection point of the element.

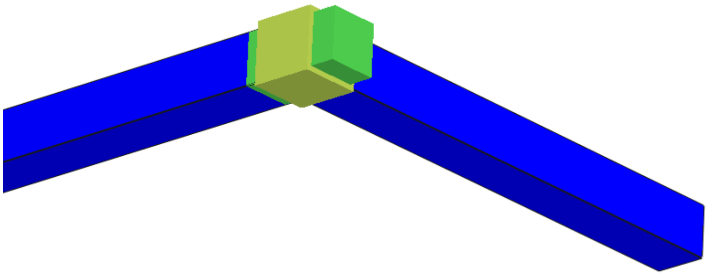


Figure 3.31: PrismaticJoint

Data objects of PrismaticJoint:

Data name	type	R	default	description
element_type	string		"PrismaticJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"PrismaticJoint"	name of the element
element_number	integer	R	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Graphics.draw_size	double		-1	cone size for standard joint drawing
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.standard_joint_drawing	bool		1	flag for drawing mode; 1 == draw constraint nicely; 0 == show constrained directions and rotations;
Graphics.rail_length	double		-1	length of the prismatic joint (for drawing)
Graphics.joint_cube_size	vector		[-1, -1, -1]	cube dimension of prismatic joint (for drawing); [lx (in sl. dir.),ly (normal to sl. dir.),lz (normal to sl. dir.)]
Physics				
Physics.use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.damping	double		100	damping parameter used for translation and rotation
Physics.Penalty.stiffness	double		1e+006	stiffness parameter used for translation and rotation
Physics.Lagrange				

Physics.Lagrange. constrained_directions	vector	**R**	[0, 1, 1]	constrained directions cannot be changed
Physics.Lagrange. constrained_rotations	vector	**R**	[1, 1, 1]	constrained rotations cannot be changed
Physics.sliding_direction	vector		[1, 0, 0]	local sliding direction w.r.t body 1 coordinate system
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-5
Internal.algebraic_variable	algebraic variables of the element. range: 1-5
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi
Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.

Example

see file PrismaticJointShort.txt

```
l = 1 % m

force
{
  load_type = "ForceVector3D"
  force_vector = [10,10,10]
}
nForce = AddLoad(force)

rigidBody
```

```
{
  element_type= "Rigid3D"
  loads= [nForce]
  Graphics.body_dimensions= [1, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

prismaticJoint
{
  element_type= "PrismaticJoint"
  Physics.sliding_direction = [1,0,0]
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-1/2, 0, 0]  %local position
  }
  Position2.position= [-1/2, 0, 0]
}
AddConnector(prismaticJoint)
```

3.3.13 UniversalJoint

Short description

The UniversalJoint constains the local position of two elements and keeps two axes, one on each body, perpendicular to each other.

Degrees of freedom

The vector of the DOF contains the Lagrangian parameters $\lambda = [\lambda_1\ \lambda_2\ \lambda_3\ \lambda_4]^{\mathbf{T}}$, where $\lambda_1, \lambda_2, \lambda_3$ are measures for the violation of the displacement condition and λ_4 is a measure for the violation of the orthogonality condition of the two axes.

Geometry

For this constraint one needs to specify the axes of the cross and the directions in which the hinges are drawn. The direction of the hinge and the axis connected to this hinge have to be given in the local coordinate system of the respective body. See figure 3.33

Equations

The positions and axes are given in local coordinates of body 1 respectively body 2. However the calculations are done internally in global coordinates.
Let

$$\mathbf{x}^i = \left[\begin{array}{ccc} x_1^i & x_2^i & x_3^i \end{array}\right]^{\mathbf{T}}$$

be the position (in global coordinates) where the joint is connected to the first body and let

$$\mathbf{x}^j = \left[\begin{array}{ccc} x_1^j & x_2^j & x_3^j \end{array}\right]^{\mathbf{T}}$$

be the position (in global coordinates) where the joint is connected to the second body.
Let

$$\mathbf{a}^i = \left[\begin{array}{ccc} a_1^i & a_2^i & a_3^i \end{array}\right]^{\mathbf{T}}$$

be the axis (in global coordinates) connected to the first body and let

$$\mathbf{a}^j = \left[\begin{array}{ccc} a^j_1 & a^j_2 & a^j_3 \end{array}\right]^{\mathbf{T}}$$

be the axis (in global coordinates) connected to the second body. Then the constraint equations at position level are

$$\mathbf{C} = \left[\begin{array}{c} \mathbf{x}^i - \mathbf{x}^j \\ {\mathbf{a}^i}^{\mathbf{T}} \cdot \mathbf{a}^j \end{array}\right] = \mathbf{0}.$$

The first three constraints restrict the position of the connection points of body 1 and 2. The fourth equation ensures that the two axes of the cross are perpendicular to each other.
The constraint equations at velocity level are

$$\mathbf{C} = \left[\begin{array}{c} \frac{\partial \mathbf{x}^i}{\partial t} - \frac{\partial \mathbf{x}^j}{\partial t} \\ \frac{\mathbf{a}^i}{\partial t}^{\mathbf{T}} \cdot \frac{\mathbf{a}^j}{\partial t} \end{array}\right] = 0.$$

Limitations

No penalty formulation is available.

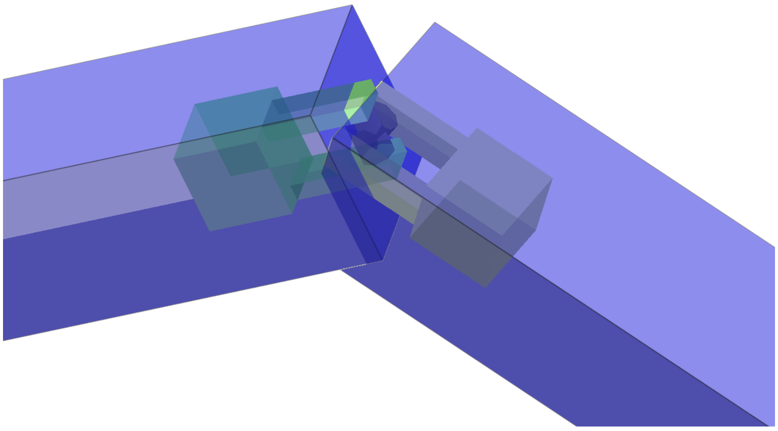


Figure 3.32: UniversalJoint

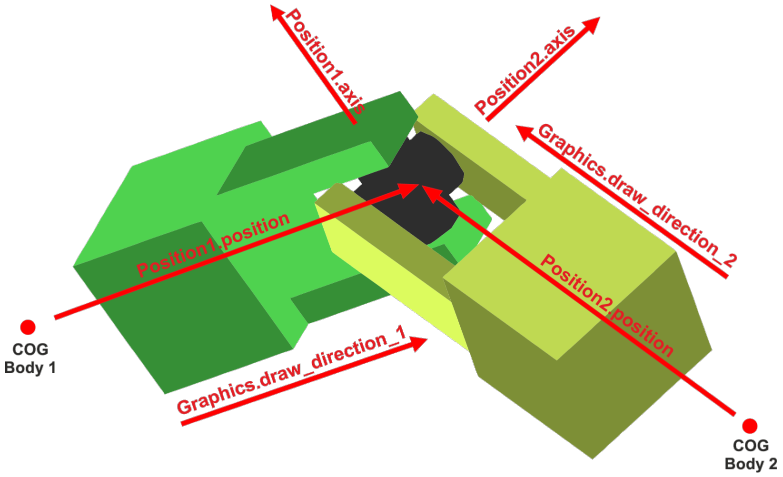


Figure 3.33: UniversalJoint

Data objects of UniversalJoint:

Data name	type	R	default	description
element_type	string		"UniversalJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"UniversalJoint"	name of the element
element_number	integer	R	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of the hinge connected to the first body, range = 0..1
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] color of the hinge connected to the first body, range = 0..1
Graphics.color_cross	vector		[0.2, 0.2, 0.2]	[red, green, blue] color of the cross shaft
Graphics.draw_length	double		-1	length of the universal joint (for drawing)
Graphics.draw_width	double		-1	width of the universal joint (for drawing)
Graphics. draw_direction_1	vector		[1, 0, 0]	direction from body 1 to joint (for drawing)
Graphics. draw_direction_2	vector		[-1, 0, 0]	direction from body 2 to joint (for drawing)
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position
Position1.axis	vector		[0, 1, 0]	the axis of the cross connected to body 1 in local coordinates
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position
Position2.axis	vector		[0, 0, 1]	the axis of the cross connected to body 2 in local coordinates

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-4
Internal.algebraic_variable	algebraic variables of the element. range: 1-4

Example

see file UniversalJoint.txt

```
rotor
{
  element_type= "Rigid3D"
  name= "rotor1"
  Graphics
  {
    body_dimensions= [1, 0.1, 0.1]
  }
  Physics
  {
    moment_of_inertia= [sqr(0.05)*0.5, 0, 0
        0, 1/12*(3*sqr(0.05)+1), 0
        0, 0, 1/12*(3*sqr(0.05)+1)]
    volume= sqr(0.05)*Pi
    mass= 1
  }
  Initialization
  {
    initial_rotation = [0, 0, 0]
    initial_position = [0, 0, 0]
    initial_angular_velocity= [0, 0, 0] %Angular velocity vector in global coordinates: [ang_X,
  }
}
nRotor1 = AddElement(rotor)

rotor.name = "rotor2"
rotor.Initialization.initial_rotation = [0, 0, pi/4]
rotor.Initialization.initial_position = [0.5+0.5*sqrt(0.5), 0.5*sqrt(0.5), 0]
nRotor2 = AddElement(rotor)

universalJoint
{
  element_type= "UniversalJoint"
  name= "UniversalJoint"
  Graphics
  {
    show_connector= 1
    color_body1= [0.3, 0.8, 0.3]
    color_body2= [0.7, 0.8, 0.3]
    color_cross= [0.2, 0.2, 0.2]
    draw_length= -1
    draw_width= -1
    draw_direction_1= [1, 0, 0]
    draw_direction_2= [-1, 0, 0]
  }
  Position1
  {
    element_number= nRotor1
    position= [0.5, 0, 0]
    axis= [0, 1, 0]
  }
  Position2
  {
    element_number= nRotor2
    position= [-0.5, 0, 0]
    axis= [0, 0, 1]
  }
}
AddConnector(universalJoint)
```

3.3.14 RigidJoint

Short description

The RigidJoint constrains the position and relative angles of an element at a specified local position. If only one element is specified, a ground joint is realized. A penalty formulation exists, which replaces the exact lagrange constraint by an approximation with joint stiffness and damping. The RigidJoint is equivalent to a GenericBodyJoint (3.3.10) with all directions and rotations constrained Please read also the documentation of GenericBodyJoint for details and limitations.

The constraint forces and moments are applied as follows:

Connecting element to element:

The constraint forces and moments are applied on both elements at the position of the connection point of the second element.

Connecting element to ground:

The constraint forces are applied on the position of the connection point of the element.

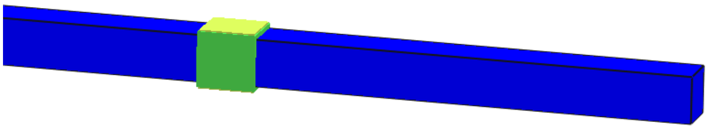


Figure 3.34: RigidJoint

Data objects of RigidJoint:

Data name	type	R	default	description
element_type	string		"RigidJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"RigidJoint"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics. draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Graphics.draw_size	double		-1	cone size for standard joint drawing
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics. standard_joint_drawing	bool		1	flag for drawing mode; 1 == draw constraint element; 0 == show constrained directions and rotations;
Graphics.cube_length	double		-1	rigid joint dimension (for drawing)
Physics				
Physics. use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.damping	double		100	damping parameter used for translation and rotation
Physics.Penalty.stiffness	double		1e+006	
Physics.Lagrange				
Physics.Lagrange. constrained_directions	vector	**R**	[1, 1, 1]	constrained directions cannot be changed
Physics.Lagrange. constrained_rotations	vector	**R**	[1, 1, 1]	constrained rotations cannot be changed
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-6
Internal.algebraic_variable	algebraic variables of the element. range: 1-6
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi
Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.

Example

see file RigidJointShort.txt

```
l = 1 % m
g = 9.81 % m/s^2

gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = g
}
nLoad = AddLoad(gravLoad)

rigidBody
{
  element_type= "Rigid3D"
  loads= [nLoad]
  Graphics.body_dimensions= [l, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

rigidJoint
{
  element_type= "RigidJoint"
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-l/2, 0, 0]  %local position
  }
  Position2.position= [-l/2, 0, 0]
}
AddConnector(rigidJoint)
```

3.3.15 CylindricalJoint

Short description

The CylindricalJoint constrains like the RevoluteJoint, but allows additionally translation along the rotational axis. A penalty formulation exists, which replaces the exact lagrange constraint by a approximation with joint stiffness and damping.
The CylindricalJoint is equivalent to a GenericBodyJoint (3.3.10) with all directions and rotations constrained except the translation and rotation about the local x axis. The joint local frame is chosen such that the local x axis is the rotation and sliding axis. Please read also the documentation of GenericBodyJoint for details and limitations.
The constraint forces and moments are applied as follows:
Connecting element to element:
The constraint forces and moments are applied on both elements at the position of the connection point of the second element.
Connecting element to ground:
The constraint forces are applied on the position of the connection point of the element.

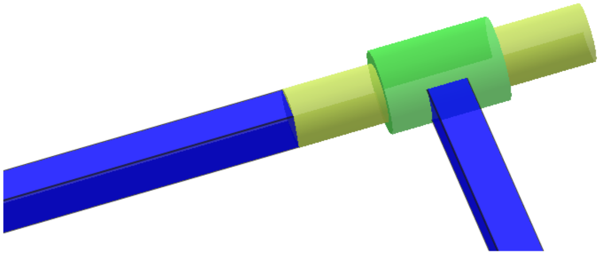


Figure 3.35: CylindricalJoint

Data objects of CylindricalJoint:

Data name	type	R	default	description
element_type	string		"CylindricalJoint"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"CylindricalJoint"	name of the element
element_number	integer	R	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size_joint_local_frame	double		0	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Graphics.draw_size	double		-1	cone size for standard joint drawing

Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics. standard_joint_drawing	bool		1	flag for drawing mode; 1 == draw constraint element; 0 == show constrained directions and rotations;
Graphics. joint_cylinder_size	vector		[-1, -1]	cylinder dimension of cylindrical joint (for drawing); [lx (cyl. length, in sl. dir.),d (cylinder diameter)]
Graphics.axis_length	double		-1	axis length of the revolute joint (for drawing)
Physics				
Physics. use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.damping	double		100	damping parameter used for translation and rotation
Physics.Penalty.stiffness	double		1e+006	stiffness parameter used for translation and rotation
Physics.Lagrange				
Physics.Lagrange. constrained_directions	vector	**R**	[0, 1, 1]	constrained directions cannot be changed
Physics.Lagrange. constrained_rotations	vector	**R**	[0, 1, 1]	constrained rotations cannot be changed
Physics. rotation_sliding_axis	vector		[1, 0, 0]	local rotation/sliding axis w.r.t body 1 coordinate system
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-4
Internal.algebraic_variable	algebraic variables of the element. range: 1-4
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi

Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.

Example

see file CylindricalJointShort.txt

```
l = 1 % m

force
{
  load_type = "ForceVector3D"
  force_vector = [10,10,10]
}
nForce = AddLoad(force)

rigidBody
{
  element_type= "Rigid3D"
  loads= [nForce]
  Graphics.body_dimensions= [l, 0.05, 0.05]
}
nRigid = AddElement(rigidBody)

cylindricalJoint
{
  element_type= "CylindricalJoint"
  Physics.rotation_sliding_axis = [1,0,0]
  Position1
  {
    element_number= nRigid  %number of constrained element
    position= [-l/2, 0, 0]  %local position
  }
  Position2.position= [-l/2, 0, 0]
}
AddConnector(cylindricalJoint)
```

3.3.16 SpringDamperActuator

Short description

The Spring-Damper-Actuator connects two points with a spring, a damper and a actor element, in which actuator force fa remains constant. The resultant force is applied in the connection line of these points. There are different modes available, how the spring and damper force is calculated. It is also possible to change the neutral spring length. This joint is realized in Penalty formulation only.

Equations

point positions: $\mathbf{p}^{(1)} = \left[p_x^{(1)} p_y^{(1)} p_z^{(1)}\right]^T; \qquad \mathbf{p}^{(2)} = \left[p_x^{(2)} p_y^{(2)} p_z^{(2)}\right]^T.$

point velocities: $\dot{\mathbf{p}}^{(1)} = \left[\dot{p}_x^{(1)} \dot{p}_y^{(1)} \dot{p}_z^{(1)}\right]^T$; $\qquad \dot{\mathbf{p}}^{(2)} = \left[\dot{p}_x^{(2)} \dot{p}_y^{(2)} \dot{p}_z^{(2)}\right]^T$.

spring length: l_0

direction vector: $\mathbf{dir} = \frac{\mathbf{p}^{(1)}-\mathbf{p}^{(2)}}{\sqrt{\left(p_x^{(1)}-p_x^{(2)}\right)^2+\left(p_y^{(1)}-p_y^{(2)}\right)^2+\left(p_z^{(1)}-p_z^{(2)}\right)^2}}$

spring elongation: $\Delta x = l - l_0 = \left(\mathbf{p}^{(1)} - \mathbf{p}^{(2)}\right)^T \mathbf{dir} - l_0$

spring velocity: $v = \left(\dot{\mathbf{p}}^{(1)} - \dot{\mathbf{p}}^{(2)}\right)^T \mathbf{dir}$

resultant force (see section forcemode):
forcemode 0: $f = k\,\Delta x + d\,v + f_a$ (a)
forcemode 1: $f = k\left(\Delta x\right)\Delta x + d\left(v\right)v + f_a$ (b)
forcemode 2: $f = f_k + f_d + f_a$ (c)
forcemode 3: $f = f_k\left(\Delta x\right) + f_d\left(v\right) + f_a$ (d)

Limitations

If the 2 end points of the spring are the same point in the initial configuration, this may lead to problems! The direction of the spring can not be determined in that case!

Description of the different modi

element to ground	Position2.element_number AND Position2.node_number have to be equal to 0
element to element	Position2.element_number and/or Position2.node_number must not be equal to 0
forcemode	**Physics.forcemode = 0:** Force is computed as (a) with constant stiffness and damping factors k and d. The factors can be defined in the two fields in Physics.Linear. **Physics.forcemode = 1:** 2 MathFunctions are used to describe piecewise linear stiffness $k\left(\Delta x\right)$ and damping $d\left(v\right)$, see formula (b) and Physics.MathFunction. **Physics.forcemode = 2:** 2 IOElementDataModifiers describe the force (c) due to stiffness and damping. You should use this mode if full nonlinear behavior is required, e.g. $f_k = f_k\left(t, l, v, ...\right)$ and $f_d = d\left(t, l, v, ...\right)$. **Physics.forcemode = 3:** 2 MathFunctions are used to describe piecewise linear spring force $f_k\left(\Delta x\right)$ and damping force $f_d\left(v\right)$, see formula (d) and Physics.MathFunction. modifier value names for forcemode == 2: f_k: Connector.SpringDamperActuator.spring_force' f_d: Connector.SpringDamperActuator.damper_force'

spring length offset	It is possible to change the spring length l_0 (neutral length of the spring) during the simulation, e.g. for the usage of the SpringDamperActuator as a linear actuator. In standard mode the value in the field Physics.spring_length remains constant. This value can be modified by a IOElementDataModifier via 'Connector.SpringDamperActuator.spring_length_offset'.
additional actor force	In Physics.actor_force a constant offset force f_a can be added.

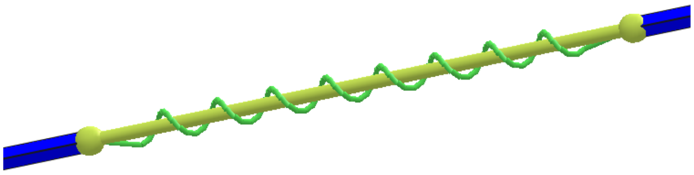


Figure 3.36: SpringDamperActuator

Data objects of SpringDamperActuator:

Data name	type	R	default	description
element_type	string		"SpringDamperActuator"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"SpringDamperActuator"	name of the element
element_number	integer	R	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint (spring), range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint (damper), range = 0..1, use default color:[-1,-1,-1]
Graphics. spring_diameter	double		-1	spring diameter used for drawing only.
Graphics.spring_coils	double		10	spring coils used for drawing. If set to 0, then a cylinder with the value 'spring_diameter' as diameter is shown instead of the coils.
Graphics. damper_diameter	double		-1	damper diameter used for drawing only. If set to 0, then the damper is not shown. It's recommended to choose the value smaller then the spring diameter.
Physics				
Physics.spring_length	double		0	length of the spring in the initial configuration
Physics.actor_force	double		0	constant force acting on the spring

Physics.forcemode	integer		0	defines how the spring and damper force is computed: 0..constant coefficient, 1..MathFunction (stiffness and damping), 2..spring and damper force prescribed by IOElementDataModifier, 3..MathFunction (spring force and damping force)
Physics.Linear				
Physics.Linear. spring_stiffness	double		100	stiffness coefficient of the linear spring. Only used if forcemode is 0.
Physics.Linear.damping	double		1	damping coefficient for viscous damping. Only used if forcemode is 0.
Physics.MathFunction				
Physics.MathFunction.MathFunction_k				
Physics.MathFunction. MathFunction_k. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_k. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_k. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_k. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_k. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
Physics.MathFunction. MathFunction_k. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
Physics.MathFunction.MathFunction_d				
Physics.MathFunction. MathFunction_d. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_d. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_d. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_d. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_d. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
Physics.MathFunction. MathFunction_d. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
Physics.MathFunction.MathFunction_fk				
Physics.MathFunction. MathFunction_fk. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_fk. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation

Physics.MathFunction. MathFunction_fk. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_fk. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_fk. parsed_function	string		""	string representing parsed function, e.g. ’A*sin(omega*t)’
Physics.MathFunction. MathFunction_fk. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. ’t’

Physics.MathFunction.MathFunction_fd

Physics.MathFunction. MathFunction_fd. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_fd. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_fd. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_fd. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_fd. parsed_function	string		""	string representing parsed function, e.g. ’A*sin(omega*t)’
Physics.MathFunction. MathFunction_fd. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. ’t’

Position1

Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position1.node_number	integer		0	local or global (if element_number == 0) node number.

Position2

Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!
Position2.node_number	integer		0	local or global (if element_number == 0) node number.

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-14
Internal.second_order_variable	second order variables of the element. range: 1-7
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-7

Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.SpringDamperActuator.force	internal resultant force of connector
Connector.SpringDamperActuator.spring_length	actual spring length
Connector.SpringDamperActuator.spring_elongation	elongation of spring
Connector.SpringDamperActuator.spring_velocity	spring velocity

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.SpringDamperActuator.spring_length_offset	prescribe the neutral spring length
Connector.SpringDamperActuator.spring_force	prescribe the stiffness force
Connector.SpringDamperActuator.damper_force	prescribe the damping force

Example

see file SpringDamperActuator.txt

```
l = 0.5 % m
m = 10 % kg
g = 9.81 % m/s^2

gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = g
}
nLoad = AddLoad(gravLoad)

mass
{
  element_type = "Mass3D"  %specification of element type.
  loads = [nLoad]
  Initialization.initial_position = [0, 0, l]  %initial position
  Physics.mass = m  %total mass
}
nMass = AddElement(mass)

springDamperActuator
{
  element_type = "SpringDamperActuator"
  Physics.forcemode = 2 % nonlinear spring
  Position1.element_number = nMass  %number of constrained element
  Position2.element_number = 0 %number of constrained element
}
```

```
nSpringDamperActuator = AddConnector(springDamperActuator)

disp
{
  sensor_type = "FVElementSensor"
  element_number = nMass
  field_variable = "displacement"
  component = "z"
}
nDisp = AddSensor(disp)

nonlinearStiffnessForce
{
  element_type = "IOMathFunction"
  Graphics
  {
    position = [0, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
  IOBlock
  {
    input_element_numbers = [nDisp]  %element connected to input
    input_element_types = [2]  %2=Sensor
    input_local_number = [1]
    MathFunction
    {
      piecewise_mode = 1  %modus for piecewise interpolation: 1=linear
      piecewise_points = [-0.3,-0.2,-0.15,0,0.15,0.2,0.3]  %m, supporting points
      piecewise_values = [-5000,-300,-30,0,30,300,5000]  %N, values at s. p.
    }
  }
}
nNonlinearStiffnessForce = AddElement(nonlinearStiffnessForce)

modifier_SDA
{
  element_type = "IOElementDataModifier"
  Graphics
  {
    position = [30, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
  IOBlock
  {
    input_element_numbers = [nNonlinearStiffnessForce]  %element connected to input
    input_element_types = [1]
    input_local_number = [1]
    mod_variable_name = "Connector.SpringDamperActuator.spring_force"  %modified element data
    mod_element_number = nSpringDamperActuator  %modified constraint
  }
}
AddElement(modifier_SDA)
```

3.3.17 RigidLink

Short description

A rigid link is a rigid constraint element that provides a stiff connection between nodes or positions in the model. In standard mode the distance between the connected points remains constant. In extended mode it is possible to change the distance as a function of time or input. There is only a Lagrange formulation implemented.

Equations

point positions: $\mathbf{p}^{(1)} = \left[p_x^{(1)} p_y^{(1)} p_z^{(1)}\right]^T; \qquad \mathbf{p}^{(2)} = \left[p_x^{(2)} p_y^{(2)} p_z^{(2)}\right]^T.$
point velocities: $\dot{\mathbf{p}}^{(1)} = \left[\dot{p}_x^{(1)} \dot{p}_y^{(1)} \dot{p}_z^{(1)}\right]^T; \qquad \dot{\mathbf{p}}^{(2)} = \left[\dot{p}_x^{(2)} \dot{p}_y^{(2)} \dot{p}_z^{(2)}\right]^T.$
link length: l_0
time derivative of link length: v (equates $\dot{l}_0$)
direction vector: $\mathbf{dir} = \frac{\mathbf{p}^{(1)}-\mathbf{p}^{(2)}}{\sqrt{\left(p_x^{(1)}-p_x^{(2)}\right)^2+\left(p_y^{(1)}-p_y^{(2)}\right)^2+\left(p_z^{(1)}-p_z^{(2)}\right)^2}}$
position constraint: $\mathbf{C} = \left(\mathbf{p}^{(1)} - \mathbf{p}^{(2)}\right)^T \mathbf{dir} - l_0 = 0$ (a)
velocity constraint: $\dot{\mathbf{C}} = \left(\dot{\mathbf{p}}^{(1)} - \dot{\mathbf{p}}^{(2)}\right)^T \mathbf{dir} - v = 0$ (b)
$\frac{\partial \mathbf{C}}{\partial \mathbf{q}}^T = \left(\frac{\partial \mathbf{p}^{(1)}}{\partial \mathbf{q}} - \frac{\partial \mathbf{p}^{(2)}}{\partial \mathbf{q}}\right)^T \mathbf{dir}$

Limitations

For a position constraint (index 3 solver) with variable distance it is necessary to define the link length l_0 as a function of time. In this case the velocity input v (the derivative of the distance with respect to time) is not considered, see formula (a).Reverse conditions apply to the velocity constraint with formula (b).

Description of the different modi

element to ground	Position2.element_number AND Position2.node_number have to be equal to 0
element to element	Position2.element_number and/or Position2.node_number must not be equal to 0
distancemode	**Physics.distancemode = 0:** The distance remains constant. The value can be defined in the field Physics.Constant.link_length. **Physics.distancemode = 1:** A MathFunction is used to describe piecewise linear distance or velocity development over time t, e.g. for a rigid link actuator. See Physics.MathFunction. **Physics.distancemode = 2:** A IOElementDataModifier describes the developing distance or velocity over time t, e.g. for a rigid link actuator. See section limitations.

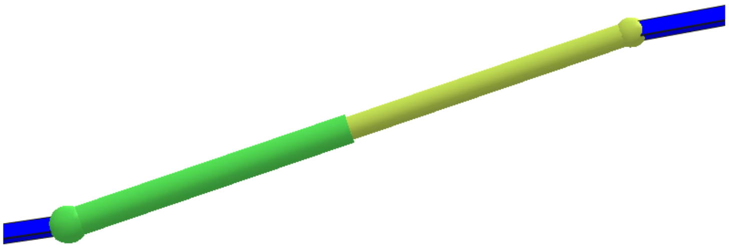


Figure 3.37: RigidLink

Data objects of RigidLink:

Data name	**type**	**R**	**default**	**description**
element_type	string		"RigidLink"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"RigidLink"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics. cylinder1_diameter	double		-1	cylinder one diameter (drawing only).
Graphics. cylinder2_diameter	double		0	cylinder two diameter (drawing only). Only used if distance not constant = distancemode 1 or 2.
Graphics. cylinder1_length	double		0	cylinder one length (drawing only). Only used if distance not constant = distancemode 1 or 2.
Physics				
Physics.distancemode	integer		0	defines the distance: 0..constant distance, 1..MathFunction, 2..IOElementDataModifier
Physics.Constant				
Physics.Constant. link_length	double		0	constant distance is used, when distancemode = 0
Physics.MathFunction				
Physics.MathFunction.MathFunction_1				
Physics.MathFunction. MathFunction_1. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_1. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_1. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_1. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_1. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'

Physics.MathFunction. MathFunction_l. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
Physics.MathFunction.MathFunction_v				
Physics.MathFunction. MathFunction_v. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_v. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_v. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_v. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_v. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
Physics.MathFunction. MathFunction_v. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
Position1				
Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!
Position1.node_number	integer		0	local or global (if element_number == 0) node number.
Position2				
Position2. element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!
Position2.node_number	integer		0	local or global (if element_number == 0) node number.

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.algebraic_variable	algebraic variables of the element. range: 1-1

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.RigidLink.link_length	distance between the connected points (l0)
Connector.RigidLink.link_velocity	derivative of the distance with respect to time (v)

Example

see file RigidLink.txt

```
l = 0.5 % m
m = 10 % kg
v = 0.1 % m/s
g = 9.81 % m/s^2

gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = g
}
nLoad = AddLoad(gravLoad)

mass
{
  element_type = "Mass3D"
  loads = [nLoad]
  Initialization.initial_position = [l, 0, 0]  %initial position
  Physics.mass = m  %total mass of point mass
}
nMass = AddElement(mass)

%link
rigidLink
{
  element_type = "RigidLink"
  Physics.distancemode = 2 % link length by modifier
  Graphics
  {
  show_connector = 1
  cylinder1_diameter = 0.1
  cylinder2_diameter = 0.08
  cylinder1_length = l/2
  }
  Position1.element_number = nMass  %number of constrained element
  Position2.element_number = 0  %number of constrained element
}
nRigidLink = AddConnector(rigidLink)

time
{
  element_type = "IOTime"
  Graphics
  {
    position = [-30, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
}
nTime = AddElement(time)

vel
{
  element_type = "IOMathFunction"
  IOBlock
  {
    input_element_numbers = [nTime]  %element connected to input
    input_element_types = [1]  %1=IOElement
    input_local_number = [1]  %i-th number of output of previous IOelement
    MathFunction
    {
      piecewise_mode = 1  %modus for piecewise interpolation: 1=linear
      piecewise_points = [0,1,2,3]  %supporting points
      piecewise_values = [1,1,2*l,2*l]  %values at supporting points
    }
  }
}
nVel = AddElement(vel)

modifier
{
  element_type = "IOElementDataModifier"
  Graphics
  {
    position = [30, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
  IOBlock
  {
    input_element_numbers = [nVel]  %element connected to input
    input_element_types = [1]  %1=IOElement
    input_local_number = [1]  %i-th number of output connected to this element
    mod_variable_name = "Connector.RigidLink.link_length"  %variable name
    mod_element_number = nRigidLink  %element number
  }
}
AddElement(modifier)
```

3.3.18 RotatorySpringDamperActuator

Short description

The RotatorySpringDamperActuator connects two elements with rotatory spring, damper and a constant actuator moment ma. Positive rotation around rotation axis according to right hand rule. There are different modes available, how the spring and damper moment is calculated. It is also possible to change the neutral spring angle. This joint is realized in Penalty formulation only.

Equations

spring angular deflection $\Delta\phi = \phi - \phi_0$
spring angular velocity ω

resultant moment (see section forcemode):
forcemode 0: $m = k\,\Delta\phi + d\,\omega + m_a$ (a)
forcemode 1: $m = k\,(\Delta\phi)\,\Delta\phi + d\,(\omega)\,\omega + m_a$ (b)
forcemode 2: $m = m_k + m_d + m_a$ (c)

Limitations

The RotatorySpringDamperActuator should be used together with a RevoluteJoint to avoid useless simulation results. It is important to ensure that the relative angle of rotation between the two bodies must never be greater than $\pm\pi$. This has to be taken into accout when using an offset angle ϕ_0.

Description of the different modi

element to ground	Position2.element_number AND Position2.node_number have to be equal to 0
element to element	Position2.element_number and/or Position2.node_number must not be equal to 0
forcemode	**Physics.forcemode = 0:** Moment is computed as (a) with constant stiffness and damping factors k and d. The factors can be defined in the two fields in Physics.Linear. **Physics.forcemode = 1:** A MathFunction is used to describe piecewise linear stiffness $k\,(\Delta\phi)$ and damping $d\,(\omega)$, see formula (b) and Physics.MathFunction. **Physics.forcemode = 2:** 2 IOElementDataModifiers describe the moment (c) due to stiffness and damping. You should use this mode if full nonlinear behavior is required, e.g. $m_k = m_k\,(t, \phi, \omega, ...)$ and $m_d = d\,(t, \phi, \omega, ...)$.
spring angle offset	It is possible to change the spring angle ϕ_0 (neutral angle of the spring) during the simulation, e.g. for the usage of the RotatorySpringDamperActuator as a rotational actuator. In standard mode the offset remains constant. The value can be defined in the field Physics.spring_angle_offset. This offset can be modified by a IOElementDataModifier via 'Connector.RotatorySpringDamperActuator.angle_offset'.
additional actuator moment	In Physics.actuator_torque a constant offset moment m_a can be added.

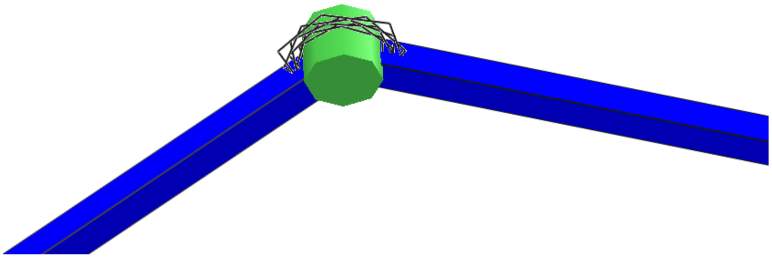


Figure 3.38: RotatorySpringDamperActuator

Data objects of RotatorySpringDamperActuator:

Data name	**type**	**R**	**default**	**description**
element_type	string		"RotatorySpringDamperActuator"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"RotatorySpringDamperActuator"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.spring_size	double		-1	radius of torsional spring. This parameter is used for drawing only.
Graphics.windings	double		10	number of windings of torsional spring. This parameter is used for drawing only.
Graphics.axis_radius	double		-1	radius of torsional spring axis (cylinder). This parameter is used for drawing only.
Physics				
Physics.spring_angle_offset	double		0	spring angle offset is used if constant_spring_angle_offset is enabled. A positive offset equates a positve angle about the rotation axis.
Physics.actuator_torque	double		0	constant torque of an actuator. A positive torque is acting about the rotation axis in a positive sense.
Physics.rotation_axis	vector		[0, 0, 0]	local axis of rotation w.r.t. body 1 coordinate system in inital configuration
Physics.forcemode	integer		0	defines how the spring and damper moment is computed: 0..constant coefficient, 1..MathFunction, 2..IOElementDataModifier
Physics.Linear				
Physics.Linear.spring_stiffness	double		100	stiffness parameter of the rotatory spring. Only used if forcemode is 0.
Physics.Linear.damping	double		1	damping coefficient for viscous damping. Only used if forcemode is 0.
Physics.MathFunction				
Physics.MathFunction.MathFunction_k				
Physics.MathFunction.MathFunction_k.piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic

Physics.MathFunction.MathFunction_k.piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction.MathFunction_k.piecewise_values	vector		[]	values at supporting points
Physics.MathFunction.MathFunction_k.piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction.MathFunction_k.parsed_function	string		""	string representing parsed function, e.g. ’A*sin(omega*t)’
Physics.MathFunction.MathFunction_k.parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. ’t’

Physics.MathFunction.MathFunction_d

Physics.MathFunction.MathFunction_d.piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction.MathFunction_d.piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction.MathFunction_d.piecewise_values	vector		[]	values at supporting points
Physics.MathFunction.MathFunction_d.piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction.MathFunction_d.parsed_function	string		""	string representing parsed function, e.g. ’A*sin(omega*t)’
Physics.MathFunction.MathFunction_d.parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. ’t’

Position1

Position1.element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0, 0]	local position. Only used if node_number == 0!

Position2

Position2.element_number	integer		0	Number of constrained element
Position2.position	vector		[0, 0, 0]	local or global (if element_number == 0) position. Only used if node_number == 0!

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-14
Internal.second_order_variable	second order variables of the element. range: 1-7
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-7

Connector.force	force applied to the kinematic pairs due to the connector. range: 1-3, corresponds to force in global x-y-z direction
Connector.moment	internal global moment of connector
Connector.force_local	internal local force of connector (joint coordinate system JAi)
Connector.moment_local	internal local moment of connector (joint coordinate system JAi)
Connector.displacement	displacement between the joint coordinate systems JAi and JAj expressed in coordinate system JAi
Connector.angle	bryant angles between the joint coordinate systems JAi and JAj. All constrained components are zero.
Connector.RotatorySpringDamperActuator.moment	internal moment of connector

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.RotatorySpringDamperActuator.angle_offset	prescribe the angle offset
Connector.RotatorySpringDamperActuator.spring_moment	prescribe the stiffness moment
Connector.RotatorySpringDamperActuator.damper_moment	prescribe the damping moment

Example

see file RotationalSpringDamperActuator.txt

```
l = 0.5 % m
r = 0.05
m = 10 % kg
Ix = m*r*r/2 % kg*m^2
Iy = m*l*l/3 % kg*m^2
Iz = Iy

force
{
  load_type = "ForceVector3D"
  force_vector = [0,-100,0]
  position = [l/2,0,0]
  local_force = 1
}
nForce = AddLoad(force)

body
{
  element_type = "Rigid3D"
  loads = [nForce]
  Physics
  {
   mass = m
   moment_of_inertia = [Ix,0.,0.;0.,Iy,0.;0.,0.,Iz]
  }
  Graphics
  {
    RGB_color= [0., 0., 1.]  %[red, green, blue]
    show_element = 1  %Flag to draw element
    body_dimensions = [l,r,r]
  }
  Initialization.initial_position = [l/2,0,0]
}
nBody = AddElement(body)

revoluteJoint
{
  element_type = "RevoluteJoint"
  Physics.rotation_axis = [0,0,1]
  Graphics.show_connector = 0

  Position1.element_number = nBody  %number of constrained element
  Position1.position = [-l/2,0,0]  %local position
  Position2.element_number = 0  %number of constrained element
}
nRevoluteJoint = AddConnector(revoluteJoint)

rotSpringDamperActuator
{
  element_type = "RotatorySpringDamperActuator"
  Physics
  {
  forcemode = 2 % force by nonlinear spring
  rotation_axis = [0,0,1]
  }
  Graphics
  {
  show_connector = 1
  }
  Position1.element_number = nBody  %number of constrained element
  Position1.position = [-l/2,0,0]  %local position
  Position2.element_number = 0  %number of constrained element
}
nRotSpringDamperActuator = AddConnector(rotSpringDamperActuator)

phi
{
  sensor_type = "ElementSensor"
  element_number = nRevoluteJoint
  value = "Connector.angle[1]" %about x-axis (joint coordinate system)
}
nPhi = AddSensor(phi)

nonlinearStiffnessMoment
```

```
{
  element_type = "IOMathFunction"
  Graphics
  {
    position = [0, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
  IOBlock
  {
    input_element_numbers = [nPhi]  %element connected to input
    input_element_types = [2]  %2=Sensor
    input_local_number = [1]  %i-th number of output
    MathFunction
    {
      piecewise_mode = 1  %1=linear
      piecewise_points = [-1.2,-0.8,-0.5,0,0.5,0.8,1.2]  %m, supporting points
      piecewise_values = [200,100,10,0,-10,-100,-200]  %N, values at s. p.
    }
  }
}
nNonlinearStiffnessMoment = AddElement(nonlinearStiffnessMoment)

modifier
{
  element_type = "IOElementDataModifier"
  Graphics
  {
    position = [30, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
  IOBlock
  {
    input_element_numbers = [nNonlinearStiffnessMoment]  %element connected to input
    input_element_types = [1]  %1=IOElement
    input_local_number = [1]  %i-th number of output
    mod_variable_name = "Connector.RotatorySpringDamperActuator.spring_moment"  %variable name
    mod_element_number = nRotSpringDamperActuator  %element number
  }
}
AddElement(modifier)
```

3.3.19 SpringDamperActuator2D

Short description

The SpringDamperActuator2D is a simplified version of the SpringDamperActuator for 2D elements. Nodes are not supported in the 2D version. Apart from that the constraint has the same functionality as the 3D version. See the SpringDamperActuator documentation for more information.

Description of the different modi

element to ground	Position2.element_number has to be equal to 0
element to element	Position2.element_number must not be equal to 0
Lagrange	If Physics.use_penalty_formulation = 0, then no stiffness and no damping parameters are used.

Data objects of SpringDamperActuator2D:

Data name	**type**	**R**	**default**	**description**
element_type	string		"SpringDamperActuator2D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"SpringDamperActuator2D"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.color_body1	vector		[0.3, 0.8, 0.3]	[red, green, blue] first color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics.color_body2	vector		[0.7, 0.8, 0.3]	[red, green, blue] second color of constraint, range = 0..1, use default color:[-1,-1,-1]
Graphics. spring_diameter	double		-1	spring diameter used for drawing only.
Graphics.spring_coils	double		10	spring coils used for drawing. If set to 0, then a cylinder with the value 'spring_diameter' as diameter is shown instead of the coils.
Graphics. damper_diameter	double		-1	damper diameter used for drawing only. If set to 0, then the damper is not shown. It's recommended to choose the value smaller then the spring diameter.
Physics				
Physics.spring_length	double		0	length of the spring in the initial configuration
Physics.actor_force	double		0	constant force acting on the spring
Physics.forcemode	integer		0	defines how the spring and damper force is computed: 0..constant coefficient, 1..MathFunction, 2..IOElementDataModifier
Physics.Linear				
Physics.Linear. spring_stiffness	double		100	stiffness coefficient of the linear spring. Only used if forcemode is 0.
Physics.Linear.damping	double		1	damping coefficient for viscous damping. Only used if forcemode is 0.
Physics.MathFunction				
Physics.MathFunction.MathFunction_k				
Physics.MathFunction. MathFunction_k. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_k. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_k. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_k. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation

Physics.MathFunction. MathFunction_k. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
Physics.MathFunction. MathFunction_k. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'

Physics.MathFunction.MathFunction_d

Physics.MathFunction. MathFunction_d. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
Physics.MathFunction. MathFunction_d. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
Physics.MathFunction. MathFunction_d. piecewise_values	vector		[]	values at supporting points
Physics.MathFunction. MathFunction_d. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
Physics.MathFunction. MathFunction_d. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
Physics.MathFunction. MathFunction_d. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'

Position1

Position1. element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0]	local position 1

Position2

Position2. element_number	integer		0	Number of constrained element (0 if ground joint)
Position2.position	vector		[0, 0]	local or global position 2

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-14
Internal.second_order_variable	second order variables of the element. range: 1-7
Internal.second_order_variable_velocity	velocities of second order variables of the element. range: 1-7
Connector.SpringDamperActuator.force	resultant force of connector
Connector.force	force applied to the kinematic pairs due to the connector. range: 1-2, corresponds to force in global x-y direction
Connector.SpringDamperActuator.spring_length	actual spring length
Connector.SpringDamperActuator.spring_elongation	elongation of spring
Connector.SpringDamperActuator.spring_velocity	magnitude of spring velocity

Controllable special values:

For more information see section 3.1

value name	**description**
Connector.SpringDamperActuator.spring_length_offset	prescribe the neutral spring length
Connector.SpringDamperActuator.spring_force	prescribe the stiffness force
Connector.SpringDamperActuator.damper_force	prescribe the damping force

Example

see file SpringDamperActuator2D.txt

```
mass
{
  element_type= "Mass2D"  %specification of element type.
  Initialization.initial_position= [1, 0.5]  %initial position [x,y]
  Physics.mass= 1  %total mass of point mass
}
nMass = AddElement(mass)

sda
{
  element_type= "SpringDamperActuator2D"  %specification of element type.
  Position1.element_number= nMass  %Number of constrained element
}
nSDA = AddConnector(sda)
```

3.3.20 PointJoint2D

Short description

The PointJoint2D is a simplified version of the PointJoint for 2D elements. It constrains two elements at at local element positions. If only one element is specified (second element 0), a ground PointJoint is realized. It provides both Lagrangian and penalty formulation.

Description of the different modi

element to ground	Position2.element_number has to be equal to 0
element to element	Position2.element_number must not be equal to 0
Lagrange	If Physics.use_penalty_formulation = 0, then no stiffness and no damping parameters are used.

Data objects of PointJoint2D:

Data name	**type**	**R**	**default**	**description**

element_type	string		"PointJoint2D"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"PointJoint2D"	name of the element
element_number	integer	**R**	2	number of the element in the mbs
Graphics				
Graphics.RGB_color	vector		[0.3, 0.8, 0.3]	[red, green, blue] color of element, range = 0..1, use default color:[-1,-1,-1]
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.draw_size_joint_local_frame	double		-1	drawing dimensions of joint local frame. If set to -1, then global_draw_scalar_size is used. If set to 0, then no joint local frame is drawn.
Graphics.draw_size	double		-1	drawing dimensions of constraint. If set to -1, then global_draw_scalar_size is used.
Geometry				
Geometry.use_joint_local_frame	bool		0	Use a special joint local frame
Geometry.joint_local_frame	double		0	Prerotate stiffness vector w.r.t. global coordinate system or local coordinate system of body 1 with angle phi_z about the z axis. Just used if use_joint_local_frame == 1
Geometry.use_local_coordinate_system	bool		0	0=use global coordinates, 1=use local coordinate system of Body 1
Physics				
Physics.use_penalty_formulation	bool		0	0 = use lagrange multipliers (index 3 DAE, exact), 1 = use penalty formulation (no additional equation added, approximate constraint)
Physics.Penalty				
Physics.Penalty.spring_stiffness	double		1e+008	general or penalty stiffness parameter
Physics.Penalty.spring_stiffness_vector	vector		[0, 0]	penalty stiffness parameter [kx,ky]. Just used if scalar spring_stiffness == 0, otherwise kx=ky=spring_stiffness
Physics.Penalty.damping	double		1	damping coefficient for viscous damping (F = d*v), applied in all constrained directions
Physics.Lagrange				
Physics.Lagrange.constrained_directions	vector		[1, 1]	[x,y]...(1 = constrained, 0 = free), can be defined as local or global directions (see Geometry)
Position1				
Position1.element_number	integer		1	Number of constrained element
Position1.position	vector		[0, 0]	local position 1
Position2				
Position2.element_number	integer		0	Number of constrained element (0 if ground joint)
Position2.position	vector		[0, 0]	local or global position 2

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-2
Internal.algebraic_variable	algebraic variables of the element. range: 1-2

Connector.force	force applied to the kinematic pairs due to the connector. range: 1-2, corresponds to force in global x-y direction

Example

see file PointJoint2D.txt

```
grav
{
  load_type= "Gravity"  %specification of load type.
  direction= 1  %global direction of the gravity
  gravity_constant= 9.81  %use negative sign if necessary
}
nLoad = AddLoad(grav)

mass
{
  element_type= "Mass2D"  %specification of element type.
  loads= [nLoad]
  Initialization.initial_position= [1, 0.5]  %initial position [x,y]
  Physics.mass= 1  %total mass of point mass
}
nMass = AddElement(mass)

sda
{
  element_type= "PointJoint2D"  %specification of element type.
  Position1.element_number= nMass  %Number of constrained element
}
nSDA = AddConnector(sda)
```

3.4 Control elements

These control elements are available:

- IODiscreteTransferFunction, 3.4.1
- IODigitalFilter, 3.4.2
- IORandomSource, 3.4.3
- IOLinearTransformation, 3.4.4
- IOQuantizer, 3.4.5
- IOContinuousTransferFunction, 3.4.6
- IOLinearODE, 3.4.7
- IOMathFunction, 3.4.8
- IOSaturate, 3.4.9
- IODeadZone, 3.4.10
- IOProduct, 3.4.11
- IOTime, 3.4.12
- IOPulseGenerator, 3.4.13
- IOTimeWindow, 3.4.14
- IOStopComputation, 3.4.15
- IOElementDataModifier, 3.4.16
- IODisplay, 3.4.17
- IOMinMax, 3.4.18
- IOTCPIPBlock, 3.4.19
- IOX2C, 3.4.20
- IOLinearTransducer, 3.4.21

Control elements are connectors which have input- and/or output-ports.
Note:
In HOTINT several classes are treated as 'elements'. Connectors and control elements are also 'elements', and can therefore be edited and deleted in the GUI with the menu items of the elements.
In the script language the command `AddConnector` has to be used for connectors and also for the control elements in the list above.

3.4.1 IODiscreteTransferFunction

Short description

Discontinuous transfer function in z-space. It is a SISO (single input-single output) control element. Inital state is zero.

Equations

$y(z) = \mathbf{G}(z)u(z)$

$\mathbf{G}(z) = \frac{\mathbf{num}}{\mathbf{den}}$

user input:
$\mathbf{num}(z) = num_1 + num_2 z + num_3 z^2 + ... + num_{n+1} z^n$
$\mathbf{den}(z) = den_1 + den_2 z + den_3 z^2 + ... + den_{n+1} z^n$
Theoretical background: Realization of z-transfer function as time discrete state space model

$$\begin{bmatrix} z_{k+1,1} \\ . \\ . \\ . \\ z_{k+1,n} \end{bmatrix} = \begin{bmatrix} 0 & . & . & 0 & -den_1 \\ 1 & 0 & . & 0 & -den_2 \\ . & . & . & . & . \\ 0 & 0 & . & 1 & -den_n \end{bmatrix} \cdot \begin{bmatrix} z_{k,1} \\ . \\ . \\ . \\ z_{k,n} \end{bmatrix} + \begin{bmatrix} num_1 - num_{n+1} den_1 \\ . \\ . \\ num_n - num_{n+1} den_n \end{bmatrix} u_k \qquad (3.30)$$

$$y_k = z_{k,n} + num_{n+1} u_z; \qquad (3.31)$$

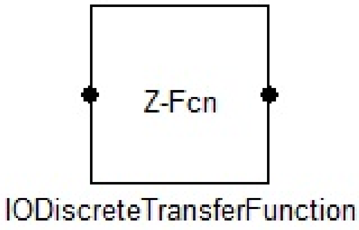


Figure 3.39: IODiscreteTransferFunction

Data objects of IODiscreteTransferFunction:

Data name	type	R	default	description
element_type	string		"IODiscreteTransferFunction"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IODiscreteTransferFunction"	name of the element

element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock. discrete_time_step	double		0	Sample time dT
IOBlock. discrete_time_offset	double		0	Sample offset off: Tk = k*dT + off
IOBlock.num_coeffs	vector		[1]	Coefficients of numerator polynomial of z-function
IOBlock.den_coeffs	vector		[1]	Coefficients of denominator polynomial of z-function

Observable special values:

For more information see section 3.1

value name	**description**
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-2
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file ZTransferFunction.txt

```
Include("addTime.txt")

ztransferfunction
{
  element_type = "IODiscreteTransferFunction"
  IOBlock
  {
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
    discrete_time_step = 0.5
  }
  Graphics
  {
    position = [50,0]
  }
}
nZTransferFunction = AddElement(ztransferfunction)

nSens = nZTransferFunction
nDisp = nZTransferFunction

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.2 IODigitalFilter

Short description

A digital (highpass or lowpass) filter of 2nd order.

Equations

The filter is fully defined by the following parameters:

- fc, cut off frequency in Hz
- fs = 1/deltaT, sampling frequency
- Q-factor, see remarks below

The Q-factor is influencing the damping of the filter, the following values are important:

- Q=0.5 critically damped
- Q<0.5 overdamped
- Q=1/sqrt(2) Butterworth
- Q=1/sqrt(3) Bessel

The coefficients of the 2nd order discrete transfer function are computed automatically in this IOBlock

Description of the different modi

lowpass (default)	the flag highpass is set to 0.
highpass	the flag highpass has to be set to 1

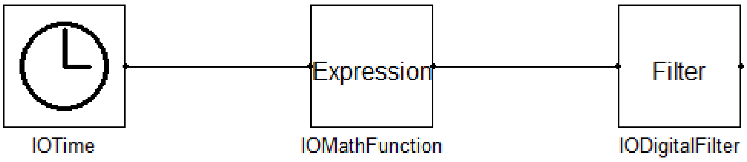


Figure 3.40: IODigitalFilter

Data objects of IODigitalFilter:

Data name	type	R	default	description
element_type	string		"IODigitalFilter"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IODigitalFilter"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock. discrete_time_step	double		0.001	Sample time dT
IOBlock.highpass	bool		0	1..highpass, 0..lowpass
IOBlock.fc	double		100	cut off frequency of filter in Hz
IOBlock.Q	double		0.707	Q-factor: Q = 0.5..critically damped, Q smaller 0.5..overdamped

IOBlock.num_coeffs	vector	**R**	[0.0675, 0.135, 0.0675]	Coefficients of numerator polynomial of z-function
IOBlock.den_coeffs	vector	**R**	[0.413, -1.14, 1]	Coefficients of denominator polynomial of z-function

Observable special values:

For more information see section 3.1

value name	**description**
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-4
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file IODigitalFilter.txt

```
Time.element_type= "IOTime"
nETime = AddElement(Time)

Signal
{
  element_type= "IOMathFunction"
  Graphics.position= [50, 0]
  IOBlock.input_element_numbers= [nETime]
  IOBlock.input_element_types= [1]  % 1=IOElement
  IOBlock.input_local_number= [1]
  IOBlock.MathFunction.parsed_function= "10*sin(2*pi*5*t)+2*sin(2*pi*300*t)"
  IOBlock.MathFunction.parsed_function_parameter= "t"
}
nESig=AddElement(Signal)

Filter
{
  element_type= "IODigitalFilter"
  Graphics.position= [100, 0]
  IOBlock.input_element_numbers= [nESig]
  IOBlock.input_element_types= [1]  % 1=IOElement
  IOBlock.input_local_number= [1]
  IOBlock.discrete_time_step= 0.001  %Sample time dT
  IOBlock.fc= 10  %cut off frequency of filter in Hz
  IOBlock.Q= 0.7  %Q-factor
}
nEFilt=AddElement(Filter)

SensorOutput
{
  sensor_type= "ElementSensor"
  element_number= nESig
  value= "IOBlock.output[1]"
}
AddSensor(SensorOutput)

SensorOutput.element_number= nEFilt
AddSensor(SensorOutput)
```

3.4.3 IORandomSource

Short description

Discontinuous random source using alternatively an internal C++ based pseudo random generator or a linear feedback shift register. It has no input and one output.

Description of the different modi

method 0	IOBlock.method must be set to 0. The built-in random generator is used.
method 1	IOBlock.method must be set to 1. Generate a pseudo random binary signal by using Linear Feedback Shift Register.

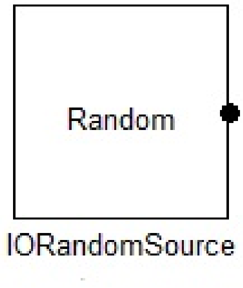


Figure 3.41: IORandomSource

Data objects of IORandomSource:

Data name	type	R	default	description
element_type	string		"IORandomSource"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!

name	string		"IORandomSource"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. discrete_time_step	double		0	Sample time dT
IOBlock. discrete_time_offset	double		0	Sample offset off: Tk = k*dT + off
IOBlock.max_amplitude	double		0	Max. amplitude of random value.
IOBlock.mean_value	double		0	Offset of random signal.
IOBlock.method	bool		0	Random generator method.
IOBlock.bits	integer		15	Number of bits for random signal.
IOBlock. constant_amplitude	bool		0	Output values are +amplitude or -amplitude if flag is activate.
IOBlock.seed	double		0	seed Ă [0.,1.]... initialization of random generator
IOBlock.init_val	double		0	initial value of the generator x(t=0) = y(t=0)

Observable special values:

For more information see section 3.1

value name	**description**
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-1
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file addRandomSource.txt

```
random
{
  element_type = "IORandomSource"
  IOBlock
  {
    discrete_time_step = 0.05
    discrete_time_offset = 0.0
    max_amplitude = 2
    mean_value =2.5
    method = 1
    bits = 15
    constant_amplitude = 0
    seed = 0.5
    init_val = 2.5
  }
  Graphics
  {
    position = [0,-50]
  }
}
nRandom = AddElement(random)
```

3.4.4 IOLinearTransformation

Short description

Continuous linear transformation. The transfer function type is SISO (single input-single output) or MIMO (multi input-multi output).

Equations

$$\mathbf{y} = \mathbf{A}\mathbf{u} + \mathbf{b}; \tag{3.32}$$

Matrix **A** and vector **b** are user defined.

Description of the different modi

linear transformation $y = A\,u$	Set **b** to zero.
gain $y_1 = A_{1,1}u_1$	Set A as scalar value and b is zero.
constant $y_1 = b_1$	Set A to zero and b to the constant value.

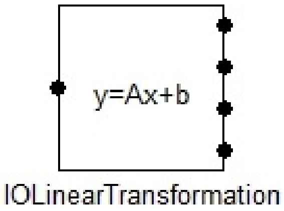


Figure 3.42: IOLinearTransformation

Data objects of IOLinearTransformation:

Data name	type	R	default	description
element_type	string		"IOLinearTransformation"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOLinearTransformation"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	4	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.A_matrix	matrix		[0, 0, 0, 0; 0, 0, 0, 0; 0, 0, 0, 0; 0, 0, 0, 0]	transformation matrix A: y=A.u+b
IOBlock.b_vector	vector		[0, 0, 0, 0]	offset vector b: y=A.u+b

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file LinearTransformation.txt

```
Include("addTime.txt")

transformation
{
  element_type = "IOLinearTransformation"
  Graphics
  {
    position = [50, 0]  %reference drawing position
  }
  IOBlock
  {
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
    A_matrix = [2]
    b_vector = [0.5]
  }
}
nTrans = AddElement(transformation)

nSens = nTrans
nDisp = nTrans

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.5 IOQuantizer

Short description

A quantizer block passes its input signal through a stair-step function so that many neighboring points on the input axis are mapped to one point on the output axis. The effect is to quantize a smooth signal into a stair-step output. It is a SISO (single input-single output) control element.

Equations

$$y(u) = \begin{cases} r\,floor\left(\frac{u}{r} + 0.5r\right), & \text{if } r! = 0 \\ u, & \text{if } r = 0 \end{cases} \tag{3.33}$$

The user defined rounding value is r.

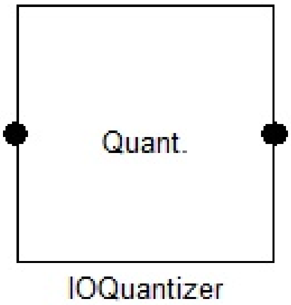


Figure 3.43: IOQuantizer

Data objects of IOQuantizer:

Data name	type	R	default	description
element_type	string		"IOQuantizer"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOQuantizer"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.rounding_value	double		0.1	Max. amplitude of random value.

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file Quantizer.txt

```
Include("addTime.txt")

quantizer
{
  element_type = "IOQuantizer"
  IOBlock
  {
    rounding_value = 0.2
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nQuantizer = AddElement(quantizer)

nSens = nQuantizer
nDisp = nQuantizer

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.6 IOContinuousTransferFunction

Short description

The STransferFunction is a linear transfer function for continuous state-space elements. It is a SISO (single input-single output) type.

Equations

$y(s) = \mathbf{G}(s)u(s)$

$\mathbf{G}(s) = \frac{\mathbf{num(s)}}{\mathbf{den(s)}}$

user input:
$\mathbf{num}(s) = num_1 + num_2 s + num_3 s^2 + ... + num_{n+1} s^n$
$\mathbf{den}(s) = den_1 + den_2 s + den_3 s^2 + ... + den_{n+1} s^n$

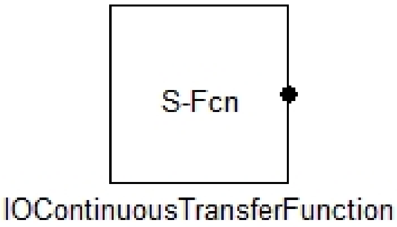


Figure 3.44: IOContinuousTransferFunction

Data objects of IOContinuousTransferFunction:

Data name	type	R	default	description
element_type	string		"IOContinuousTransferFunction"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOContinuousTransferFunction"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	3	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor

IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.numerator	vector		[1, 0, 0, 0]	ascending numerator coefficients n of transfer-function. TF = num/den with num = n(1)*1+n(2)*s+n(3)*s*s+... Will be normalized automatically!
IOBlock.denominator	vector		[0, 0, 0, 1]	ascending denominator coeffs d of transfer-function. TF = num/den with den = d(1)*1+d(2)*s+d(3)*s*s+... Will be normalized automatically!

Observable special values:

For more information see section 3.1

value name	**description**
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-3
Internal.first_order_variable	first order variables of the element. range: 1-3
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file STransferFunction.txt

```
Include("addTime.txt")

stransferfunction
{
  element_type = "IOContinuousTransferFunction"
  IOBlock
  {
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nSTransferFunction = AddElement(stransferfunction)

nSens = nSTransferFunction
nDisp = nSTransferFunction

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.7 IOLinearODE

Short description

The LinearODE Element represents a linear ordinary differential equation of SISO (single input-single output) or MIMO (multi input-multi output) type.

Equations

$$\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\,\mathbf{x} + \mathbf{D}\,\mathbf{u}$$

Matrices **A**, **B**, **C** and **D** are user defined.

lin ODE
IOLinearODE

Figure 3.45: IOLinearODE

Data objects of IOLinearODE:

Data name	type	R	default	description
element_type	string		"IOLinearODE"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOLinearODE"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics.background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics.foreground_color	vector		[0, 0, 0]	foreground color
Graphics.input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock.number_of_inputs	integer	**R**	0	number of inputs

IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.A_coeffs	matrix		[0]	Coefficients of state matrix A, x_dot = A*x + B*u
IOBlock.B_coeffs	matrix		[0]	Coefficients of input matrix B, x_dot = A*x + B*u
IOBlock.C_coeffs	matrix		[0]	Coefficients of output matrix C, y = C*x + D*u
IOBlock.D_coeffs	matrix		[0]	Coefficients of output matrix D, y = C*x + D*u
IOBlock.initital_vector	vector		[]	Initial values of time-domain variables

Observable special values:

For more information see section 3.1

value name	**description**
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file LinearODE.txt

```
Include("addTime.txt")

lin
{
  element_type = "IOLinearODE"
  Graphics
  {
    position = [50, 0]  %reference drawing position
  }
  IOBlock
  {
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
    A_coeffs = [1]
    B_coeffs = [1]
    C_coeffs = [1]
    D_coeffs = [1]
    initital_vector = [1]
  }
}
nLin = AddElement(lin)

nSens = nLin
nDisp = nLin

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.8 IOMathFunction

Short description

A IOMathFunction contains a mathematical expression with functions and logical operators or a lookup table with different modes for piecewise interpolation. The output is result of the evalutation of the MathFunction as a function of input.

Description of the different modi

parsed function	In order to use the parser for mathematical expressions, the variable IOBlock.MathFunction.piecewise_mode has to be set to -1. In IOBlock.MathFunction.parsed_function one specifies a string representing parsed function, e.g. '$A * sin(u)$' with function parameter u defined in IOBlock.MathFunction.parsed_function_parameter.
piecewise mode - constant	IOBlock.MathFunction.piecewise_mode must be set to 0. The vectors IOBlock.MathFunction.piecewise_points and IOBlock.MathFunction.piecewise_values are used. The output value is piecewise constant with jumps at the supporting points.
piecewise mode - linear	IOBlock.MathFunction.piecewise_mode must be set to 1. The vectors IOBlock.MathFunction.piecewise_points and IOBlock.MathFunction.piecewise_values are used. The output value is piecewise linear between the supporting points.
piecewise mode - quadratic	IOBlock.MathFunction.piecewise_mode must be set to 2 and in addition to the other piecwise modes the vector IOBlock.MathFunction.piecewise_diff_values is needed. The output is a quadratic interpolation between the supporting points.

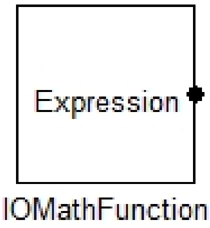


Figure 3.46: IOMathFunction

Data objects of IOMathFunction:

Data name	**type**	**R**	**default**	**description**
element_type	string		"IOMathFunction"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOMathFunction"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4==360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.MathFunction				
IOBlock.MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
IOBlock.MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation

IOBlock.MathFunction. piecewise_values	vector		[]	values at supporting points
IOBlock.MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
IOBlock.MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
IOBlock.MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'

Observable special values:

For more information see section 3.1

value name	**description**
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-1
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file IOMathFunction.txt

```
Time
{
  element_type= "IOTime"
}
nTime = AddElement(Time)

% IOMathfunction with one input piecewise
IOBlock
{
  element_type= "IOMathFunction"
  Graphics.position= [50, 0]
  IOBlock
  {
    input_element_numbers= [nTime]
    input_element_types= [1]
    input_local_number= [1]
    MathFunction
    {
      piecewise_mode= 0
      piecewise_points= [0, 1, 1.5, 2]
      piecewise_values= [0, 1, 0.7, 0]
    }
  }
}
```

```
nElem = AddElement(IOBlock)

SensorOutput
{
  sensor_type= "ElementSensor"
  element_number= nElem
  value= "IOBlock.output[1]"
}
AddSensor(SensorOutput)

% IOMathfunction with multiple inputs and parsed function
IOBlock
{
  element_type= "IOMathFunction"
  Graphics.position= [100, 0]
  IOBlock
  {
    input_element_numbers= [nTime, nTime]
    input_element_types= [1, 1]
    input_local_number= [1, 1]
    MathFunction
    {
      piecewise_mode= -1
      parsed_function = "u*((v>4)&&(v<6))"
      parsed_function_parameter = "u,v"
    }
  }
}
nElem = AddElement(IOBlock)

SensorOutput.element_number = nElem
AddSensor(SensorOutput)
```

3.4.9 IOSaturate

Short description

Continuous saturation element for upper and lower limits. It is a SISO (single input-single output) control element.

Equations

$$y(u) = \begin{cases} ul, & \text{if } u > ul \\ u, & \text{if } ll \leq u \leq ul \\ ll, & \text{if } u < ll \end{cases} \tag{3.34}$$

In the defined equation ul is the upper limit and ll is the lower limit.

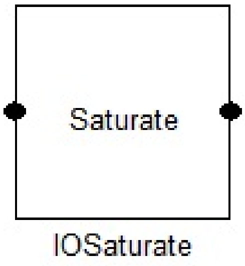


Figure 3.47: IOSaturate

Data objects of IOSaturate:

Data name	type	R	default	description
element_type	string		"IOSaturate"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOSaturate"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics.background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics.foreground_color	vector		[0, 0, 0]	foreground color
Graphics.input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock.number_of_inputs	integer	**R**	0	number of inputs
IOBlock.number_of_outputs	integer	**R**	0	number of outputs
IOBlock.number_of_states	integer	**R**	0	number of states
IOBlock.input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock.input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock.input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.upper_limit	double		0.1	Upper limit of saturate.
IOBlock.lower_limit	double		0	Lower limit of saturate.

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file Saturate.txt

```
Include("addTime.txt")

saturate
{
  element_type = "IOSaturate"
  IOBlock
  {
    upper_limit = 2.6
    lower_limit = 2.4
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nSaturate = AddElement(saturate)

nSens = nSaturate
nDisp = nSaturate

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.10 IODeadZone

Short description

Continuous dead-zone element. The outputs between upper and lower limit is zero. This leads to an offset of the input signal by the corresponding lower or upper limit. It is a SISO (single input-single output) control element.

Equations

$$y(u) = \begin{cases} u - sd, & \text{if } u < sd \\ 0, & \text{if } u \geq sd \text{ and } u \leq ed \\ u - ed, & \text{if } u > ed \end{cases} \tag{3.35}$$

In the defined equation sd is the start dead-zone value, ed is the end dead-zone value.

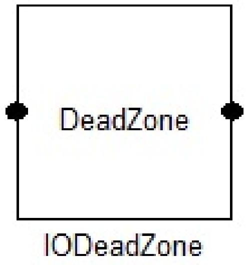


Figure 3.48: IODeadZone

Data objects of IODeadZone:

Data name	type	R	default	description
element_type	string		"IODeadZone"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IODeadZone"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor

IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.start_deadzone	double		0	Start of dead zone.
IOBlock.end_deadzone	double		0	End of dead zone.

Observable special values:

For more information see section 3.1

value name	**description**
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file DeadZone.txt

```
Include("addTime.txt")

deadzone
{
  element_type = "IODeadZone"
  IOBlock
  {
    start_deadzone = 1
    end_deadzone = 2
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nDeadZone = AddElement(deadzone)

nSens = nDeadZone
nDisp = nDeadZone

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.11 IOProduct

Short description

Continuous product (or division) of one or more inputs. A dedicated exponent for every input and a offset can be applied.

Equations

$$y(\mathbf{u}) = u_1^{exp_1} u_2^{exp_2} ... u_n^{exp_n} + offset \tag{3.36}$$

All exponents are stored in a vector. For a simple multiplication with a input the dedicated exponent is set to 1, for a division the exponent is set to -1. The offset is a scalar value.

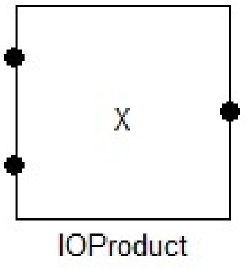


Figure 3.49: IOProduct

Data objects of IOProduct:

Data name	type	R	default	description
element_type	string		"IOProduct"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOProduct"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states

IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.exponents	vector		[0]	Exponent of inputs. y=u1êxp1*u2êxp2*...*unêxpn+offset.
IOBlock.offset	double		0	Output offset.

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file Product.txt

```
Include("addTime.txt")
Include("addRandomSource.txt")

product
{
  element_type = "IOProduct"
  IOBlock
  {
    exponents = [2,3]
    offset = -1
    input_element_numbers = [nTime,nRandom]
    input_element_types = [1,1]
    input_local_number = [1,1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nProduct = AddElement(product)

nSens = nProduct
nDisp = nProduct
```

```
Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.12 IOTime

Short description

Continuous time source. This element simply outputs the time.

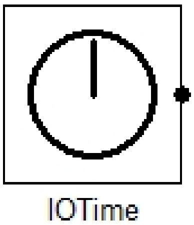


Figure 3.50: IOTime

Data objects of IOTime:

Data name	type	R	default	description
element_type	string		"IOTime"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOTime"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file addTime.txt

```
time
{
  element_type = "IOTime"
  name = "time"
  Graphics
  {
    position = [0, 0]  %reference drawing position
    draw_size = [20, 20, 0]  %draw size
  }
}
nTime = AddElement(time)
```

3.4.13 IOPulseGenerator

Short description

Continuous pulse generator. This element outputs repeating sequence or rectangular pulses after a certain delay. It has no input and one output.

Equations

$$\Delta t = t - t_{offset} \tag{3.37}$$

$$t_{rest} = \Delta t \bmod p \tag{3.38}$$

$$y(t) = \begin{cases} a, & \text{if } \Delta t \geq 0 \text{ and } t_{rest} < pw \\ 0, & \text{else} \end{cases} \tag{3.39}$$

User defined variables are pulse amplitude a, time offset t_{offset}, signal period p and pulse width pw.

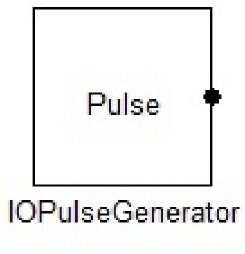


Figure 3.51: IOPulseGenerator

Data objects of IOPulseGenerator:

Data name	**type**	**R**	**default**	**description**
element_type	string		"IOPulseGenerator"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOPulseGenerator"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock.amplitude	double		1	Amplitude of rectangle pulse generator.
IOBlock.offset	double		0	Time offset (s).
IOBlock.period	double		1	Period of signal (s).
IOBlock.pulse_width	double		0.5	Pulse width (s).
IOBlock. use_external_time_source	integer	**R**	0	1|(0) ... (Don't) use external input as time source.

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file addPulseGenerator.txt

```
pulse
{
  element_type = "IOPulseGenerator"
  IOBlock
  {
  amplitude = 2
  offset = 1
  period = 0.2
  pulse_width = 0.1
  }
  Graphics
  {
    position = [0,-50]
  }
}
nPulse = AddElement(pulse)
```

3.4.14 IOTimeWindow

Short description

This element helps to capture a special time window. It has two inputs and one output.

Equations

$$(a) \qquad y(\mathbf{u}) = \begin{cases} u_2, & \text{if } t_{start} \leq u_1 \leq t_{end} \\ 0, & \text{else} \end{cases} \tag{3.40}$$

$$(b) \qquad y(\mathbf{u}) = \begin{cases} u_2, & \text{if } t_{start} \leq u_1 \\ 0, & \text{else} \end{cases} \tag{3.41}$$

Description of the different modi

$t_{end} > t_{start}$	Output is determined with inequation (a).
$t_{end} \leq t_{start}$	Output is determined with inequation (b).

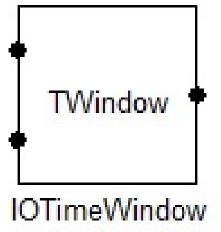


Figure 3.52: IOTimeWindow

Data objects of IOTimeWindow:

Data name	type	R	default	description
element_type	string		"IOTimeWindow"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOTimeWindow"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.t_start	double		0	Start time (s).
IOBlock.t_end	double		0	End time (s).

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file TimeWindow.txt

```
Include("addTime.txt")
Include("addPulseGenerator.txt")

window
{
  element_type = "IOTimeWindow"
  IOBlock
  {
    t_start = 1
    t_end = 2
    input_element_numbers = [nTime,nPulse]
    input_element_types = [1,1]
    input_local_number = [1,1]
  }
  Graphics
  {
    position = [50,0]
  }
}
nWindow = AddElement(window)

nSens = nWindow
nDisp = nWindow

Include("addSens.txt")
Include("addDisplay.txt")
```

3.4.15 IOStopComputation

Short description

This element stops the computation, if input is unequal zero. It has one input and no output.

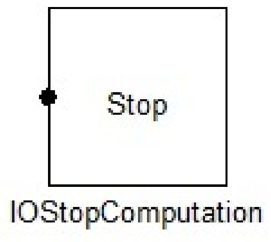


Figure 3.53: IOStopComputation

Data objects of IOStopComputation:

Data name	type	R	default	description
element_type	string		"IOStopComputation"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOStopComputation"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file StopComputation.txt

```
Include("addPulseGenerator.txt")

stop
{
  element_type = "IOStopComputation"
  IOBlock
  {
    input_element_numbers = [nPulse]
    input_element_types = [1]
    input_local_number = [1]
  }
  Graphics
  {
    position = [50,-50]
  }
}
nStop = AddElement(stop)
```

3.4.16 IOElementDataModifier

Short description

This element can be used to modify data of a constraint or element. It has one input and no output.

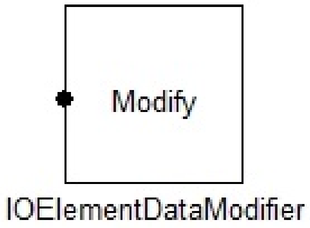


Figure 3.54: IOElementDataModifier

Data objects of IOElementDataModifier:

Data name	**type**	**R**	**default**	**description**
element_type	string		"IOElementDataModifier"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOElementDataModifier"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics.background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics.foreground_color	vector		[0, 0, 0]	foreground color
Graphics.input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock.number_of_inputs	integer	**R**	0	number of inputs
IOBlock.number_of_outputs	integer	**R**	0	number of outputs
IOBlock.number_of_states	integer	**R**	0	number of states
IOBlock.input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock.input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock.input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.start_of_timestep_only	bool		0	modify element data at start time step only.
IOBlock.mod_variable_name	string		""	variable name of the modified element data
IOBlock.mod_element_number	integer		1	element number of the modified element or constraint

Observable special values:

For more information see section 3.1

value name	description
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file ElementDataModifier.txt

```
gravLoad
{
  load_type = "Gravity"
  direction = 3 % z - direction
  gravity_constant = 9.81 % m/s^2
}
nLoad = AddLoad(gravLoad)

mass3D
{
  element_type = "Mass3D"
  loads = [nLoad]
  Physics.mass= 1
}
nMass3D = AddElement(mass3D)

IOTime
{
  element_type = "IOTime"
}
nIOTime = AddElement(IOTime)

springDamperActuator
{
  element_type = "SpringDamperActuator"
  Position1.element_number = nMass3D %number of constrained element
}
nSpringDamperActuator = AddConnector(springDamperActuator)

modifier
{
  element_type = "IOElementDataModifier"
  IOBlock
  {
    input_element_numbers = [nIOTime]  %element connected to input
    input_element_types = [1]
    input_local_number = [1]
    mod_variable_name = "Connector.SpringDamperActuator.spring_length_offset"  %modified elemen
    mod_element_number = nSpringDamperActuator  %modified constraint
  }
}
AddElement(modifier)
```

3.4.17 IODisplay

Short description

This element can be used to display any (single) numberical value fed into the (single) input.

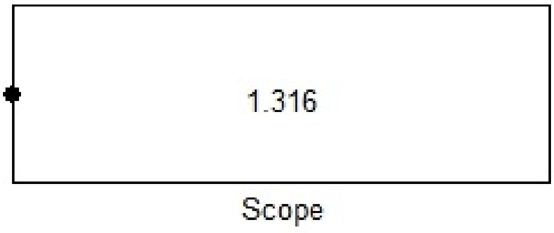


Figure 3.55: IODisplay

Data objects of IODisplay:

Data name	type	R	default	description
element_type	string		"IODisplay"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IODisplay"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[60, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4==360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs

IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock. number_of_digits	integer		3	number of digits

Observable special values:

For more information see section 3.1

value name	**description**
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-1
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file Display.txt

```
Include("addTime.txt")

display
{
  element_type = "IODisplay"
  IOBlock
  {
    input_element_numbers = [nTime]
    input_element_types = [1]
    input_local_number = [1]
    number_of_digits = 3
  }
  Graphics
  {
    position = [70,0]
  }
}
nDisplay = AddElement(display)
```

3.4.18 IOMinMax

Short description

This block returns the minimum, maximum or average value of the input. Up to a specific point of time, this functionality is switched off and the output y is equal to the input u. This block can be used to postprocess sensor values.

Description of the different modi

1 = minimum	$y = u$ for $t \leq t_0$ $y = \min_{t \geq t_0}(u)$ for $t > t_0$ with t_0 = IOBlock.start_time
2 = maximum	$y = u$ for $t \leq t_0$ $y = \max_{t \geq t_0}(u)$ for $t > t_0$
3 = average	$y = u$ for $t \leq t_0$ $y = \frac{1}{N} \sum_{t_i \geq t_0} u_i$ for $t_i > t_0$
4 = minimum(abs)	$y = u$ for $t \leq t_0$ $y = \min_{t \geq t_0}(\lvert u \rvert)$ for $t > t_0$
5 = maximum(abs)	$y = u$ for $t \leq t_0$ $y = \min_{t \geq t_0}(\lvert u \rvert)$ for $t > t_0$
6 = average(abs)	$y = u$ for $t \leq t_0$ $y = \frac{1}{N} \sum_{t_i \geq t_0} \lvert u_i \rvert$ for $t_i > t_0$

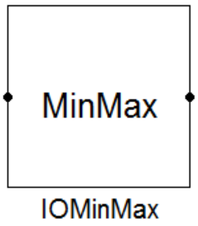


Figure 3.56: IOMinMax

Data objects of IOMinMax:

Data name	**type**	**R**	**default**	**description**
element_type	string		"IOMinMax"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOMinMax"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color

Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	1	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.mode	integer		1	1..min, 2..max, 3..avg, 4..min(abs), 5..max(abs), 6..avg(abs)
IOBlock.start_time	double		0	Up to this point of time, the output is equal to the input. Afterwards the output is computed according to the mode.

Observable special values:

For more information see section 3.1

value name	description
Internal.data_variable	data varibales of the element which are no degrees of freedom (e.g. inelastic strain, contact state, friction state, etc.). range: 1-2
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file IOMinMax.txt

```
Time.element_type= "IOTime"
nElem1 = AddElement(Time)

MinMax
{
  element_type= "IOMinMax"
  Graphics.position= [50, 0]
  IOBlock
  {
    input_element_numbers= [nElem1]
    input_element_types= [1]
    input_local_number= [1]
  mode = 1 % minimum
  start_time = 0.5
  }
}
nElem2 = AddElement(MinMax)

SensorOutput
{
  sensor_type= "ElementSensor"
  element_number= nElem1
  value= "IOBlock.output[1]"
}
AddSensor(SensorOutput)

SensorOutput.element_number= nElem2
AddSensor(SensorOutput)
```

3.4.19 IOTCPIPBlock

Short description

This I/O element is a communication block based on TCP/IP which allows HOTINT to connect to other programs or tools, opening up a wide range of possible applications including external control, user-defined "add-ons", or even co-simulation. Based on the specified IP (v4) address and port number the IOTCPIPBlock sets up a server socket and waits for a connection request from a client. Hence, HOTINT here plays the server role, and the external program is the client application.

Limitations

For the use of this element some kind of active network adapter is required. If you only want to communicate locally on your computer and do not have an active network adapter, you can alternatively use a so-called "loopback device" which emulates an active real network adapter in a real network. To this end, either use the *localhost* address 127.0.0.1 (this is the default) – or one address from the 127.0.0.0/8 subnet (127.0.0.1-127.255.255.254) – or create and configure an actual virtual network adapter. The following steps summarize how such a loopback adapter can be installed on Microsoft Windows:

(1) Open the device manager
(2) Select the network category and choose "Action → Add legacy hardware" via the menu
(3) Choose the option for manual installation and select the category "network adapters" from the list
(4) In the next dialog select "Microsoft" as vendor and "Microsoft loopback adapter" as hardware component
(5) Proceed and finish the installation

Example: Communication with Simulink/Matlab
This example demonstrates how to realize a connection between HOTINT and Matlab/Simulink. The purpose of the TCP/IP block is to use other powerful tools for some computations. For example it is possible to do the control law calculations for the actuation of the multibody system

in Simulink (as alternative to the IOBlocks in HOTINT). It's also very simple to do a parameter variation, see the advanced example in the folder "examples/balancing_cart_TCPIP".
In "examples/TCPIP" a very simple communication example is included: From the HOTINT side four different double values (simulation time t multiplied by the gain factors one to four) are transmitted to Simulink, see figure 3.58. Simulink summates the first and second respectively the third and fourth value and sends the two double values back to HOTINT. The values are captured by sensors, stored in the solution file and can be visualized in the plot tool.
Comment: For testing purposes you can also use the executable "TCPIP_client.exe" which has the same functionality as the Simulink example. To use this client executable create a "IP.txt" file in the same folder. The first four lines represent the IP address of the HOTINT computer, the fifth line is the port number.

To start this example, following things have to be done:
(1) Start Matlab/Simulink and open the file communication.mdl in the folder "examples/TCPIP", see figure 3.59.
Comment: If the "Instrument Control Toolbox" is not installed the TCP/IP communication blocks appear red and indicate an unresolved reference to a library block (bad link). The figure shows the basic structure that should not be changed. The output of the "TCP/IP Receive" block is a vector $y_{rec} = [t, x_1, ..., x_n, f]^T$ with HOTINT time t, data variables x_1 to x_n and the handling flag f. The "Selector" block outputs the last element of the vector (flag f) for the flag handling. You have to adapt the these two blocks if you want to change the number of received variables. There is no need to change the "flag handling in" block.
(2) Make sure that the "Current Folder" is the folder which include the communication.mdl file.
(3) Double click the "TCP/IP Receive" block and select the "Remote address" (i.e., the IP address) of the computer HOTINT is running on and select a "Port". Repeat this point for the "TCP/IP Send" block.
Comment: If HOTINT and Simulink is running on the same computer you can also choose localhost ("127.0.0.1").
(4) Set the "Sample Time" of every block (TCP/IP Receive, Constants,...) and choose fixed step size in the "Solver Options".
(5) Open the subsystem "computations", see figure 3.60. This subsystem contains all computations $\mathbf{y} = \mathbf{f}(\mathbf{u})$ with input $\mathbf{u}$ and output $\mathbf{y}$.
Comment: Change this subsystem to your needs.
(6) Open the subsystem "flag handling out", see figure 3.62. In default no handling flags are transmitted to HOTINT.
Comment: Change this subsystem to your needs.
(7) Save the mdl file. (8) Open the TCPIP.hid HOTINT file and type in the same "ip_address" and "port_number" as for the Matlab/Simulink side.
(9) Make sure that "max_step_size" and "min_step_size" in the subtree "SolverOptions.Timeint" are set to the same value as the fixed "Sample Time" in Simulink.
Comment: This is very important especially for the case of time dependent blocks like integrators in Simulink.
(10) Save the file.
(11) Load the communication.hid file in HOTINT.
(12) Click the "Start simulation" button in Simulink.
(13) Click the "Start!" button in HOTINT.

Comment: The points 11-13 have to be executed within the timeout limits. You can change the latter in the TCP/IP blocks for both HOTINT and Simulink. During these steps connection

errors might occur due to firewall restrictions; you will probably have to set the corresponding permissions in your firewall(s).
It is also recommended to choose the Simulink "Simulation stop time" higher as the "end_time" in HOTINT. The reason is that HOTINT sends a stop flag after the last simulation step and in Simulink this flag is used to execute a "Stop" block which ends the communication and simulation.

Additional notes

Data exchange is performed at a stage before every time step in HOTINT, following below protocol:
The outgoing data, i.e. the data sent from HOTINT to the client, is an array of 8-byte double precision numbers which contains, in that order, the current simulation time (1 double), the current values of the inputs of the I/O element, and one additional element corresponding to a communication control flag (see the *Communication flags* - section below for more details). Hence, the total amount of outgoing data is (number of inputs + 2) times 8 bytes (double precision numbers). After the client has received and processed that data, it sends back a data package to HOTINT – the incoming data for the I/O element – which again consists of an array of double precision numbers, this time with the length (number of outputs + 1). The first (number of output) double precision values determine the outputs of the I/O element, and the last element again is used for the transfer of communication flags.
HOTINT now begins the computation of one time step, where the transmitted data from the client is accessible via the outputs of the IOTCPIPBlock.

Important notes

– The waiting procedure for the client connection request, as well as the send and receive operations all are so-called "blocking calls". This means that HOTINT will wait for those operations to finish, and during that time, not respond to any user input. Therefore, a reasonable timeout (default is 30 seconds) should be specified for the IOTCPIPBlock to allow TCP/IP connection or transmission error handling.

– You will probably have to adjust your firewall settings and set appropriate permissions for HOTINT and the client application.

– Depending on the implementation of the client, it might be neccessary to start the server, i.e., HOTINT, first.

– Since HOTINT is running on Microsoft Windows, the memory byte order, also called "endianness", is "Little Endian", which means that the least significant bytes/digits are stored "first" in memory, i.e., on the smallest memory address. Therefore, any data sent from or received by the IOTCPIPBlock has or must have that byte order, respectively. You probably have to take that into account on the client side, especially if the client is running on a different platform and/or architecture on another computer.

Communication flags

Currently, the following 4-byte flags are implemented:

(1) Neutral flag: `0x00000000` (integer value: `0`). This flag signals that the application is running (properly) and no further action is required.
(2) Reset flag: `0x00000001` (integer value: `1`). This flag is sent from HOTINT to the client in the first step of the computation. This can be used, for instance, to reset the client application.

(3) Error flag: `0x00000002` (integer value: `2`). Indicates that an error has occurred. If HOTINT receives the error flag, an error message is issued, the connection is closed and the program execution terminated.
(4) Close flag: `0x00000003` (integer value: `3`). This flag is sent from HOTINT to the client to indicate that the computation has finished and the connection will be closed, which is the case when the computation has actually finished, or the "Stop"-button has been hit.
(5) Any other value: Treated as error flag (3).

One of these flags is stored in and read from the last 8 bytes of the exchanged data – corresponding to one additional double precision number – in either direction in every time step. Currently, for simplicity, the flag is just casted explicitly from an integer to a double precision number which then can be transmitted and casted back to an integer exactly. Of course, this procedure must be followed on both the server and the client side.

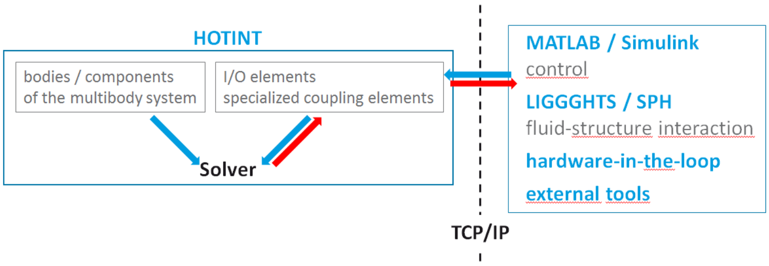

HOTINT
bodies / components of the multibody system
I/O elements specialized coupling elements
Solver
TCP/IP
MATLAB / Simulink
control
LIGGGHTS / SPH
fluid-structure interaction
hardware-in-the-loop
external tools

Figure 3.57: general concept of TCP/IP coupling

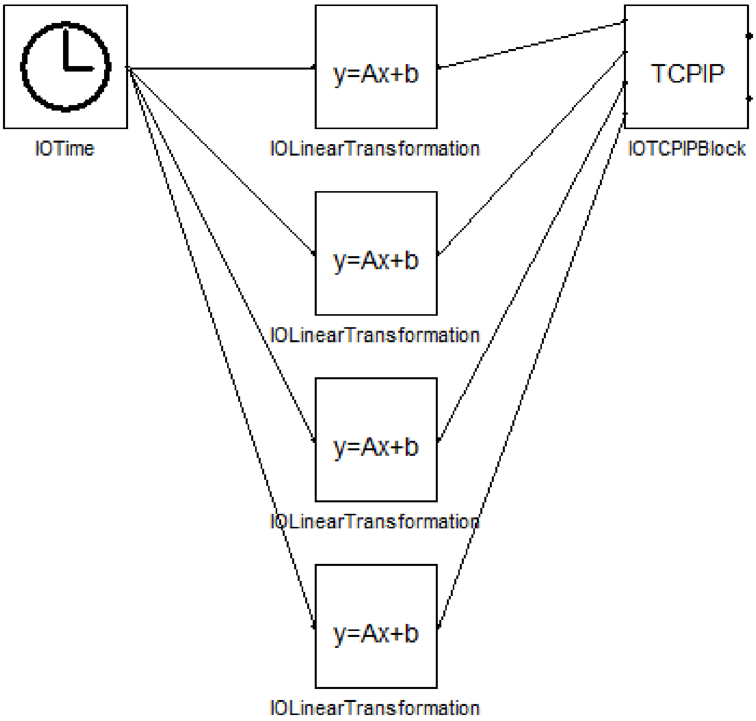


Figure 3.58: TCP/IP Block with 4 inputs and 2 outputs

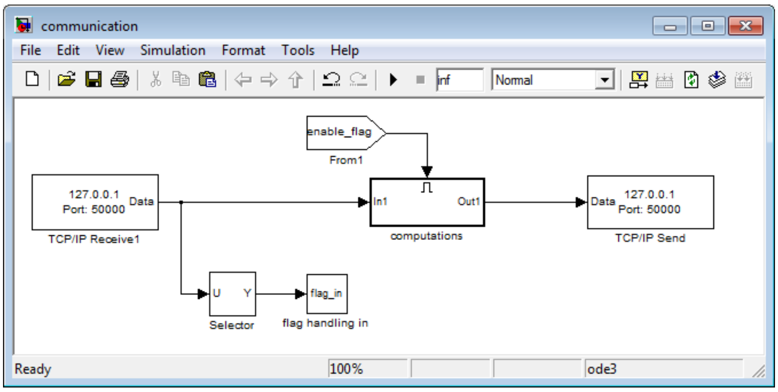


Figure 3.59: TCP/IP communication with Matlab/Simulink (do not change this structure)

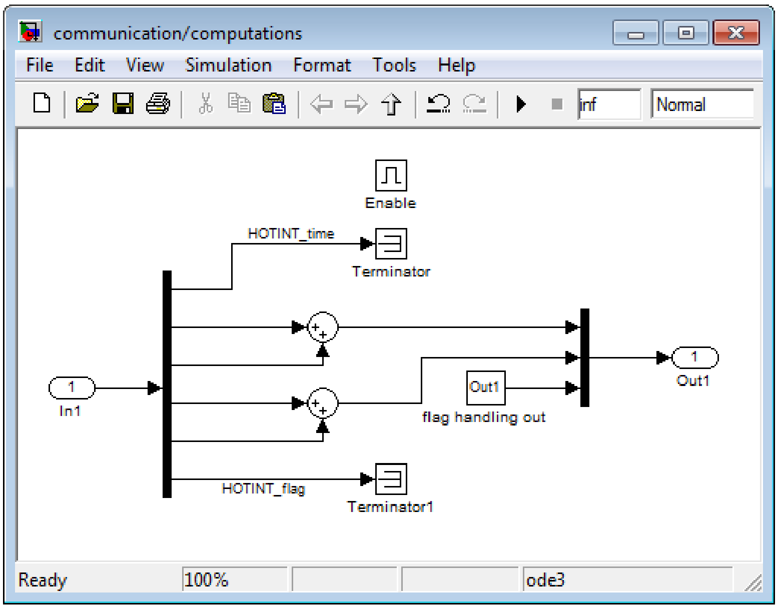


Figure 3.60: Subsystem computations

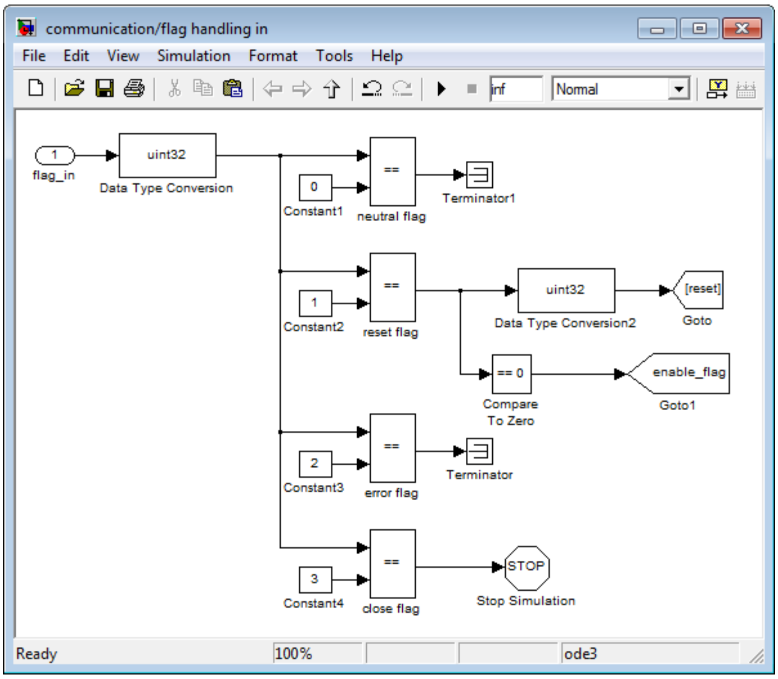


Figure 3.61: TCP/IP subsystem “flag handling in”

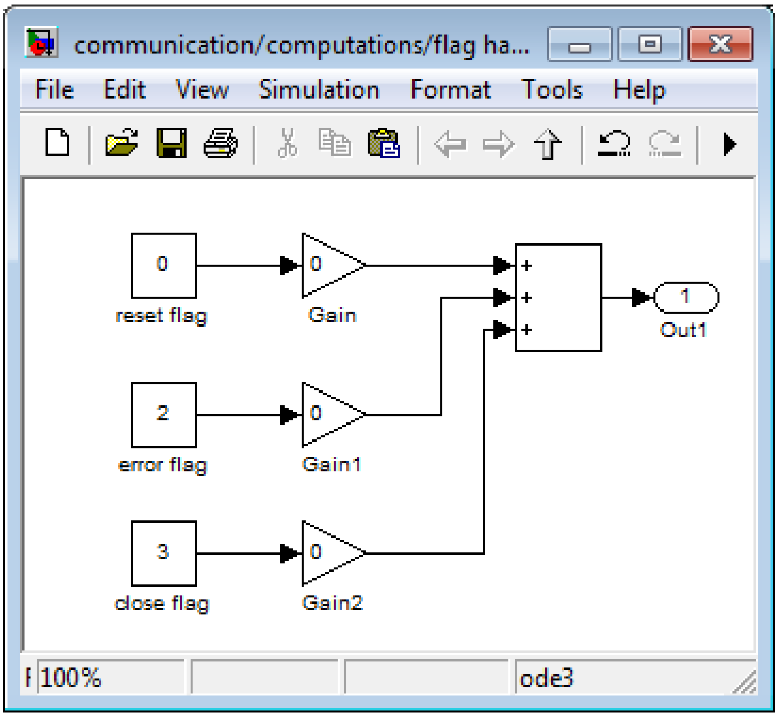

communication/computations/flag ha...
File Edit View Simulation Format Tools Help
0
reset flag
0
Gain
2
error flag
0
Gain1
3
close flag
0
Gain2
+
+
+
1
Out1
100%
ode3

Figure 3.62: TCP/IP subsystem “flag handling out”

Data objects of IOTCPIPBlock:

Data name	type	R	default	description

element_type	string		"IOTCPIPBlock"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOTCPIPBlock"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics.background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics.foreground_color	vector		[0, 0, 0]	foreground color
Graphics.input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock.number_of_inputs	integer	**R**	0	number of inputs
IOBlock.number_of_outputs	integer	**R**	0	number of outputs
IOBlock.number_of_states	integer	**R**	0	number of states
IOBlock.input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock.input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock.input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.port_number	integer		50000	Port number, e.g. '50000'.
IOBlock.ip_address	string		"127.0.0.1"	IP address, e.g. '127.0.0.1' (localhost). Do not neglect the dots between the numbers.
IOBlock.received_data_size	integer		0	Number of received data values (outputs). This number has to be consistent with the transmitted data values of the other communication side (the additional double for the communication flags is not corresponding to this number).
IOBlock.timeout	integer		30000	TCP/IP timeout in milliseconds; default is 30000.

Observable special values:

For more information see section 3.1

value name	**description**
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

Example

see file TCPIP.txt

```
time.element_type= "IOTime"  %specification of element type.
nTime = AddElement(time)

gain
{
  element_type= "IOLinearTransformation"  %specification of element type.
  Graphics.position= [50, 0]
  IOBlock
  {
    input_element_numbers= [nTime]  %v. of element(s) or sensor number(s)
    input_element_types= [1]  %v. with types of connected inputs; 1=IOElement
    input_local_number= [1]  %v. with i-th number of output of previous IOelement
    A_matrix= [1]  %transformation matrix A: y=A.u+b
    b_vector= [0]  %offset vector b: y=A.u+b
  }
}
nGain1 = AddElement(gain)

gain.IOBlock.A_matrix= [2]
gain.Graphics.position= [50, -30]
nGain2 = AddElement(gain)
gain.IOBlock.A_matrix= [3]
gain.Graphics.position= [50, -60]
nGain3 = AddElement(gain)
gain.IOBlock.A_matrix= [4]
gain.Graphics.position= [50, -90]
nGain4 = AddElement(gain)

TCPIP
{
  element_type= "IOTCPIPBlock"  %specification of element type.
  Graphics.position= [100, 0]
  IOBlock
  {
    input_element_numbers= [nGain1,nGain2,nGain3,nGain4]  %v. of sensor number(s)
    input_element_types= [1,1,1,1]  %v. w. types of connected inputs; 1=IOElement
    input_local_number= [1,1,1,1]  %v. w. i-th number of output
    port_number= 50000  %Port number, e.g. '50000'.
    ip_address= "127.0.0.1"  %IP address, e.g. '127.0.0.1'.
    received_data_size = 2  %Number of received values (outputs).
    timeout= 10000  %TCP/IP timeout in milliseconds; default is 10000.
  }
}
nTCPIP = AddElement(TCPIP)

sensor.sensor_type= "ElementSensor"
sensor.element_number= nTCPIP
sensor.value= "IOBlock.output[1]"
nSensor1= AddSensor(sensor)
```

```
sensor.name= "sens2"
sensor.value= "IOBlock.output[2]"
nSensor2= AddSensor(sensor)
SolverOptions.Timeint.max_step_size = 1
SolverOptions.Timeint.min_step_size = 1
SolverOptions.start_time = 0
SolverOptions.end_time = 10
```

3.4.20 IOX2C

Short description

This I/O element is a communication block based on IOTCPIPBlock which allows HOTINT to connect to X2C. For the mapping of the inputs and outputs in HOTINT and X2C strings are used.

Additional notes

HOTINT is the server and X2C Application the client.HOTINT needs synchronisation time (sample time * 1.5) as double value.
Initialization

- X2C builds connection
- X2C sends synchronisation time
- HOTINT sends mapping of inports and outports (see Port Identifier Order)

Initialization of inports and outports

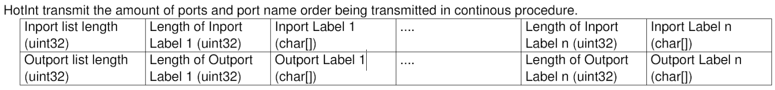
- HOTINT sends (init-) inport values to X2C (see Port Value Exchange)
- X2C Update

Continuous communication

- X2C sends outport values to HOTINT (see Port Value Exchange)
- HOTINT Update
- HOTINT sends inport values to X2C (see Port Value Exchange)
- X2C Update

Closing of Communication

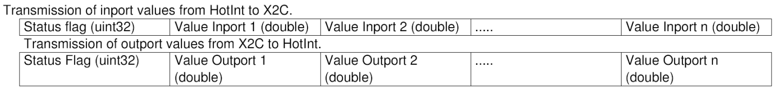
- usually done by server (HOTINT) by setting status flag
- in error case also possible for client by setting status flag



HotInt transmit the amount of ports and port name order being transmitted in continous procedure.

Inport list length (uint32)	Length of Inport Label 1 (uint32)	Inport Label 1 (char[])	....	Length of Inport Label n (uint32)	Inport Label n (char[])
Outport list length (uint32)	Length of Outport Label 1 (uint32)	Outport Label 1 (char[])	....	Length of Outport Label n (uint32)	Outport Label n (char[])

Figure 3.63: IOX2C Port Identifier Order



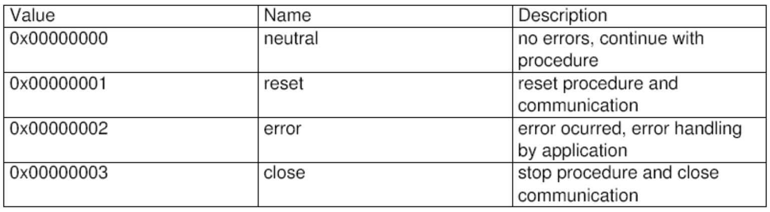
Transmission of inport values from HotInt to X2C.

Status flag (uint32)	Value Inport 1 (double)	Value Inport 2 (double)	.....	Value Inport n (double)

Transmission of outport values from X2C to HotInt.

Status Flag (uint32)	Value Outport 1 (double)	Value Outport 2 (double)	.....	Value Outport n (double)

Figure 3.64: IOX2C Port Value Exchange



Value	Name	Description
0x00000000	neutral	no errors, continue with procedure
0x00000001	reset	reset procedure and communication
0x00000002	error	error ocurred, error handling by application
0x00000003	close	stop procedure and close communication

Figure 3.65: IOX2C Status Flag Values

Data objects of IOX2C:

Data name	type	R	default	description
element_type	string		"IOX2C"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOX2C"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	0	number of inputs
IOBlock. number_of_outputs	integer	**R**	0	number of outputs
IOBlock. number_of_states	integer	**R**	0	number of states
IOBlock. input_element_numbers	vector		[]	vector of element(s) or sensor number(s) connected to input, only valid element numbers permitted!
IOBlock. input_element_types	vector		[]	vector with types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[]	vector with i-th number of output of previous IOelement connected to this element
IOBlock.port_number	integer		50000	Port number, e.g. '50000'.
IOBlock.ip_address	string		"127.0.0.1"	IP address, e.g. '127.0.0.1' (localhost). Do not neglect the dots between the numbers.
IOBlock.timeout	integer		30000	TCP/IP timeout in milliseconds; default is 30000.

IOBlock.input_names	string		""	Names of the inputs, separated with commas. Names have to be consistent with settings in X2C. Order of names has to match vectors specifying input element types. e.g. voltage,current
IOBlock.output_names	string		""	Names of the outputs, separated with commas. Names have to be consistent with settings in X2C. e.g. position,velocity

Observable special values:

For more information see section 3.1

value name	**description**
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock, if available
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock, if available

3.4.21 IOLinearTransducer

Short description

The LinearTransducer realizes an electro-magnetic linear transducer.

Degrees of freedom

The element has 1 degree of freedom, the magnetic flux

Equations

The LinearTransducer computes the force which has to be applied to the mechanical bodies, e.g. Rigid3D. The magnetic flux Ψ is computed as

$$\frac{d\Psi}{dt} = u - Ri\left(z, \Psi\right) \tag{3.42}$$

with displacement z, current i, voltage u and resistance R. The force f of the transducer is a function of displacement and current, $f\left(z, i\right)$, whereas the current is a function of displacement and magnetic flux $i\left(z, \Psi\right)$.To compute these values the LinearTransducer uses radial basis functions (RBF) of the form

$$y\left(\mathbf{x}\right) = \sum_{i=1}^{N} w_i \phi\left(\|\mathbf{x}_{sc} - \mathbf{c}_i\|\right) + \mathbf{v}\begin{bmatrix} 1 \\ \mathbf{x}_{sc} \end{bmatrix} \tag{3.43}$$

with N supporting points (centers) $\mathbf{c}$ and the weights $\mathbf{w}$ and $\mathbf{v}$. The argument $\mathbf{x} = [z, \Psi]$ for the current and $\mathbf{x} = [z, i]$ for the force.The argument $\mathbf{x}$ is scaled with the scaling vector $\mathbf{s}$,

$$x_{sc_i} = x_i / s_i \tag{3.44}$$

Different kernels of the RBF are available:

- RBF kernel = 1: $\phi\left(\|\mathbf{x}_{sc} - \mathbf{c}_i\|\right) = \phi\left(r\right) = r^3$
- RBF kernel = 2: $\phi\left(\|\mathbf{x}_{sc} - \mathbf{c}_i\|\right) = \phi\left(r\right) = r^2\ln(r)$

In the linear case you can use the following simplification for the force

$$\mathbf{v} = [F_0, C_m, k_i] \tag{3.45}$$

$$\mathbf{w} = [] \tag{3.46}$$

$$\mathbf{c} = [] \tag{3.47}$$

$$\mathbf{s} = [1; 1] \tag{3.48}$$

Note: C_m is the destabilizing (negative) magnetic stiffness.The linear case is therefore equal to

$$f(z, i) = F_0 + C_m z + k_i i \tag{3.49}$$

and for the current

$$\mathbf{v} = [-\frac{\Psi_0}{L}, -\frac{k_i}{L}, \frac{1}{L}] \tag{3.50}$$

$$\mathbf{w} = [] \tag{3.51}$$

$$\mathbf{c} = [] \tag{3.52}$$

$$\mathbf{s} = [1; 1] \tag{3.53}$$

which is equal to

$$i(z, \Psi) = -\frac{\Psi_0}{L} - \frac{k_i}{L}z + \frac{1}{L}\Psi \tag{3.54}$$

Inputs and Outputs

- input 1: voltage u in V
- input 2: displacement z in m
- output 1: force f in N
- output 2: negative force $-f$ in N
- output 3: current i in A

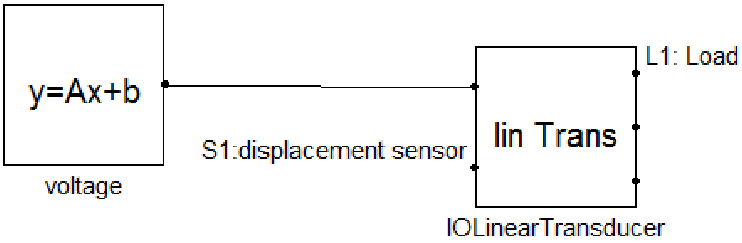


Figure 3.66: IOLinearTransducer

Data objects of IOLinearTransducer:

Data name	**type**	**R**	**default**	**description**
element_type	string		"IOLinearTransducer"	specification of element type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"IOLinearTransducer"	name of the element
element_number	integer	**R**	1	number of the element in the mbs
Graphics				
Graphics.show_connector	bool		1	Flag to draw connector
Graphics.position	vector		[0, 0]	reference drawing position
Graphics.draw_size	vector		[20, 20, 0]	draw size
Graphics.rotation	double		0	rotation: 1==90ř, 2==180ř, 3==270ř, 4=360ř
Graphics. background_color	vector		[-1, -1, -1]	background color; -1=transparent
Graphics. foreground_color	vector		[0, 0, 0]	foreground color
Graphics. input_nodes_num	vector		[]	number of input of drawing position "input_nodes"
Graphics.input_nodes	matrix		[]	
IOBlock				
IOBlock. number_of_inputs	integer	**R**	2	number of inputs
IOBlock. number_of_outputs	integer	**R**	3	number of outputs
IOBlock. number_of_states	integer	**R**	1	number of states
IOBlock. input_element_numbers	vector		[2, 3]	numbers of IOElement or sensor providing the inputs [voltage, displacement]
IOBlock. input_element_types	vector		[1, 1]	types of connected inputs; 1=IOElement, 2=Sensor
IOBlock. input_local_number	vector		[1, 1]	i-th number of output of previous IOelement connected to this element
Physics				
Physics.resistance	double		0.6	electrical resistance in Ohm
Physics.RBF_Force				
Physics.RBF_Force. scaling_vector	vector		[1, 1]	scaling of displacement, x(1), and current, x(2): x_scaled(i) = x(i)/scaling_vector(i)
Physics.RBF_Force. centers	matrix		[]	c(i) is the scaled supporting point
Physics.RBF_Force. weights_RBF	vector		[]	w(i) is the weight of the supporting point c(i)
Physics.RBF_Force. weights_poly	vector		[0, 1.4e+004, 14]	v(i), weights of the polynomial v*[1; x(1); x(2)], in linear case: v = [F0, Cm, ki]
Physics.RBF_Force. RBF_kernel	integer		1	kernel of the RBF: 1..r*r*r, 2..r*r*ln(r)
Physics.RBF_Current				
Physics.RBF_Current. scaling_vector	vector		[1, 1]	scaling of displacement, x(1), and current, x(2): x_scaled(i) = x(i)/scaling_vector(i)
Physics.RBF_Current. centers	matrix		[]	c(i) is the scaled supporting point
Physics.RBF_Current. weights_RBF	vector		[]	w(i) is the weight of the supporting point c(i)
Physics.RBF_Current. weights_poly	vector		[0, -1e+003, 71.4]	v(i), weights of the polynomial v*[1; x(1); x(2)], in linear case: v = [-Psi0/L, -ki/L, 1/L]

Physics.RBF_Current. RBF_kernel	integer		1	kernel of the RBF: 1..r*r*r, 2..r*r*ln(r)

Observable special values:

For more information see section 3.1

value name	description
Internal.DOF	degrees of freedom (or generalized unknowns) of the element. range: 1-1
Internal.first_order_variable	first order variables of the element. range: 1-1
IOBlock.output	IOBlock.output[i] ... measures the i-th output of this IOBlock
IOBlock.input	IOBlock.input[i] ... access to the i-th input of this IOBlock

Example

see file IOLinearTransducer.txt

```
volt.element_type= "IOLinearTransformation"
volt.IOBlock.A_matrix= []
volt.IOBlock.b_vector= [10] % constant voltage of 10 V
nEVolt = AddElement(volt)

rigidbody.element_type= "Mass1D"
rigidbody.Physics.mass= 0.315     % add 1 body
nEBody = AddElement(rigidbody)

elemSet.set_type = "ElementSet"
elemSet.element_numbers= [nEBody]   % set with 1 body
nESetTilgerMoving = AddSet(elemSet)

spring.element_type= "CoordinateConstraint"   % spring w.r.t. ground
spring.Physics.use_penalty_formulation= 1     % spring has to be sufficient stiff!
spring.Physics.Penalty.damping= 15       % damping coefficient Dp for viscous damping
spring.Physics.Penalty.spring_stiffness= 20000  % general or penalty stiffness parameter Sp
spring.Coordinate1.element_number= nEBody
spring.Coordinate1.local_coordinate= 1
AddConnector(spring)

sens.sensor_type= "FVElementSensor"
sens.element_number= nEBody
sens.field_variable= "displacement"       % displacement is needed as input
nSensDisp = AddSensor(sens)

elektroMagnet
{
  element_type= "IOLinearTransducer"
  Graphics.position= [100, 0]
  IOBlock.input_element_numbers= [nEVolt,nSensDisp] % inputs [voltage, displacement]
  IOBlock.input_element_types= [1, 2]         % types; 1=IOElement, 2=Sensor
}
nELinTransducer = AddElement(elektroMagnet)

force
{
  load_type= "GCLoad"
  load_function_type= 2           %time dependency of the load: 2..IOElement
  IOElement.input_element_number= nELinTransducer  %number of IOElement in the mbs
  IOElement.input_local_number= 1  %number of output of IOElement connected to this element
}
nLTilger = AddLoad(force)         % add load to mbs
AssignLoad(nESetTilgerMoving,nLTilger)    % assign load to element

sensEl
{
  name= "IO voltage"
  sensor_type= "ElementSensor"
  element_number= nELinTransducer
  value= "IOBlock.input[1]"
}
AddSensor(sensEl)

sensEl.name= "IO displacement"
sensEl.value= "IOBlock.input[2]"
AddSensor(sensEl)
sensEl.name= "IO force"
sensEl.value= "IOBlock.output[1]"
AddSensor(sensEl)
sensEl.name= "IO current"
sensEl.value= "IOBlock.output[3]"
AddSensor(sensEl)
```

3.5 Material

These materials are available:

- Material, 3.5.1

Note:
In HOTINT several classes are treated as 'material'. BeamProperties are also 'materials', and can therefore be edited and deleted in the GUI with the menu items of the materials.
In the script language the command `AddMaterial` is just available for the materials in the list above.

3.5.1 Material

Short description

Material is the basic Object for defining material properties for standard finite elements (in contrast to structural finite elements such as beams and plates).

Additional notes

For static problems define the elastic properties `Solid.youngs_modulus` and `Solid.poisson_ratio`, whereas for dynamic problems also `Solid.density` is required. If the problem is planar (`Solid.plane` is set to 1), then the plane strain case is assumed unless `Solid.plane_stress` is set to 1. If the material is inelastic, then also the properties in the subtree `Inelasticity` have to be set.

Data objects of Material:

Data name	**type**	**R**	**default**	**description**
Material_number	integer	**R**	2	
material_type	string		"Material"	specification of material type. Once the material is added to the mbs, you MUST NOT change this type anymore!
Graphics				
Graphics.color	vector		[0, 0, 1]	material color (as yet used with FEMesh, only)
name	string		"Material"	name of the material
Solid				
Solid.density	double		0	density (rho) for gravitational force
Solid.youngs_modulus	double		0	Youngs modulus
Solid.poisson_ratio	double		0	Poisson ratio
Solid.plane	bool		0	true: 2D, false: 3D
Solid.plane_stress	bool		0	for 2D-Elements only; 1: plane stress, 0: plane strain
Inelasticity				
Inelasticity.yield_stress	double		0	Yield Stress s_y, e.g., |dev s| <= s_y
Inelasticity.tangent_module	double		0	Modulus of hardening H
Inelasticity.inelasticity_type	string		"linear_elastic"	linear_elastic, elasto_plastic (= Prandtl Reuss plasticity + isotropic hardening), nonlinear_elastic_Simo_Hughes (see Simo and Hughes, Computational Inelasticity 1998: S=lambda/2*(J*J-1)/C + mu*(1-1/C))
Inelasticity.inelasticity_solution_method	string		"default"	fixed_point, return_mapping, consistent_tangent_stiffness (see Simo and Hughes, Computational Inelasticity 1998)

Example

see file Material.txt

```
my_material
{
  material_type= "Material"
  Solid.density = 7800
  Solid.youngs_modulus = 2e10
}
nMaterial = AddMaterial(my_material)
```

3.6 BeamProperties

These beam properties are available:

- Beam3DProperties, 3.6.1

Note:
In HOTINT several classes are treated as 'material'. BeamProperties are also 'materials', and can therefore be edited and deleted in the GUI with the menu items of the materials.
In the script language the command `AddBeamProperties` has to be used for the beam properties in the list above.

3.6.1 Beam3DProperties

Short description

Beam3DProperties defines material and geometric properties for beam structural finite elements.

Additional notes

First, specify the `cross_section_type` of the beam, which may be either rectangular (if set to 1), circular (if set to 2) or polygonal (if set to 3). In either case the `cross_section_size` is a vector of 2, 1, or $2\,n$ entries, where n confers to the number of vertices of a closed polygon. Then specify the stiffnesses and moments of inertias, as they are neede by your beam and problem.

Data objects of Beam3DProperties:

Data name	type	R	default	description
Material_number	integer	**R**	2	
material_type	string		"Beam3DProperties"	specification of material type. Once the material is added to the mbs, you MUST NOT change this type anymore!
Graphics				
Graphics.color	vector		[0, 0, 1]	material color (as yet used with FEMesh, only)
name	string		"Beam3DProperties"	name of the material
Inelasticity				
Inelasticity.inelasticity_type	string		"linear_elastic"	linear_elastic, elasto_plastic (= Prandtl Reuss plasticity + isotropic hardening), nonlinear_elastic_Simo_Hughes (see Simo and Hughes, Computational Inelasticity 1998: S=lambda/2*(J*J-1)/C + mu*(1-1/C))
Inelasticity.inelasticity_solution_method	string		"default"	fixed_point, return_mapping, consistent_tangent_stiffness (see Simo and Hughes, Computational Inelasticity 1998)
cross_section_type	integer		1	1: rectangular, 2: circular, 3: tubular, 4: polygonal

cross_section_size	vector		[0, 0]	vector length of cross_section_size depends on cross_section_type: length 1 for circular cross section (radius), length 2 for rectangular cross section (y and z extension) or tubular cross section (outer and inner diameter) , and length 2*n for polygonal cross section (p1y,p1z,p2y,p2z,...,pny,pnz)
EA	double		0	youngs modulus * area
EIy	double		0	bending stiffness w.r.t. y-axis (2D-beam)
EIz	double		0	bending stiffness w.r.t. z-axis
GAky	double		0	shear stiffness including shear correction factor ky (2D-beam)
GAkz	double		0	shear stiffness including shear correction factor kz
GJkx	double		0	torsional stiffness including shear correction factor kx
RhoA	double		0	density * area
RhoIx	double		0	density * second area of moment w.r.t. x-axis
RhoIy	double		0	density * second area of moment w.r.t. y-axis (2D-beam)
RhoIz	double		0	density * second area of moment w.r.t. z-axis
density	double		0	density (rho) for gravitational force

Example

see file Beam3DProperties.txt

```
bp
{
  material_type= "Beam3DProperties"
  cross_section_type= 1
  cross_section_size= [0.1, 0.1]
  EA= 2100000000
  EIy= 1750000
  EIz= 1750000
  GJkx= 2692307.692307693
}
nBeamProperties = AddBeamProperties(bp)
```

3.7 Node

These nodes are available:

- Node3DS1rot1, 3.7.1
- Node3DS2S3, 3.7.2
- Node3DRxyz, 3.7.3
- Node3DR123, 3.7.4
- Node3DS1S2, 3.7.5
- Node3DThermoMechanic, 3.7.6
- Node3DThermo, 3.7.7

Note:
In HOTINT different types of nodes exist. The main difference between these types are the number of degrees of freedom. Depending on the chosen type of an element, the correct node has to be used. Each element provides some information about the needed nodes.
In the script language the command `AddNode` is used for adding a node to the system.

3.7.1 Node3DS1rot1

Short description

Node3DS1rot1 is a finite element node for elements in 3D, and provides 7 degrees of freedom.

Degrees of freedom

This node provides 7 degrees of freedom: the first 3 degrees of freedom are the displacement $(q_1, q_2, q_3)^T = \mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the next 3 DOFs denote the change of the *first slope*, which is the partial derivative of the position $(q_4, q_5, q_6)^T = \mathbf{r}_{,\xi} - \mathbf{r}_{0,\xi}$ with ξ denoting the first of the three coordinates (ξ, η, ζ) of the reference element, and the 7th degree of freedom is the local rotation $q_7 = \theta$ of the node around its current direction $S1$.

Geometry

The reference geometry of the node is defined by the user via (a) `Geometry.reference_position` and (b) the rotation `Geometry.reference_rot_angles`. The rotation is prescribed by the user in form of kardan angles (initially, local (S_1, S_2, S_3) and global frame (x, y, z) are identical, then rotate local frame around $S1$, then $S2$ and finally $S3$). The current position is evaluated by adding displacement (the first three degrees of freedom) to the reference position of the node (degrees of freedom: $(q_1, q_2, q_3)^T$), and the current rotation of the node is obtained by adding the change of the first axis of the local frame (DOFs: $(q_4, q_5, q_6)^T$) to the first axis of the local frame in reference configuration of the node, and finally rotating the two other axes around the first axis of the local frame by the amount of the 7th degree of freedom $q_7 = \theta$.

Data objects of Node3DS1rot1:

Data name	type	R	default	description

node_type	string		"Node3DS1rot1"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DS1rot1"	Node identifier.
node_number	integer		1	Node Number.
Geometry				
Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_rot_angles	vector		[0, 0, 0]	Kardan rotation angles (X,Y,Z) in rad in global frame of node in reference configuration.
Initialization				
Initialization. node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

Example

see file Node3DS1rot1.txt

```
node
{
  node_type= "Node3DS1rot1"
  Geometry
  {
    reference_position= [0, 0, 0]
    reference_rot_angles= [0, 0, 0]
  }
}
nNode1 = AddNode(node)
```

3.7.2 Node3DS2S3

Short description

Node3DS2S3 is a finite element node for elements in 3D, and provides 9 degrees of freedom.

Degrees of freedom

This node provides 9 degrees of freedom: the first 3 degrees of freedom are the displacement $(q_1, q_2, q_3)^T = \mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the next 3 DOFs denote the change of the *second slope*, which are the partial derivatives of the position $(q_4, q_5, q_6)^T = \mathbf{r}_{,\eta} - \mathbf{r}_{0,\eta}$ and $(q_7, q_8, q_9)^T = \mathbf{r}_{,\zeta} - \mathbf{r}_{0,zeta}$, where η and ζ denote the second and third of the three coordinates (ξ, η, ζ) of the reference element.

Geometry

The reference geometry of the node is defined by the user via (a) `Geometry.reference_position` and (b) the slopes `Geometry.ref_slope2` and `Geometry.ref_slope3`. The current position is evaluated by adding the displacement (the first three degrees of freedom $(q_1, q_2, q_3)^T$) to the reference position of the node, and further the current slopes of the node are obtained by adding the change of the second and third slopes (DOFs: $(q_4, q_5, q_6)^T$ and $(q_7, q_8, q_9)^T$) to the second and third slopes in reference configuration of the node.

Data objects of Node3DS2S3:

Data name	type	R	default	description
node_type	string		"Node3DS2S3"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DS2S3"	Node identifier.
node_number	integer		1	Node Number.
Geometry				
Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_slope2	vector		[0, 1, 0]	slope 2 of node in reference configuration.
Geometry. reference_slope3	vector		[0, 0, 1]	slope 3 of node in reference configuration.
Initialization				
Initialization. node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.7.3 Node3DRxyz

Short description

Node3DRxyz is a finite element node in 3D. It has a 3D reference position, a reference orientation described by bryant angles and 6 degrees of freedom.

Degrees of freedom

The first 3 degrees of freedom are used to describe the displacement $(q_1, q_2, q_3)^T = \mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the last 3 are used for the description of linearized (small) angles $(\phi_x, \phi_y, \phi_z)^T$. All degrees of freedom are w.r.t. the global coordinate system.

Geometry

The reference position of the node is defined by the user via `Geometry.reference_position` and the reference orientation via `Geometry.reference_rot_angles`. The current position is evaluated by adding the displacement (the first three degrees of freedom $(q_1, q_2, q_3)^T$) to the reference position of the node.

Data objects of Node3DRxyz:

Data name	type	R	default	description
node_type	string		"Node3DRxyz"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DRxyz"	Node identifier.
node_number	integer		1	Node Number.

Geometry				
Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_rot_angles	vector		[0, 0, 0]	Kardan rotation angles (X,Y,Z) in rad in global frame of node in reference configuration.
Initialization				
Initialization. node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.7.4 Node3DR123

Short description

Node3DR123 is a finite element node in 3D. It has a 3D reference position, a reference orientation described by bryant angles and 6 degrees of freedom.

Degrees of freedom

The first 3 degrees of freedom are used to describe the displacement $(q_1, q_2, q_3)^T = \mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the last 3 are used for the description of linearized (small) angles $(\phi_x, \phi_y, \phi_z)^T$. All degrees of freedom are w.r.t. the reference coordinate system of the node.

Geometry

The reference position of the node is defined by the user via `Geometry.reference_position` and the orientation via `Geometry.reference_rot_angles`. The current position is evaluated by adding the displacement (the first three degrees of freedom $(q_1, q_2, q_3)^T$ transformed into the global coordinate system) to the reference position of the node.

Data objects of Node3DR123:

Data name	type	R	default	description
node_type	string		"Node3DR123"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DR123"	Node identifier.
node_number	integer		1	Node Number.
Geometry				
Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_rot_angles	vector		[0, 0, 0]	Kardan rotation angles (X,Y,Z) in rad in global frame of node in reference configuration.
Initialization				
Initialization. node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.7.5 Node3DS1S2

Short description

Node3DS1S2 is a finite element node for elements in 3D, and provides 9 degrees of freedom.

Degrees of freedom

This node provides 9 degrees of freedom: the first 3 degrees of freedom are the displacement $(q_1, q_2, q_3)^T = \mathbf{u} = \mathbf{r} - \mathbf{r}_0$, the next 3 DOFs denote the change of the *first slope*, which are the partial derivatives of the position $(q_4, q_5, q_6)^T = \mathbf{r}_{,\xi} - \mathbf{r}_{0,\xi}$ and $(q_7, q_8, q_9)^T = \mathbf{r}_{,\eta} - \mathbf{r}_{0,eta}$, where ξ and η denote the first and second of the three coordinates (ξ, η, ζ) of the reference element.

Geometry

The reference geometry of the node is defined by the user via (a) `Geometry.reference_position` and (b) the slopes `Geometry.ref_slope1` and `Geometry.ref_slope2`. The current position is evaluated by adding the displacement (the first three degrees of freedom $(q_1, q_2, q_3)^T$) to the reference position of the node, and further the current slopes of the node are obtained by adding the change of the first and second slopes (DOFs: $(q_4, q_5, q_6)^T$ and $(q_7, q_8, q_9)^T$) to the first and second slopes in reference configuration of the node.

Data objects of Node3DS1S2:

Data name	type	R	default	description
node_type	string		"Node3DS1S2"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DS1S2"	Node identifier.
node_number	integer		1	Node Number.
Geometry				
Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_slope1	vector		[1, 0, 0]	slope 1 of node in reference configuration.
Geometry. reference_slope2	vector		[0, 1, 0]	slope 2 of node in reference configuration.
Initialization				
Initialization. node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.7.6 Node3DThermoMechanic

Short description

Node3DThermoMechanic is the basic finite element node in 3D with an additional degree of freedom for temperature. It owns a reference position in 3D, and 3 degrees of freedom resembling the displacement in 3D. Additionally, it owns a scalar degree of freedom.

Degrees of freedom

This node provides four degrees of freedom, three components of the displacement vector $\mathbf{u} = (q_1, q_2, q_3)^T$ measured in the global frame of the multibody system. The fourth degree of freedom defines the scalar temperature T

Geometry

The geometry of the node is defined by its current position $\mathbf{r}$ measured in the global frame of the multibody system, which is the sum of the user defined reference position $\mathbf{r}_0$ and the displacement vector $\mathbf{u}$, which is composed of the nodal degrees of freedom. Addionally, the scalar degree of freedom defines the temperature as a scalar unknown.

Data objects of Node3DThermoMechanic:

Data name	**type**	**R**	**default**	**description**
node_type	string		"Node3DThermoMechanic"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DThermoMechanic"	Node identifier.
node_number	integer		1	Node Number.
Geometry				
Geometry.reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Initialization				
Initialization.node_initial_values	vector		[0, 0, 0, 0, 0, 0, 0, 0]	initial values for all degrees of freedom of node
Graphics				
Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.7.7 Node3DThermo

Short description

Node3DThermo is the basic finite element node in 3D with an additional degree of freedom for temperature. It owns a reference position and velocity in 3D and one scalar degree of freedom.

Degrees of freedom

This node provides one degree of freedom namely the temperature T

Geometry

The geometry of the node is defined by its current position $\mathbf{r}$ measured in the global frame of the multibody system.

Data objects of Node3DThermo:

Data name	**type**	**R**	**default**	**description**

node_type	string		"Node3DThermo"	specification of node type. Once the node is added to the mbs, you MUST NOT change this type anymore!
name	string		"Node3DThermoMechanic"	Node identifier.
node_number	integer		1	Node Number.

Geometry

Geometry. reference_position	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.
Geometry. reference_velocity	vector		[0, 0, 0]	Position (2D/3D) in reference configuration.

Initialization

Initialization. node_initial_values	vector		[0, 0]	initial values for all degrees of freedom of node

Graphics

Graphics.RGB_color	vector		[0.4, 0.4, 0.1]	[red, green, blue] color of element, range = 0..1,
Graphics.visible	integer		1	Visibility of node.

3.8 Load

These loads are available:

- GCLoad, 3.8.1
- BodyLoad, 3.8.2
- ForceVector2D, 3.8.3
- ForceVector3D, 3.8.4
- MomentVector3D, 3.8.5
- Gravity, 3.8.6
- SurfacePressure, 3.8.7

For all loads it is possible to vary the value of the load with respect to time. The following options are available:

1. MathFunction
2. IOElement

3.8.0.1 MathFunction

The value $F(t)$ of a load at time t is computed as:

$$F(t) = f(t)\vec{F} \tag{3.55}$$

$f(t)$ represents the value of the MathFunction at time t, e.g. $f(t) = sin(t)$.
$\vec{F}$ represents the (constant) force vector, if a force vector is used in the specific type of load, e.g. ForceVector3D.
If no force vector is available for the load, then the load is defined by $f(t)$ only. Any additional scalar value (e.g. load_value in GCLoad) is set to 1!

3.8.0.2 IOElement

The value $F(t)$ of a load at time t is computed as:

$$F(t) = f(t)\vec{F} \tag{3.56}$$

$f(t)$ represents the value of the output of the IOElement at time t. By the use of IOElements it is possible to define loads, that are not only dependent on time, but on any possible input of an IOElement.
$\vec{F}$ represents the (constant) force vector, if a force vector is used in the specific type of load, e.g. ForceVector3D.
If no force vector is available for the load, then the load is defined by $f(t)$ only. Any additional scalar value (e.g. load_value in GCLoad) is set to 1!

3.8.1 GCLoad

A load acting on a generalized coordinate (gc) of the element.

Data objects of GCLoad:

Data name	**type**	**R**	**default**	**description**
name	string		"Load"	name of the load
load_type	string		"GCLoad"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
generalized_coordinate	integer		1	(local) number of the generalized coordinate
load_value	double		0	value of the load acting in the direction of generalized_coordinate
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

Example

see file GCLoad.txt

```
myLoad    % define the load
{
  load_type = "GCLoad"
  generalized_coordinate = 1  %(local) number of the generalized coordinate
  load_value = 10
}
nLoad=AddLoad(myLoad)

emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)
```

3.8.2 BodyLoad

The load value is integrated over the volume of the body and applied to the body in the specified direction. For the case of a rigid body, a force of size load_value = density*gravity_constant applies a force according to the gravitational force.

Data objects of BodyLoad:

Data name	**type**	**R**	**default**	**description**
name	string		"Load"	name of the load
load_type	string		"BodyLoad"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
direction	integer		1	direction of the load
load_value	double		0	value of the load acting
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

3.8.3 ForceVector2D

Data objects of ForceVector2D:

Data name	**type**	**R**	**default**	**description**
name	string		"Load"	name of the load
load_type	string		"ForceVector2D"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
force_vector	vector		[0, 0]	defines the magnitude and direction of the force
position	vector		[0, 0]	(local) position where the force is applied to the element
local_force	integer		0	flag which describes, if local or global coordinate system is used: 1 = force in local body coordinate system, 0 = global force
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement

MathFunction

MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'

IOElement

IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

Example

see file ForceVector2D.txt

```
myLoad    % define the load
{
  load_type = "ForceVector2D"
  force_vector = [10,0] % magnitude and direction
}
nLoad=AddLoad(myLoad)

L_x = 0.10 % length
L_y = 0.20 % width
L_z = 0.01 % height (for drawing and computation of mass)
density= 7850

myRigid2D % add rigid body
{
  element_type= "Rigid2D"  %specification of element type.
  name= "R2D"  %name of the element
  Graphics.body_dimensions = [L_x, L_y, 0]
  loads = [nLoad] % add the load to the element
   Physics
  {
    mass= density*L_x*L_y*L_z
    moment_of_inertia= 1.0/12.0*mass*(L_x^2+L_y^2)
  }
  Initialization
  {
    initial_position= [0, 0]  %[X, Y]
    initial_rotation= [0.0]  % rot1_Z in rad
    initial_velocity= [0, 0]  %[X, Y]
    initial_angular_velocity= [0] %rad/s
  }
}
nElement = AddElement(myRigid2D)
```

3.8.4 ForceVector3D

A load acting on an element at a specified (local) position.

Data objects of ForceVector3D:

Data name	type	R	default	description
name	string		"Load"	name of the load
load_type	string		"ForceVector3D"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
force_vector	vector		[0, 0, 0]	defines the magnitude and direction of the force
position	vector		[0, 0, 0]	(local) position where the force is applied to the element
local_force	integer		0	flag which describes, if local or global coordinate system is used: 1 = force in local body coordinate system, 0 = global force
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

Example

see file ForceVector3D.txt

```
myLoad    % define the load
{
  load_type = "ForceVector3D"
  force_vector = [10,0,0] % magnitude and direction
}
nLoad=AddLoad(myLoad)
```

```
emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)

ViewingOptions.Loads.show_loads = 1
ViewingOptions.Loads.arrow_size = 0.2
```

3.8.5 MomentVector3D

A torque acting on an element at a specified (local) position.

Data objects of MomentVector3D:

Data name	**type**	**R**	**default**	**description**
name	string		"Load"	name of the load
load_type	string		"MomentVector3D"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
moment_vector	vector		[0, 0, 0]	defines the magnitude and direction of the moment
position	vector		[0, 0, 0]	(local) position where the moment is applied to the element
local_moment	integer		0	flag which describes, if local or global coordinate system is used: 1 = moment in local body coordinate system, 0 = global moment
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

3.8.6 Gravity

The load is integrated over the volume of the body and applied to the body in the specified direction. The density of the body is used to compute the force.

Data objects of Gravity:

Data name	type	R	default	description
name	string		"Load"	name of the load
load_type	string		"Gravity"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
direction	integer		1	global direction of the gravity
gravity_constant	double		9.81	use negative sign if necessary
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

Example

see file Gravity.txt

```
myLoad    % define the load
{
  load_type = "Gravity"
  name = "gravity for all elements"
  direction = 2
  gravity_constant = 9.81
}
nLoad=AddLoad(myLoad)

emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)

ViewingOptions.Loads.show_loads = 1
ViewingOptions.Loads.arrow_size = 0.2
```

3.8.7 SurfacePressure

Data objects of SurfacePressure:

Data name	**type**	**R**	**default**	**description**
name	string		"Load"	name of the load
load_type	string		"SurfacePressure"	specification of load type. Once the load is added to the mbs, you MUST NOT change this type anymore!
load_number	integer	**R**	1	number of the load in the mbs
direction	integer		1	local surface (inner/outer)
surface_pressure	double		0	use negative sign if necessary
load_function_type	integer		0	time dependency of the load: 0..constant, 1..MathFunction, 2..IOElement
MathFunction				
MathFunction. piecewise_mode	integer		-1	modus for piecewise interpolation: -1=not piecewise, 0=constant, 1=linear, 2=quadratic
MathFunction. piecewise_points	vector		[]	supporting points (e.g. time or place) for piecewise interpolation
MathFunction. piecewise_values	vector		[]	values at supporting points
MathFunction. piecewise_diff_values	vector		[]	differential values at supporting points - for quadratic interpolation
MathFunction. parsed_function	string		""	string representing parsed function, e.g. 'A*sin(omega*t)'
MathFunction. parsed_function_parameter	string		""	string representing parameter of parsed function, e.g. 't'
IOElement				
IOElement. input_element_number	integer		0	number of IOElement in the mbs
IOElement. input_local_number	integer		0	number of output of IOElement connected to this element

Example

see file SurfacePressure.txt

3.9 Sensor

These sensors are available:

- FVElementSensor, 3.9.1
- ElementSensor, 3.9.2
- LoadSensor, 3.9.3
- MultipleSensor, 3.9.4
- SystemSensor, 3.9.5

In HOTINT it is possible to access all degrees of freedom and many more interesting values with sensors. In general these values are stored to a file at specified time steps. Many options concerning these settings are available in SolverOptions.Solution.
You can use the PlotTool to visualize the sensor values but it is also possible to import the solution file easily in other software for postprocessing.
If you want to modify sensor values online (e.g. convert the units from rad to degrees or subtract an offset) it is recommended to use ControlElements.
In the script language the command `AddSensor` is used to add a sensor to the system.

3.9.1 FVElementSensor

The FieldVariableElementSensor evaluates the value of a field variable at a specified position. There are two possibilities to define this position:

- element number + local position
- element number + local node number

The descriptions of the elements above include a list of available field variables for each element. Possible field variables are e.g.

- position, velocity and displacement
- bryant_angle and angular_velocity
- beam_axial_extension, beam_torsion, beam_curvature
- many more

Data objects of FVElementSensor:

Data name	**type**	**R**	**default**	**description**
sensor_number	integer	**R**	1	number of the sensor in the mbs
name	string		"sensor"	name of the sensor for the output files and for the plot tool
sensor_type	string		"FVElementSensor"	specification of sensor type. Once the sensor is added to the mbs, you MUST NOT change this type anymore!
element_number	integer		1	number of the element, to which the sensor is applied

node_number	integer		0	local node number. If > 0, then the position of this node is used.
local_position	vector		[0, 0, 0]	local position at which the field variable is evaluated.
field_variable	string		"position"	name of the field variable, e.g. 'position', see the documentation of the elements for the available field variables
component	string		"x"	component of the field variable, e.g. 'x'

Example

see file FVElementSensor.txt

```
emptyMass3D
{
  element_type = "Mass3D"
  Physics.mass= 1
}
nElement = AddElement(emptyMass3D)

sensor
{
  sensor_type= "FVElementSensor"
  element_number= nElement    %number of the element
  field_variable= "position"  %name of the field variable
  component= "x"         %component of the field variable
}
nSensor = AddSensor(sensor)
```

3.9.2 ElementSensor

The ElementSensor returns special values evaluated in the element. It can be used e.g. for measuring a specific degree of freedom of an element. The descriptions of the elements above include a list of available special values for each element.

Data objects of ElementSensor:

Data name	**type**	**R**	**default**	**description**
sensor_number	integer	**R**	1	number of the sensor in the mbs
name	string		"sensor"	name of the sensor for the output files and for the plot tool
sensor_type	string		"ElementSensor"	specification of sensor type. Once the sensor is added to the mbs, you MUST NOT change this type anymore!
element_number	integer		1	number of the element, to which the sensor is applied
value	string		""	special value of the element, use "[]" to access vector or matrix values, e.g. force[1] or stress[2,3]

Example

see file ElementSensor.txt

```
emptyMass3D
{
  element_type = "Mass3D"
  Physics.mass= 1
}
nElement = AddElement(emptyMass3D)

ElemSensor
{
  sensor_type= "ElementSensor"
  element_number= nElement
  value= "Internal.second_order_variable[1]"
}
nElemSensor = AddSensor(ElemSensor)
```

3.9.3 LoadSensor

The LoadSensor can be applied to loads in order to measure their time dependency. The value $F(t)$ of a load at time t is computed (see the description of the loads for more details) as:

$$F(t) = f(t)\vec{F} \tag{3.57}$$

The LoadSensor returns the value of the factor $f(t)$ and not the value $F(t)$. If the LoadSensor is used for a scalar load (e.g. GCLoad), then $f(t)$ and $F(t)$ are equal. If the LoadSensor is used for a load vector (e.g. ForceVector3D) then $f(t)$ and $F(t)$ may not be equal.
The LoadSensor can not be shown in the graphical output, because the load does not have a position by itself and may be applied to several elements or nodes.

Data objects of LoadSensor:

Data name	**type**	**R**	**default**	**description**
sensor_number	integer	**R**	1	number of the sensor in the mbs
name	string		"sensor"	name of the sensor for the output files and for the plot tool
sensor_type	string		"LoadSensor"	specification of sensor type. Once the sensor is added to the mbs, you MUST NOT change this type anymore!
load_number	integer		1	number of the load, to which the sensor is applied

Example

see file LoadSensor.txt

```
myLoad    % define the load
{
  load_type = "ForceVector3D"
  force_vector = [10,0,0] % magnitude and direction
  load_function_type = 1  % time dependent load
  MathFunction
  {
    piecewise_mode= -1  %modus -1=not piecewise
  parsed_function= "sin(100*t)"  %string representing parsed function
  parsed_function_parameter= "t"  % parameter of parsed function
  }
}
nLoad=AddLoad(myLoad)

emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)

sensor
{
  sensor_type= "LoadSensor"
  load_number= nLoad  %number of the load
}
nSensor = AddSensor(sensor)
```

3.9.4 MultipleSensor

The MultipleSensor applies mathematical operations to a list of sensors. The sensor can be used, e.g. to get the maximum or average value of a list of sensors. The following mathematical operations are possible (use these words for 'operation'):

- average
- minimum
- maximum
- sum
- norm
- norm2

If weights are used, then the value of each sensor is multiplied with the weight before the mathematical operation is performed. To compute a weighted sum of the first 4 sensors, the entries would be e.g. sensor_numbers = [1,2,3,4] and weights = [0.125,0.125,0.25,0.5].

Data objects of MultipleSensor:

Data name	**type**	**R**	**default**	**description**
sensor_number	integer	**R**	1	number of the sensor in the mbs
name	string		"sensor"	name of the sensor for the output files and for the plot tool

sensor_type	string		"MultipleSensor"	specification of sensor type. Once the sensor is added to the mbs, you MUST NOT change this type anymore!
sensor_numbers	vector		[]	number of the sensors, that are used for computation
weights	vector		[]	weights for e.g. a weighted sum. This vector must have the same length as sensor_numbers or must be empty!
operation	string		"maximum"	mathematical operation that is applied to the sensor values, e.g. 'maximum','average',...

3.9.5 SystemSensor

The SystemSensor can be applied to global degrees of freedom, eigenvalues, several iteration numbers or performance indicators. It returns the value of the specified quantity at time t, and can not be shown in the graphical output, because a system quantity does in general not have a position by itself.

Data objects of SystemSensor:

Data name	**type**	**R**	**default**	**description**
sensor_number	integer	**R**	1	number of the sensor in the mbs
name	string		"Systemsensor"	name of the sensor for the output files and for the plot tool
sensor_type	string		"SystemSensor"	specification of sensor type. Once the sensor is added to the mbs, you MUST NOT change this type anymore!
object	string		"none"	Object tracked by systemsensor. Is either 'DOF' (global degree of freedom), 'EV' (global eigenvalue), 'jacobians', 'newton_iterations', 'newton_iterations_total', 'discontinuous_iterations', 'rhs_evaluations', 'rhs_evaluations_jacobian', 'volume', 'potential_energy', 'kinetic_energy', 'FE_color_minimum', 'FE_color_maximum', 'NNodes' or 'NElements'
global_index	integer		0	Number of the global index. Has to be set if (and only if) object is 'DOF' or 'EV'.

Example

see file SystemSensor.txt

```
myLoad    % define the load
{
  load_type = "ForceVector3D"
  force_vector = [10,0,0] % magnitude and direction
  load_function_type = 1  % time dependent load
  MathFunction
  {
    piecewise_mode= -1  %modus -1=not piecewise
  parsed_function= "sin(100*t)"  %string representing parsed function
  parsed_function_parameter= "t"  % parameter of parsed function
  }
}
nLoad=AddLoad(myLoad)

emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)

Systemsensor
{
  name= "Systemsensor Jacobians"  %name of the sensor for the output files and for the plot too
  sensor_type= "SystemSensor"  %specification of sensor type. Once the sensor is added to the m
  object= "jacobians"  %Object tracked by systemsensor. Is either 'DOF' (global degree of freed
  global_index= 0  %Number of the global index. Has to be set if (and only if) object is 'DOF'
}
AddSensor(Systemsensor)

Systemsensor.object= "rhs_evaluations"
Systemsensor.name= "Systemsensor RHS Evaluations"
AddSensor(Systemsensor)

Systemsensor.object= "DOF"
Systemsensor.global_index= 4
Systemsensor.name= "Systemsensor DOF 4"
AddSensor(Systemsensor)
```

3.10 GeomElement

These GeomElements are available:

- GeomMesh3D, 3.10.1
- GeomCylinder3D, 3.10.2
- GeomSphere3D, 3.10.3
- GeomCube3D, 3.10.4
- GeomOrthoCube3D, 3.10.5

GeomElements are used in HOTINT to improve the appearance of your simulation model. GeomElements do not have any physical meaning in HOTINT and have to be attached to the ground or some (real) reference element. The GeomElement will move with this reference element.

GeomElements are also a good tool to define surfaces e.g. used for coupled simulations with fluid-structure interaction.

In the script language the command `AddGeomElement` is used to add GeomElements to the system.

3.10.1 GeomMesh3D

Data objects of GeomMesh3D:

Data name	**type**	**R**	**default**	**description**
geom_element_type	string		"GeomMesh3D"	specification of GeomElement type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"GeomElement"	name of the GeomElement
reference_element_number	integer		0	0 ... ground, otherwise insert number of existing element
Graphics				
Graphics.RGB_color	vector		[0.2, 0.2, 0.8]	[red, green, blue], range = 0..1
Graphics.transparency	double		-1	transparency [0..1], 0=transparent, 1=solid, set -1 if global transparency is used
Graphics.drawstyle	integer		3	+1: draw outline, +2 fill area, +4 highlight points, +8 colored: outline
Graphics.pointsize	double		0.1	size for highlighted points [m]
Graphics.linethickness	double		1	thickness of lines [pts]
Graphics. smooth_drawing	bool		1	Draw smooth interpolation of surface
Graphics. draw_edge_angle	double		36	Minimum angle between two triangles that defines an edge (ř)
Geometry				
Geometry. transform_scale	vector		[1, 1, 1]	Resize GeomElement in X, Y and Z direction [sX, sY, sZ]
Geometry. transform_rotation	vector		[0, 0, 0]	Resize GeomElement in X, Y and Z direction [sX, sY, sZ]
Geometry. transform_position	vector		[0, 0, 0]	Translate GeomElement in X, Y and Z direction [tX, tY, tZ]
MeshData				
MeshData.triangles	matrix		[]	Fill in point numbers of each triangle: p1, p2, p3; p4, p5, p6 ...

MeshData.points	matrix		[]	Fill in point coordinates: X1, Y1, Z1; X2, Y2, Z2 ...

3.10.2 GeomCylinder3D

Data objects of GeomCylinder3D:

Data name	type	R	default	description
geom_element_type	string		"GeomCylinder3D"	specification of GeomElement type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"GeomElement"	name of the GeomElement
reference_element_number	integer		0	0 ... ground, otherwise insert number of existing element
Graphics				
Graphics.RGB_color	vector		[0.2, 0.2, 0.8]	[red, green, blue], range = 0..1
Graphics.transparency	double		-1	transparency [0..1], 0=transparent, 1=solid, set -1 if global transparency is used
Graphics.drawstyle	integer		3	+1: draw outline, +2 fill area, +4 highlight points, +8 colored: outline
Graphics.pointsize	double		0.1	size for highlighted points [m]
Graphics.linethickness	double		1	thickness of lines [pts]
Graphics.draw_resolution	integer		16	Number of quadrangles to draw the cylinder surface
Graphics.split_coloring	bool		0	true if one side should be slightly lighter than the other
Geometry				
Geometry.radius	double		0	radius of the cylinder
Geometry.radius_hole	double		0	inner radius of the cylinder (0 if full cylinder)
Geometry.axis_point1	vector		[0, 0, 0]	point on axis of rotation
Geometry.axis_point2	vector		[0, 0, 0]	point on axis of rotation

3.10.3 GeomSphere3D

Data objects of GeomSphere3D:

Data name	type	R	default	description
geom_element_type	string		"GeomSphere3D"	specification of GeomElement type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"GeomElement"	name of the GeomElement
reference_element_number	integer		0	0 ... ground, otherwise insert number of existing element
Graphics				
Graphics.RGB_color	vector		[0.2, 0.2, 0.8]	[red, green, blue], range = 0..1
Graphics.transparency	double		-1	transparency [0..1], 0=transparent, 1=solid, set -1 if global transparency is used
Graphics.drawstyle	integer		3	+1: draw outline, +2 fill area, +4 highlight points, +8 colored: outline
Graphics.pointsize	double		0.1	size for highlighted points [m]
Graphics.linethickness	double		1	thickness of lines [pts]

Graphics.draw_resolution	integer		16	Number of quadrangles to draw the sphere
Geometry				
Geometry.radius	double		0	radius of the sphere
Geometry.center_point	vector		[0, 0, 0]	center point of the sphere

3.10.4 GeomCube3D

Data objects of GeomCube3D:

Data name	**type**	**R**	**default**	**description**
geom_element_type	string		"GeomCube3D"	specification of GeomElement type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"GeomElement"	name of the GeomElement
reference_element_number	integer		0	0 ... ground, otherwise insert number of existing element
Graphics				
Graphics.RGB_color	vector		[0.2, 0.2, 0.8]	[red, green, blue], range = 0..1
Graphics.transparency	double		-1	transparency [0..1], 0=transparent, 1=solid, set -1 if global transparency is used
Graphics.drawstyle	integer		3	+1: draw outline, +2 fill area, +4 highlight points, +8 colored: outline
Graphics.pointsize	double		0.1	size for highlighted points [m]
Graphics.linethickness	double		1	thickness of lines [pts]
Geometry				
Geometry.point1	vector		[-0.5, -0.5, -0.5]	Bottom point 1 of bottom points: 1-2-4-3
Geometry.point2	vector		[0.5, -0.5, -0.5]	Bottom point 2 of bottom points: 1-2-4-3
Geometry.point3	vector		[-0.5, 0.5, -0.5]	Bottom point 3 of bottom points: 1-2-4-3
Geometry.point4	vector		[0.5, 0.5, -0.5]	Bottom point 4 of bottom points: 1-2-4-3
Geometry.point5	vector		[-0.5, -0.5, 0.5]	Bottom point 5 of bottom points: 5-6-8-7
Geometry.point6	vector		[0.5, -0.5, 0.5]	Bottom point 6 of bottom points: 5-6-8-7
Geometry.point7	vector		[-0.5, 0.5, 0.5]	Bottom point 7 of bottom points: 5-6-8-7
Geometry.point8	vector		[0.5, 0.5, 0.5]	Bottom point 8 of bottom points: 5-6-8-7

3.10.5 GeomOrthoCube3D

Data objects of GeomOrthoCube3D:

Data name	**type**	**R**	**default**	**description**
geom_element_type	string		"GeomOrthoCube3D"	specification of GeomElement type. Once the element is added to the mbs, you MUST NOT change this type anymore!
name	string		"GeomElement"	name of the GeomElement

reference_element_number	integer		0	0 ... ground, otherwise insert number of existing element
Graphics				
Graphics.RGB_color	vector		[0.2, 0.2, 0.8]	[red, green, blue], range = 0..1
Graphics.transparency	double		-1	transparency [0..1], 0=transparent, 1=solid, set -1 if global transparency is used
Graphics.drawstyle	integer		3	+1: draw outline, +2 fill area, +4 highlight points, +8 colored: outline
Graphics.pointsize	double		0.1	size for highlighted points [m]
Graphics.linethickness	double		1	thickness of lines [pts]
Geometry				
Geometry.center_point	vector		[0, 0, 0]	Center point in global coordinates
Geometry.size	vector		[1, 1, 1]	Dimension of cube in X, Y and Z-direction
Geometry.rotation_matrix	matrix		[1, 0, 0; 0, 1, 0; 0, 0, 1]	The rotation matrix defines the orientation of the cube (global_point = matrix . local_point).

3.11 Set

These Sets are available:

- ElementSet, 3.11.1
- GlobalNodeSet, 3.11.2
- LocalNodeSetA, 3.11.3
- LocalNodeSetB, 3.11.4
- GlobalCoordSet, 3.11.5
- LocalCoordSetA, 3.11.6
- LocalCoordSetB, 3.11.7

In HOTINT sets of different types can be defined. They fall in different categories (ElementSets, PointSet). For some categories there is more then one possibility to define a set of a given type. The PointSet can be defined as, for example list of global positions, list of elements and local nodes, ...
Sets can be manipulated via the GUI (add, edit, delete). the scipt language also provides the command `AddSet`.
The different possibilities to define a set all lead to the same data for further processing. All PointSets are automatically converted to a list of (element number,local position). Sets can be used as input parameters for functions to generate constraints, manipulate properties.

3.11.1 ElementSet

defines set of elements

Data objects of ElementSet:

Data name	**type**	**R**	**default**	**description**
set_name	string		"ElementSet"	the name of the set
set_type	string		"ElementSet"	type of the set
element_numbers	vector		[]	Elements in this set

3.11.2 GlobalNodeSet

defines set of global nodes

Data objects of GlobalNodeSet:

Data name	**type**	**R**	**default**	**description**
set_name	string		"GlobalNodeSet"	the name of the set
set_type	string		"GlobalNodeSet"	type of the set
element_numbers	vector	**R**	[]	Elements in this set
local_positions	matrix	**R**	[]	Local positions on the elements in this set
global_node_numbers	vector		[]	Global Nodes in this set

3.11.3 LocalNodeSetA

defines pairs of (element,local nodes)

Data objects of LocalNodeSetA:

Data name	**type**	**R**	**default**	**description**
set_name	string		"LocalNodeSetA"	the name of the set
set_type	string		"LocalNodeSetA"	type of the set
element_numbers	vector		[]	Elements in this set
local_positions	matrix	**R**	[]	Local positions on the elements in this set
local_node_numbers	vector		[]	Local Nodes in this set

3.11.4 LocalNodeSetB

defines set of elements and set of local nodes valid for each of these elements - all combinations

Data objects of LocalNodeSetB:

Data name	**type**	**R**	**default**	**description**
set_name	string		"LocalNodeSetB"	the name of the set
set_type	string		"LocalNodeSetB"	type of the set
element_numbers	vector	**R**	[]	Elements in this set
local_positions	matrix	**R**	[]	Local positions on the elements in this set
element_numbers_shortlist	vector		[]	Shortlist of Elements in this set
local_node_numbers_shortlist	vector		[]	Shortlist of Local Nodes in this set

3.11.5 GlobalCoordSet

defines set of global positions

Data objects of GlobalCoordSet:

Data name	**type**	**R**	**default**	**description**
set_name	string		"GlobalCoordSet"	the name of the set
set_type	string		"GlobalCoordSet"	type of the set
element_numbers	vector	**R**	[]	Elements in this set
local_positions	matrix	**R**	[]	Local positions on the elements in this set
global_positions	matrix		[]	Global positions registered in this set

3.11.6 LocalCoordSetA

defines pairs of (element,local positions)

Data objects of LocalCoordSetA:

Data name	**type**	**R**	**default**	**description**
set_name	string		"LocalCoordSetA"	the name of the set
set_type	string		"LocalCoordSetA"	type of the set

element_numbers	vector		[]	Elements in this set
local_positions	matrix		[]	Local positions on the elements in this set

3.11.7 LocalCoordSetB

defines set of elements and set of local positions valid for each of these elements - all combinations

Data objects of LocalCoordSetB:

Data name	**type**	**R**	**default**	**description**
set_name	string		"LocalCoordSetB"	the name of the set
set_type	string		"LocalCoordSetB"	type of the set
element_numbers	vector	**R**	[]	Elements in this set
local_positions	matrix	**R**	[]	Local positions on the elements in this set
element_numbers_shortlist	vector		[]	Shortlist of Elements in this set
local_positions_shortlist	matrix		[]	Shortlist of Local Positions in this set

3.12 Command

These commands are available:

- AddElement, 3.12.1
- AddGeomElement, 3.12.2
- AssignGeomElementToElement, 3.12.3
- AddConnector, 3.12.4
- AddLoad, 3.12.5
- AddSensor, 3.12.6
- AddMaterial, 3.12.7
- AddBeamProperties, 3.12.8
- AddNode, 3.12.9
- Include, 3.12.10
- Print, 3.12.11
- ReadSTLFile, 3.12.12
- LoadVectorFromFile, 3.12.13
- TransformPoints, 3.12.14
- ComputeInertia, 3.12.15
- Sum, 3.12.16
- Product, 3.12.17
- Transpose, 3.12.18
- CrossProduct, 3.12.19
- for, 3.12.20
- if, 3.12.21
- DoesEntryExist, 3.12.22
- GetByName, 3.12.23
- SetByName, 3.12.24
- Compare, 3.12.25
- StrCat, 3.12.26
- Zeros, 3.12.27

- AddSet, 3.12.28
- GenerateConstraints, 3.12.29
- AssignMaterial, 3.12.30
- AssignLoad, 3.12.31
- ChangeProperties, 3.12.32
- SetInitialCondition, 3.12.33

3.12.1 AddElement

Adds an element to the system. See the description of the elements above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the element. ATTENTION: the entry element_type must exist!

return values:

The return value of this command is the number of the element in the MBS.

Example

see file AddElement.txt

```
emptyMass3D
{
  element_type = "Mass3D"
  Physics.mass= 1
}
nElement = AddElement(emptyMass3D)
```

3.12.2 AddGeomElement

This command adds an geometric element.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the geometric element. ATTENTION: the entry geom_element_type must exist!

return values:

The return value of this command is the number of the geometric element in the MBS.

Example

see file AddGeomElement.txt

```
myCube
{
  name= "myGeomElement"
  geom_element_type = "GeomOrthoCube3D"
  Geometry.center_point= [0.0, 0.0, 0.0]
  Geometry.size= [1.0, 1.0, 1.0]
}
AddGeomElement(myCube)
```

3.12.3 AssignGeomElementToElement

This command assigns a geom element to an element. The reference point and rotation of the element are evaluated and the settings of the geom element are modified automatically, such that the current relative orientation of element and geom element keeps the same. You can therefore add and align the geom element independently from the element first and afterwards decide to connect the geom element to the element without the need of changing the settings of the geom element again.
Parameters:
The parameters of this command are

1. 1^{st} parameter: an element number
2. 2^{nd} parameter: a geom element number

-
return values:
returns 0 or an error code

Example

see file AssignGeomElementToElement.txt

```
red = [1,0,0] % colour for "relative" (geom) elements
blue = [0,0,1]  % colour for "absolute" (geom) elements

geomElement % define and add some geom element
{
  name = "absolute geom Element"
  geom_element_type = "GeomCylinder3D"
  Geometry.radius= 0.1  % radius of the cylinder
  Geometry.axis_point1= [1, 1, 0]  % point on axis of rotation
  Geometry.axis_point2= [1.5, 1, 0]  % point on axis of rotation
  Graphics.RGB_color=blue
}
nGeomEl_absolute = AddGeomElement(geomElement)
% the geomElement is added to the mbs with global positions

geomElement.Graphics.RGB_color=red
geomElement.name = "relative geom Element"
geomElement.Geometry.axis_point1= [0, 0, 0]  % point on axis of rotation
geomElement.Geometry.axis_point2= [0.5, 0, 0]  % point on axis of rotation
nGeomEl_relative = AddGeomElement(geomElement)
% the geomElement is added a second time to the mbs with different
% global positions and color
```

```
elementRelative
{
  name = "relative element"
  element_type= "Rigid3DMinCoord"
  Graphics.use_alternative_shape = 1
  Graphics.geom_elements = [nGeomEl_relative]
  Graphics.position_offset = [1,0,0]
  Graphics.RGB_color=red
}
nERel = AddElement(elementRelative)
% the geomElement "relative" is linked to the element "relative" at the time when the
% element is added to the mbs
% the coordinates of the geomElement are now relative to the reference point of the element
% the element "relative" is not visible, only the geomElement is visible

elementAbsolute
{
  name = "absolute element"
  element_type= "Rigid3DMinCoord"
  Graphics.position_offset = [1,1,0]
  Graphics.RGB_color=blue
}
nEAbs = AddElement(elementAbsolute)
% the geomElement "absolute" and the element "absolute" are both visible in mbs
% you can check the alignment

AssignGeomElementToElement(nEAbs,nGeomEl_absolute)
% the element "absolute" vanishes but the geomElement "absolute" stays at the same place
% the settings of the geomElement were adjusted automatically
% the element "absolute" is linked with the geomElement "absolute"
```

3.12.4 AddConnector

Adds a connector to the system. See the description of the connectors above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the connector. ATTENTION: the entry element_type must exist!

return values:

The return value of this command is the number of the connector in the MBS.

Example

see file AddConnector.txt

```
RigidBody % define some element
{
  element_type = "Rigid3D"
  Physics.mass = 1
  Graphics.Body_dimensions = [0.1,1,0.1]
}
nElement =AddElement(RigidBody)

myConnector
{
  element_type = "PointJoint"
  Physics
  {
    use_penalty_formulation = 0
    Lagrange
    {
      constrained_directions = [1,1,1]
    }
  }
  Position1
  {
    element_number = nElement
    position = [0,-0.5,0]
  }
  Position2
  {
    element_number = 0  % = 0 -> global node/coordinate
    position = [0,0,0]  % position of ground
  }
  Graphics.draw_size = 0.05
}
nConnector = AddConnector(myConnector)
```

3.12.5 AddLoad

Adds a load to the system. See the description of the loads above in order to get the available options. You have to adjust the value 'loads' in the element to assign the load to the element.
Parameters:
The parameter of this command is an ElementDataContainer with the data of the load. ATTENTION: the entry load_type must exist!
return values:
The return value of this command is the number of the load in the MBS.

Example

see file AddLoad.txt

```
myLoad    % define the load
{
  load_type = "Gravity"
  name = "gravity for all elements"
  direction = 2
  gravity_constant = 9.81
}
nLoad=AddLoad(myLoad)

emptyMass3D % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
  loads = [nLoad] % add the load to the element
}
nElement = AddElement(emptyMass3D)

ViewingOptions.Loads.show_loads = 1
ViewingOptions.Loads.arrow_size = 0.2
```

3.12.6 AddSensor

Adds a sensor to the system. See the description of the sensors above in order to get the available options.
Parameters:
The parameter of this command is an ElementDataContainer with the data of the sensor. ATTENTION: the entry sensor_type must exist!
return values:
The return value of this command is the number of the sensor in the MBS.

Example

see file AddSensor.txt

```
emptyMass3D   % define some element
{
  element_type = "Mass3D"
  Physics.mass= 1
}
nElement = AddElement(emptyMass3D)

mySensor
{
  sensor_type = "FVElementSensor"
  name = "Position of the Mass3D in z-direction"
  element_number = nElement
  field_variable = "position"
  component = "z"
}
nSensor = AddSensor(mySensor)

ViewingOptions.Sensors.show_sensors = 1
```

3.12.7 AddMaterial

Adds a material to the system. See the description of the materials above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the material. ATTENTION: the entry material_type must exist!

return values:

The return value of this command is the number of the material in the MBS.

Example

see file AddMaterial.txt

```
Material1
{
  material_type= "Material"
  Solid
  {
    density= 7850  % density (rho) for gravitational force
    youngs_modulus= 2.1e11  %Youngs modulus
    poisson_ratio= 0.3  %Poisson ratio
  }
}
AddMaterial(Material1)
```

3.12.8 AddBeamProperties

Adds a BeamProperty to the system. See the description of the BeamProperties above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the BeamProperties. ATTENTION: the entry material_type must exist!

return values:

The return value of this command is the number of the node in the MBS.

Example

see file AddBeamProperties.txt

```
beam1
{
  material_type = "Beam3DProperties"
  cross_section_size = [0.1,0.1]
  EA = 2e9
  EIy = 2e6
  EIz = 2e6
  GJkx = 2e6
}
AddBeamProperties(beam1)
```

3.12.9 AddNode

Adds a node to the system. See the description of the nodes above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the node. ATTENTION: the entry node_type must exist!

return values:

The return value of this command is the number of the node in the MBS.

Example

see file AddNode.txt

```
node1
{
  node_type = "Node3DS1rot1"
}
AddNode(node1)

ViewingOptions.FiniteElements.Nodes.show = 1
ViewingOptions.FiniteElements.Nodes.node_size = 0.05
```

3.12.10 Include

This command includes a file.

Parameters:

The parameter of this command is the absolut or relative filename. If a relative filename is used, then the path is relative to the last file! Be carefull, if you use this command more than one time in a file.

return values:

There is no return value defined yet.

Example

see file Include.txt

```
%Include("D:\HelloWorld.txt")           % absolute file path
Include("..\..\examples\double_pendulum.txt")   % relative path 1
%Include("AddElement.txt")                      % relative path 2 (same folder)
```

3.12.11 Print

Prints a text to the output window

Parameters:

There are three possibilities to use the command. The parameter can either be:

- a text, e.g. Print("Hello world")
- an ElementDataContainer, e.g. Print(my_mass)
- an ElementData, e.g. Print(my_mass.density)

In the case of a text or an ElementData, only the text itself is printed. In the case of an ElementDataContainer, also the name of the ElementData is printed.
return values:
There is no return value for this command

Example

see file Print.txt

```
Print("Hello world! \n")

TestEDC
{
  number = 1
  text = "this is a text in an edc"
}
Print("\nPrinting the edc:\n")
Print(TestEDC)
Print("\n")
Print("Printing elements of the edc: \n")
Print(TestEDC.text)
Print("\n")
Print(TestEDC.number)
Print("\n")
```

3.12.12 ReadSTLFile

This command reads a stl-mesh from a file and stores the data in an ElementDataContainer.
Parameters:
The parameter of this command is the absolut or relative filename.
return values:
The return value is an ElementDataContainer with 2 entries: triangles and points.

Example

see file ReadSTLFile.txt

```
STL = ReadSTLFile("mesh.stl")

myGeomElementMesh3D
{
  geom_element_type = "GeomMesh3D"
  % MeshData = STL  % not possible yet
  MeshData.triangles = STL.triangles
  MeshData.points = STL.points
}
nGeom1 = AddGeomElement(myGeomElementMesh3D)
```

3.12.13 LoadVectorFromFile

This command reads a vector from a file and returns this vector.
Parameters:
The parameters of this command are

1. The name of the file as string
2. An integer defining in which column (default) or row of the file the vector is stored
3. (optional) 0.. take the column (default), 1.. take the row

-
return values:
The return value is the vector.

Example

see file LoadVectorFromFile.txt

```
%========== basic example ===============
t = LoadVectorFromFile("solution.txt",1)  % relative path
% x = LoadVectorFromFile("D:\sol.txt",2)  % absolute also possible
x = LoadVectorFromFile("solution.txt",2)
x7 = x[7]       % direct access to element of vector

%========== extended example ============
% use loaded vectors to define a MathFunction
Time.element_type= "IOTime"
nTime = AddElement(Time)    % time as input for the Mathfunction

Mathf
{
  element_type= "IOMathFunction"
  Graphics.position= [100, 0]
  IOBlock
  {
    input_element_numbers= [nTime]
    input_element_types= [1]  % vector with types of connected inputs; 1=IOElement
    input_local_number= [1]   % i-th number of output of previous IOelement
    MathFunction
    {
      piecewise_mode= 1   % modus for piecewise interpolation: 1=linear
      piecewise_points= t   % supporting points for piecewise interpolation
      piecewise_values= x   % values at supporting points
    }
  }
}
```

```
nMF = AddElement(Mathf)

% sensor to measure the output of the mathfunction
sensor.sensor_type= "ElementSensor"
sensor.element_number= nMF
sensor.value= "IOBlock.output[1]"
AddSensor(sensor)
```

3.12.14 TransformPoints

With this command, the geometry described by the points can be transformed. It is possible to apply rotation and/or translation and/or scaling. The new point pN is computed according to the formula pN = trans + rot*p.

Parameters:

The parameters of this command are as follows

1. points: Matrix of the points: Each line represents a point p. The 3 columns are the x-, y- and z-coordinate
2. trans: Vector of translation, 3 dimensions!
3. rot: rotation matrix (3x3), can be used for scaling as well as rotation

-

return values:

The return value is a Matrix containing the transformed points pN.

Example

see file TransformPoints.txt

```
STL = ReadSTLFile("mesh.stl") % load mesh

% add geomElement with original points
myGeomElementMesh3D
{
  geom_element_type = "GeomMesh3D"
  MeshData.triangles = STL.triangles
  MeshData.points = STL.points
  Graphics.RGB_color = [0.2,0.2,0.8]
}
nGeom1 = AddGeomElement(myGeomElementMesh3D)

% transform points
vec = [0,50,0]  % translation
A = [0.75,0,0;0,0.75,0;0,0,0.75]  % scaling
points=TransformPoints(STL.points,vec,A)

% add geomElement with transformed points
myGeomElementMesh3D.MeshData.points = points
myGeomElementMesh3D.Graphics.RGB_color = [0.2,0.8,0.2]
nGeom2 = AddGeomElement(myGeomElementMesh3D)
```

3.12.15 ComputeInertia

This command computes the mass, moment of inertia, volume and center of mass based on the information about the geometry and the material of a body

Parameters:

The parameter of this command is an ElementDataContainer, with the following entries:

- density or material_number (one of these 2 has to be set!)
- One of the following options to define the geometry:
  - MeshData.triangles and MeshData.points
    both entries are Matrices with 3 columns
  - Cube.body_dimensions

-

return values:

The return value is an ElementDataContainer with 4 entries: volume, mass, moment_of_inertia and center_of_mass

Example

see file ComputeInertia.txt

```
% simple example with a cube
my_data
{
  density = 7850
  Cube.body_dimensions = [1.0,0.1,0.1]
}
CI1 = ComputeInertia(my_data)
Print(CI1)

% example with a mesh
STL = ReadSTLFile("mesh.stl")
Material1
{
  material_type= "Material"
  Solid.density= 7850
}
n = AddMaterial(Material1)

my_data2
{
  material_number = n
  MeshData
  {
    triangles = STL.triangles
    points = STL.points
  }
}
```

```
CI2 = ComputeInertia(my_data2)
Print(CI2)
```

3.12.16 Sum

This command adds two components of the same type (scalar, vector or matrix).
Parameters:
The parameters of this command are

1. 1^{st} summand, either scalar, vector or matrix

2. 2^{nd} summand, either scalar, vector or matrix

-
return values:
The return value is the sum of the two inputs.

Example

see file Sum.txt

```
Scalar = 1.5
Vector2D = [1,2]
Matrix2D = [0,1;2,0]
s = Sum(Scalar,Scalar)           % 3
v = Sum(Vector2D,Vector2D)       % [2,4]
m = Sum(Matrix2D,Matrix2D)       % [0,2;4,0]
```

3.12.17 Product

This command multiplies two components of the type (scalar, vector or matrix) when the operation is defined.
Parameters:
The parameters of this command are

1. 1^{st} factor, either scalar, vector or matrix

2. 2^{nd} factor, either scalar, vector or matrix

product of two vectors is always computed as scalar product for vector times Matrix the vector is automatically transposed if required -
return values:
The return value is the product of the two inputs.

Example

see file Product2.txt

```
Scalar = 1.5
Vector2D = [1,2]
Matrix2D = [0,1;2,0]

s1 = Product(Scalar,Scalar)        % 2.25
v1 = Product(Scalar,Vector2D)      % [1.5,3]
m1 = Product(Scalar,Matrix2D)      % [0,1.5;3,0]
s2 = Product(Vector2D,Vector2D)
v2 = Product(Vector2D,Scalar)
m2 = Product(Matrix2D,Scalar)      % [0,1.5;3,0]
```

3.12.18 Transpose

This command transposes a matrix or vector.
Parameters:
The parameters of this command are

1. vector or matrix to be transposed

-
return values:
The return value is a matrix or a vector.

Example

see file Transpose.txt

```
Vector2D = [1,2]
a = Transpose(Vector2D)          % [1;2]
b = Transpose(a)                 % [1,2]
```

3.12.19 CrossProduct

This command computes the cross product of two vectors.
Parameters:
The parameters of this command are

1. 1^{st} vector (2D or 3D)

2. 2^{nd} vector (2D or 3D)

for two 3D vectors the retuen value is also a 3D vector. For two 2D vectors the return value is a scalar. -
return values:
The return value is the scalar cross product.

Example

see file CrossProduct.txt

```
  v1 = [1,2,3]
  v2 = [2,3,4]
  C1 = CrossProduct(v1,v2)          % [-1, 2,-1]
  C2 = CrossProduct(v2,v1)          % [ 1,-2, 1]
```

3.12.20 for

This command starts a FOR loop for the subsequent block.
Parameters:
The parameters of this command are

1. 1^{st} define and initialize loop variable ("i=1")
2. 2^{nd} loop condition ("i<5")
3. 3^{rd} loop increment ("i=i+1")

the command must be followed by a container for the loop code -
return values:
The return value is the number of iterations.

Example

see file LoopCond.txt

```
%% Test 1
Test1                       % general function and tree correctness
{
  sum = 0
  for(i=1,i<11,i=i+1)
  {
    sum = sum + i
  }
  Print("Test1: ")
  Print(sum)
  Print(" (55)\n")
  if(sum==55)
  {
    Print("TEST 1 PASSED \n")
  }
}


%% Test2                     % nesting loops
%Test2
%{
  for(i=1,i<5,i=i+1)
  {
    for(j=1,j<5,j=j+1)
    {
      Mass3D
      {
        element_type = "Mass3D"
        Physics.mass= 1
        Initialization.initial_position= [i,j, 0]
        Graphics.RGB_color = [1,1,1]
      }
      if(i==j)
      {
        if(i==1)
        {
          Mass3D.Graphics.RGB_color = [0,0,0]
        }
        if(i==2)
        {
    Mass3D.Graphics.RGB_color = [1,0,0]
  }
        if(i==3)
        {
    Mass3D.Graphics.RGB_color = [0,1,0]
  }
        if(i==4)
        {
    Mass3D.Graphics.RGB_color = [0,0,1]
  }
      }
      elnr = AddElement(Mass3D)
      Print("Added Element ")
      Print(elnr)
      Print(" to MBS\n")
    }
  }
%}
```

3.12.21 if

This command evaluates an IF condition for the subsequent block.
Parameters:
The parameters of this command are

1. 1^{st} condition ("i<10")

the command must be followed by a container for the conditional code -
return values:
The return value is the 1 for true and 0 for false.

Example

see file LoopCond.txt

```
%% Test 1
Test1                        % general function and tree correctness
{
  sum = 0
  for(i=1,i<11,i=i+1)
  {
    sum = sum + i
  }
  Print("Test1: ")
  Print(sum)
  Print(" (55)\n")
  if(sum==55)
  {
    Print("TEST 1 PASSED \n")
  }
}


%% Test2                      % nesting loops
%Test2
%{
  for(i=1,i<5,i=i+1)
  {
    for(j=1,j<5,j=j+1)
    {
      Mass3D
      {
        element_type = "Mass3D"
        Physics.mass= 1
        Initialization.initial_position= [i,j, 0]
        Graphics.RGB_color = [1,1,1]
      }
      if(i==j)
      {
        if(i==1)
        {
          Mass3D.Graphics.RGB_color = [0,0,0]
        }
        if(i==2)
        {
    Mass3D.Graphics.RGB_color = [1,0,0]
  }
        if(i==3)
        {
    Mass3D.Graphics.RGB_color = [0,1,0]
  }
        if(i==4)
        {
    Mass3D.Graphics.RGB_color = [0,0,1]
  }
      }
      elnr = AddElement(Mass3D)
      Print("Added Element ")
      Print(elnr)
      Print(" to MBS\n")
    }
  }
%}
```

3.12.22 DoesEntryExist

This command checks if the specified entry exists.
Parameters:
The parameters of this command are

1. 1^{st} parameter: string with the name and tree of the entry to check

-
return values:
returns 0 if the entry does not exist, returns 1 if the entry exists, returns 2 if the entry is an EDC

Example

see file Container.txt

```
Root.NodeL.NodeL.path = [ 0 0 ]
Root.NodeL.NodeR.path = [ 0 1 ]
Root.NodeR.path = [ 1 ]

Leaf = "Root.NodeL.NodeR"
flag_exist = DoesEntryExist(Leaf)

str = StrCat(Leaf,".path")
p = GetByName(str)

newpath = [1 0]
SetByName("Root.NodeR.NodeL.path",newpath)

null = 0
SetByName("Root.NodeL",null)
```

3.12.23 GetByName

This command lets you get any (existing) EDC entry by name.
Parameters:
The parameters of this command are

1. 1^{st} parameter: string with the name and tree of the entry to get

-
return values:
returns the entry associated with the string.

Example

see file Container.txt

```
Root.NodeL.NodeL.path = [ 0 0 ]
Root.NodeL.NodeR.path = [ 0 1 ]
Root.NodeR.path = [ 1 ]

Leaf = "Root.NodeL.NodeR"
flag_exist = DoesEntryExist(Leaf)

str = StrCat(Leaf,".path")
p = GetByName(str)

newpath = [1 0]
SetByName("Root.NodeR.NodeL.path",newpath)

null = 0
SetByName("Root.NodeL",null)
```

3.12.24 SetByName

This command lets you set any EDC entry by name, the name contains the absolute treename.
Parameters:
The parameters of this command are

1. 1^{st} parameter: string with the name and tree of the entry to set
2. 2^{nd} parameter: the variable that should be assigned

-
return values:
has no return value.

Example

see file Container.txt

```
Root.NodeL.NodeL.path = [ 0 0 ]
Root.NodeL.NodeR.path = [ 0 1 ]
Root.NodeR.path = [ 1 ]

Leaf = "Root.NodeL.NodeR"
flag_exist = DoesEntryExist(Leaf)

str = StrCat(Leaf,".path")
p = GetByName(str)

newpath = [1 0]
SetByName("Root.NodeR.NodeL.path",newpath)

null = 0
SetByName("Root.NodeL",null)
```

3.12.25 Compare

This command compares two strings.
Parameters:
The parameters of this command are

1. 1^{st} parameter: string A

2. 2^{nd} parameter: string B

-
return values:
returns 0 if both strings are identical, returns >0 or <0 otherwise indicating which string has higher value .

Example

see file strings.txt

```
str = "string"
strA = "stringA"
strB = "stringB"
str1 = "1"
str2 = "2"

mone = Compare(strA,strB)
pone = Compare(strB,strA)
zero = Compare(str,str)

string1 = StrCat(str,str1)
string2 = StrCat(str,str2)
string12 = StrCat(string1,str2)
```

3.12.26 StrCat

This command joins two strings together
Parameters:
The parameters of this command are

1. 1^{st} parameter: string A

2. 2^{nd} parameter: string B

You can also use integer or double values instead of the strings. Inline definition of strings, e.g. StrCat(”this is a ”,”test”), do not work properly. The spaces are not taken into account correctly! -
return values:
returns a single string - strA+strB.

Example

see file strings.txt

```
str = "string"
strA = "stringA"
strB = "stringB"
str1 = "1"
str2 = "2"

mone = Compare(strA,strB)
pone = Compare(strB,strA)
zero = Compare(str,str)

string1 = StrCat(str,str1)
string2 = StrCat(str,str2)
string12 = StrCat(string1,str2)
```

3.12.27 Zeros

This command sets a vector or matrix variable to agiven dimension and sets all entries to 0
Parameters:
The parameters of this command are

1. 1^{st} parameter: length of the vector or first dimension of the matrix

2. 2^{nd} parameter: second dimenstion of the matrix

-
return values:
returns a vector or matrix variable

Example

see file lists.txt

```
mat22 = Zeros(2,2)    % matrix
vec3 = Zeros(3,1)     % vector
vec3t = Zeros(1,3)    % matrix

vec3[1] = 1
mat22[1,1] = 11
mat22[1,2] = 12

squares = Zeros(10,1)
for(i=1,i<11,i=i+1)
{
  squares[i] = i*i
}
```

3.12.28 AddSet

Adds a set to the system. See the description of the set above in order to get the available options.

Parameters:

The parameter of this command is an ElementDataContainer with the data of the set. ATTENTION: the entry set_type must exist!

return values:

The return value of this command is the number of the set in the MBS.

Example

see file AddSet.txt

```
myRigid
{
  element_type= "Rigid3D"  %specification of element type.
}
AddElement(myRigid)
myRigid.Initialization.initial_position= [1, 0, 0]
AddElement(myRigid)

Set1
{
  set_name = "SetOfElements1"
  set_type = "ElementSet"
  element_numbers = [1,2]
}
nSet1 = AddSet(Set1)
```

3.12.29 GenerateConstraints

This command generates constraints for the given set

Parameters:

The parameters of this command are

1. 1^{st} parameter: a set of global node numbers (more types will be added)

2. 2^{nd} parameter: parameters for the constraints

enties in the properties EDC are:

- mode - 'ground' or 'pair'(default)

- type - type string for the constraint to use

-

return values:

returns a list of element numbers for the generated constraints

Example

see file GenerateConstraints.txt

```
meshparameters
{
  mesh_type = "StructuralMesh"
  mesh_name = "Mesh1"
}
Mesh1 = GenerateNewMesh(meshparameters)

platematerial
{
  material_type = "Material"
  Solid.density = 7850
  Solid.youngs_modulus = 2.1e11
  Solid.poisson_ratio = 0.3
}
mnr_plate = AddMaterial(platematerial)

discr_x = 3
discr_y = 3
plateparameters
{
  component_name = "tile_1"
  Generation.matnr = mnr_plate
  Generation.discretization = [discr_x,discr_y]
  Generation.thickness = 0.01
}
Mesh1.GeneratePlate(plateparameters)
MeshAsVariable = Mesh1.AddMeshToMBS(1)

nnrs = [ 1 (1+discr_x) 1+discr_y*(1+discr_x) (1+discr_y)*(1+discr_x) ]
Set1
{
  set_name = "CornerNodes"
  set_type = "GlobalNodeSet"
  global_node_numbers = nnrs
}
nSet1 = AddSet(Set1)

consraintparameter.mode = "ground"
cnrs = GenerateConstraints(nSet1,consraintparameter)

Set2
{
  set_name = "theConstraints"
  set_type = "ElementSet"
  element_numbers = cnrs
}
nSet2 = AddSet(Set2)
replace.Physics.Lagrange.constrained_rotations= [0 0 0]
ChangeProperties(nSet2,replace)
```

```
g_force_factor = 1 %9.81
dir = 3
gravity
{
  name= "Acceleration on Panels"  %name of the load
  load_type= "Gravity"  %specification of load type. Once the load is added to the mbs, you MUS
  direction= dir  %global direction of the gravity
  gravity_constant= g_force_factor  %use negative sign if necessary
}
loadnr = AddLoad(gravity)

elspp = discr_x*discr_y
elnrs = Zeros(elspp,1)
for(i=1,i<=elspp,i=i+1)
{
  elnrs[i] = i
}
elementset
{
  set_name = "Elements of Panels"
  set_type = "ElementSet"
  element_numbers = elnrs
}
setnr = AddSet(elementset)

errorcode = AssignLoad(setnr,loadnr)
```

3.12.30 AssignMaterial

This command sets the material number of all elemenets of the element-set to a given number
Parameters:
The parameters of this command are

1. 1^{st} parameter: a set of elements
2. 2^{nd} parameter: material number to be assigned

-
return values:
returns 0 or an error code

Example

see file AssignMaterial.txt

```
beam_material
```

```
{
  material_type = "Beam3DProperties"
  cross_section_size = [0.1,0.1]
}
AddBeamProperties(beam_material)
AddBeamProperties(beam_material)
node
{
  node_type = "Node3DRxyz"
}
n1 = AddNode(node)
node.Geometry.reference_position = [1,0,0]
n2 = AddNode(node)
beam
{
  element_type= "LinearBeam3D"
  Physics.material_number = 1
  Geometry.node_1 = n1
  Geometry.node_2 = n2
}
nBeam = AddElement(beam)

Set1
{
  set_name = "ElementSet"
  set_type = "ElementSet"
  element_numbers = [1]
}
nSet1 = AddSet(Set1)

AssignMaterial(nSet1,2)        %% set with number nSet - assign material 2
AssignMaterial("ElementSet",1) %% set with name "ElementSet" - assign material 1
AssignMaterial(1,2)            %% first set - assign material 2
```

3.12.31 AssignLoad

This command adds a load to all elemenets of the element-set
Parameters:
The parameters of this command are

1. 1^{st} parameter: a set of elements

2. 2^{nd} parameter: load number to be added or "ClearAll" to remove all loads

-
return values:
returns 0 or an error code

Example

see file AssignLoad.txt

```
beam_material
{
  material_type = "Beam3DProperties"
  cross_section_size = [0.1,0.1]
}
AddBeamProperties(beam_material)
AddBeamProperties(beam_material)
node
{
  node_type = "Node3DRxyz"
}
n1 = AddNode(node)
node.Geometry.reference_position = [1,0,0]
n2 = AddNode(node)
beam
{
  element_type= "LinearBeam3D"
  Physics.material_number = 1
  Geometry.node_1 = n1
  Geometry.node_2 = n2
}
nBeam = AddElement(beam)

Gravity
{
  load_type = "Gravity"
  name = "Gravity"
  direction = 3
  gravity_constant = -9.81
}
gravnr = AddLoad(Gravity)

Set1
{
  set_name = "ElementSet"
  set_type = "ElementSet"
  element_numbers = [1]
}
nSet1 = AddSet(Set1)

AssignLoad(nSet1,1)                   %% set with number nSet - assign load 1
AssignLoad("ElementSet","Gravity")    %% set with name "ElementSet" - assign load named "Gravi
```

3.12.32 ChangeProperties

This command changes properties of the elements of the set
Parameters:
The parameters of this command are

1. 1^{st} parameter: a set of elements or global nodes
2. 2^{nd} parameter: EDC containing substitute parameters EDC

-

return values:
returns 0 or an error code

Example

see file ChangeProperties.txt

```
meshparameters
{
  mesh_type = "StructuralMesh"
  mesh_name = "Mesh1"
}
Mesh1 = GenerateNewMesh(meshparameters)

platematerial
{
  material_type = "Material"
  Solid.density = 7850
  Solid.youngs_modulus = 2.1e11
  Solid.poisson_ratio = 0.3
}
mnr_plate = AddMaterial(platematerial)

plateparameters
{
  component_name = "tile_1"
  Generation.P1 = [ 0., 0., 0.]
  Generation.P2 = [ 2., 0., 0.]
  Generation.P3 = [ 0., 2., 0.]
  Generation.P4 = [ 2., 2., 0.]
  Generation.matnr = mnr_plate
  Generation.discretization = [3,3]
  Generation.thickness = 0.1
}
Mesh1.GeneratePlate(plateparameters)

MeshAsVariable = Mesh1.AddMeshToMBS(1)

Set1
{
  set_name = "SomeElements"
  set_type = "ElementSet"
  element_numbers = [1,3,7,9]
}
nSet1 = AddSet(Set1)
invisible.Graphics.show_element = 0
```

```
ChangeProperties(nSet1,invisible)

Set2
{
  set_name = "AllNodes"
  set_type = "GlobalNodeSet"
  global_node_numbers = MeshAsVariable.list_of_nodes
}
nSet2 = AddSet(Set2)
initvel.Initialization.node_initial_values = [0,0,0, 0,0,0, 0,0,0, 0,0,1, 0,0,0, 0,0,0]
ChangeProperties(nSet2,initvel)
```

3.12.33 SetInitialCondition

This command sets the initial condition of all members of the element or node set.
Parameters:
The parameters of this command are

1. 1^{st} parameter: a set of elements or global nodes
2. 2^{nd} parameter: index of initial value
3. 3^{rd} parameter: expression of the value. Can contain:
   - "x" for the global reference position of the node or element
   - "y" for the global reference position of the node or element
   - "z" for the global reference position of the node or element

-

Example

see file SetInitialCondition.txt

3.13 Options

These options are available:

- SolverOptions 3.13.1
- LoggingOptions 3.13.2
- GeneralOptions 3.13.3
- ViewingOptions 3.13.4
- PlotToolOptions 3.13.5

SolverOptions can be saved in the GUI separately of the other HOTINT options.

3.13.1 SolverOptions

Data objects of SolverOptions:

Data name	**type**	**default**	**description**
SolverOptions			
SolverOptions.start_time	double	0	Starting time of simulation, usually 0; for static and timeint solver
SolverOptions.end_time	double	10	Final simulation time; for static and timeint solver
SolverOptions. do_static_computation	bool	0	Do only static computation; velocities and acceleration terms are ignored; system may not have kinematic degrees of freedom.
SolverOptions.Timeint			
SolverOptions.Timeint. max_step_size	double	0.001	Maxial step size of timeint solver.
SolverOptions.Timeint. min_step_size	double	0.0001	Minimal step size of timeint solver.
SolverOptions.Timeint. max_index	integer	2	maximum index which solver the solver needs to handle
SolverOptions.Timeint. tableau_name	string	"LobattoIIIA"	Runge Kutta tableau chosen
SolverOptions.Timeint. max_stages	integer	2	Number of stages for simulation, max. stages for variable order.
SolverOptions.Timeint. min_stages	integer	1	Min. stages for variable order.
SolverOptions.Timeint. automatic_stepsize_control	bool	0	1|(0) ... Full adaptive stepsize selection of timeint is (not) active?
SolverOptions.Timeint. init_step_size	double	0.01	Initial stepsize for timeint.
SolverOptions.Timeint. absolute_accuracy	double	0.01	Absolute accuracy, for full adaptive timeint.
SolverOptions.Timeint. relative_accuracy	double	1	Relative accuracy, for full adaptive timeint.
SolverOptions.Timeint. variable_order	integer	0	1|(0) ... Variable order algorithm is (not) active.
SolverOptions.Timeint. do_implicit_integration	bool	1	1 .. Use implicit integration, 0..use explicit integration.
SolverOptions.Timeint. reset_after_simulation	bool	1	Reset start time and initial values after each simulation.
SolverOptions.Timeint.assume_constant_mass_matrix	bool	0	Experimental version of constant mass matrix (WARNING: experimental only)

SolverOptions.Static

SolverOptions.Static.min_load_inc	double	1e-012	Minimal increment.
SolverOptions.Static.max_load_inc	double	1	Maximum load increment.
SolverOptions.Static.init_load_inc	double	1	Initial load increment.
SolverOptions.Static.load_inc_up	double	2	Increase load increment if success very often.
SolverOptions.Static.load_inc_down	double	2	Decrease load increment if no success.
SolverOptions.Static.increase_load_inc_steps	integer	1	If increase_load_inc_steps successfull steps –> leads to increase of load increment.
SolverOptions.Static.spring_regularisation_parameter	double	0	Spring-type regularisation parameter to stabilize almost kinematic systems during static comp.
SolverOptions.Static.use_tolerance_relax_factor	integer	0	Enables/disables [1/0] the use of the relaxation factor on the tolerance goal (discontinuous accuracy) within static comp. Relaxation depends on load factor (0..1)
SolverOptions.Static.max_tolerance_relax_factor	double	10	Uper bound for relaxation factor on the tolerance goal (discontinuous accuracy)
SolverOptions.Static.experimental_sparse_jacobian	bool	1	Experimental: optimized (low memory) sparse jacobian matrix

SolverOptions.Newton

SolverOptions.Newton.relative_accuracy	double	1e-008	Relative accuracy for Newton method
SolverOptions.Newton.absolute_accuracy	double	100	Absolute accuracy for Newton method
SolverOptions.Newton.num_diff_parameter	double	1e-007	Numerical differentiation parameter
SolverOptions.Newton.use_central_diff_quotient	bool	1	Use central difference quotient for numerical differentiation (slower).
SolverOptions.Newton.use_modified_newton	bool	1	Use modified Newton (approximated Jacobian, much faster).
SolverOptions.Newton.max_modified_newton_steps	integer	12	Max. modified Newton steps.
SolverOptions.Newton.max_restart_newton_steps	integer	15	Max. modified Newton steps after restart.
SolverOptions.Newton.max_full_newton_steps	integer	25	Max. full Newton steps.
SolverOptions.Newton.use_trust_region	bool	0	0...do not use trust region; 1..use line search algorithm for newton's method, usually not necessary.
SolverOptions.Newton.trust_region_division	double	0.1	Increment for line search.
SolverOptions.Newton.low_contractivity_tolerance	double	0.7	Used in modified Newton: if ratio error over last error violates this bound more than twice, then Jacobian is recomputed.
SolverOptions.Newton.high_contractivity_tolerance	double	2	Used in modified Newton: if ratio error over last error violates this bound more than twice, then switch to classical Newton method.

SolverOptions.Eigensolver

SolverOptions.Eigensolver.do_eigenmode_computation	bool	0	This overwrites the dostaticcomputation flag and activates eigenmode computation on button START.
SolverOptions.Eigensolver.reuse_last_eigenvectors	bool	0	Reuse eigenvectors from last computation (faster, but might be eigenvectors from different system).

SolverOptions.Eigensolver. n_eigvals	integer	3	Number of eigenvalues and eigenmodes to be computed for sparse iterative methods.
SolverOptions.Eigensolver. max_iterations	integer	1000	Maximum number of iterations for iterative eigenvalue solver.
SolverOptions.Eigensolver. solver_type	integer	0	Solvertype for eigenvalue computations: 0..direct (LAPACK), 1..Arnoldi (Matlab), 2..LOBPCG (HotInt).
SolverOptions.Eigensolver. n_zero_modes	integer	0	Number of zero eigenvalues (convergence check).
SolverOptions.Eigensolver. use_n_zero_modes	bool	0	Check convergence for zero eigenvalues.
SolverOptions.Eigensolver. use_preconditioning	bool	0	Use preconditioner inv(K + lambda M)
SolverOptions.Eigensolver. accuracy	double	1e-010	Tolerance for iterative Eigenvalue solver.
SolverOptions.Eigensolver. preconditioner_lambda	double	1	lambda for preconditioner inv(K + lambda M)
SolverOptions.Eigensolver. eigenmodes_scaling_factor	double	1	scaling factor for the eigenmodes
SolverOptions.Eigensolver. eigenmodes_normalization_mode	integer	0	0 (standard)... max(v) = 1 , 1.. v'v = 1
SolverOptions.Eigensolver.linearize_about_actual_solution	bool	0	Use actual solution as configuration for linearization of K/M
SolverOptions.Eigensolver. use_gyroscopic_terms	bool	0	Use gyroscopy terms for Eigenvalue computation
SolverOptions.Eigensolver. eigval_outp_format_flag	integer	3	print (bitwise sum) 1 .. eigenfreq., 2 .. eigenvec., 4 .. eigenfreq. in Hz (otherwise in rad/s)

SolverOptions.Linalg

SolverOptions.Linalg. use_sparse_solver	bool	0	1|(0) ... Sparse Jacobian and sparse solver is (not)activated.
SolverOptions.Linalg. undetermined_system	bool	0	1|(0) ... Solve system which is overdetermined (least squares solution) or underdetermined (minimum norm solution) via LAPACK routine dgels.
SolverOptions.Linalg. estimated_condition_number	double	1e+012	Used for considering equations to be linearly dependent when solving undetermined systems. Use together with option undetermined_system.

SolverOptions.Discontinuous

SolverOptions.Discontinuous. absolute_accuracy	double	0.0001	Accuracy for discontinuous problems (plasticity, contact, friction, ...).
SolverOptions.Discontinuous. max_iterations	integer	8	Max. number of iterations for discont. problems.
SolverOptions.Discontinuous. ignore_max_iterations	bool	0	continue anyway if error goal is not reached after max discontinuous iterations

SolverOptions.Solution

SolverOptions.Solution. write_solution	bool	1	(0)|1 ... (Don't) write results to file.
SolverOptions.Solution. write_solution_every_x_step	integer	1	Write solution every xx steps.
SolverOptions.Solution. immediately_write_file	bool	1	1 ... SLOW: immediately write data to file with '« flush' (no buffering), 0=FAST
SolverOptions.Solution. always_replace_files	bool	0	1 = always replace files, 0 = append solution to files

SolverOptions.Solution.SolutionFile

SolverOptions.Solution. SolutionFile. write_solution_file_header	bool	1	Write solution file header.

SolverOptions.Solution.SolutionFile.solution_file_header_comment	string	""	Comment written in solution file header.
SolverOptions.Solution.SolutionFile.output_filename	string	"sol.txt"	Filename for general solution file (sensor output).
SolverOptions.Solution.ParameterFile			
SolverOptions.Solution.ParameterFile.parameter_variation_filename	string	"solpar.txt"	Filename for parameter variation solution file.
SolverOptions.Solution.ParameterFile.write_final_sensor_values	bool	1	Write final sensor values into parameter file.
SolverOptions.Solution.ParameterFile.write_cost_function	bool	1	Write cost function of sensors into parameter file.
SolverOptions.Solution.ParameterFile.write_second_order_size	bool	0	Write second order size into parameter file.
SolverOptions.Solution.ParameterFile.write_CPU_time	bool	0	Write CPU-time into parameter file.
SolverOptions.Solution.store_solution_state	integer	0	Store final solution state in file.
SolverOptions.Solution.store_solution_state_name	string	""	Filename for final solution state storage.
SolverOptions.Solution.load_solution_state	integer	0	Load initial configuration from file.
SolverOptions.Solution.load_solution_state_name	string	""	Filename for initial configuration.
SolverOptions.Solution.Sensor			
SolverOptions.Solution.Sensor.postproc_compute_eigenvalues	bool	0	Compute eigenvalues in postprocessing.
SolverOptions.Element			
SolverOptions.Element.store_finite_elements_matrices	bool	1	Store intermediate matrices for finite elements (faster, but uses huge memory).
SolverOptions.Element.element_wise_jacobian	bool	1	Jacobian is computed only for each element, taking into account known couplings.
SolverOptions.ParameterVariation			
SolverOptions.ParameterVariation.activate	bool	0	Do multiple computations by varying a parameter in a certain range.
SolverOptions.ParameterVariation.geometric	bool	0	Vary parameter geometrically (a*x, a*a*x, a*a*a*x, ...).
SolverOptions.ParameterVariation.start_value	double	0	Start value for parameter variation.
SolverOptions.ParameterVariation.end_value	double	0	Final value for parameter variation.
SolverOptions.ParameterVariation.arithmetic_step	double	1	Arithmetic step size for parameter variation.
SolverOptions.ParameterVariation.geometric_step	double	2	Geometric factor for parameter variation.
SolverOptions.ParameterVariation.MBS_EDC_variable_name	string	""	Path and variablename in MBS EDC which shall be varied in parameter variation.
SolverOptions.ParameterVariation.Var2			

SolverOptions.ParameterVariation.Var2.activate	bool	0	Do multiple computations by varying a parameter in a certain range.
SolverOptions.ParameterVariation.Var2.geometric	bool	0	Vary parameter geometrically (a*x, a*a*x, a*a*a*x, ...).
SolverOptions.ParameterVariation.Var2.start_value	double	0	Start value for parameter variation.
SolverOptions.ParameterVariation.Var2.end_value	double	0	Final value for parameter variation.
SolverOptions.ParameterVariation.Var2.arithmetic_step	double	1	Arithmetic step size for parameter variation.
SolverOptions.ParameterVariation.Var2.geometric_step	double	2	Geometric factor for parameter variation.
SolverOptions.ParameterVariation.Var2.MBS_EDC_variable_name	string	""	Path and variablename in MBS EDC which shall be varied in parameter variation.

SolverOptions.Optimization

SolverOptions.Optimization.activate	bool	0	Do multiple computations by genetic optimization of parameter(s) in a certain range.
SolverOptions.Optimization.run_with_nominal_parameters	bool	0	(0)1 ... (Don't) perform single simulation with nominal parameters.
SolverOptions.Optimization.sensors	integer	0	Define sensor number(s) here; (the sum of) the end value(s) of the sensor signal time history is defined as cost function. The use of more than one sensor is planned.
SolverOptions.Optimization.restart	bool	0	(0)1...(Don't) continue parameters optimization based on existing parameter file. 0..create new parameter file, 1..append to existing parameter file.
SolverOptions.Optimization.method	string	"Genetic"	Genetic: optimize using random parameters, best parameters are further tracked.

SolverOptions.Optimization.Parameters

SolverOptions.Optimization.Parameters.number_of_params	integer	0	Number of parameters to optimize.
SolverOptions.Optimization.Parameters.param_name1	string	""	Parameter name.
SolverOptions.Optimization.Parameters.param_minval1	double	0	Lower limit of parameter.
SolverOptions.Optimization.Parameters.param_maxval1	double	0	Upper limit of parameter.
SolverOptions.Optimization.Parameters.param_name2	string	""	Parameter name.
SolverOptions.Optimization.Parameters.param_minval2	double	0	Lower limit of parameter.
SolverOptions.Optimization.Parameters.param_maxval2	double	0	Upper limit of parameter.
SolverOptions.Optimization.Parameters.param_name3	string	""	Parameter name.
SolverOptions.Optimization.Parameters.param_minval3	double	0	Lower limit of parameter.
SolverOptions.Optimization.Parameters.param_maxval3	double	0	Upper limit of parameter.

SolverOptions.Optimization. Parameters.param_name4	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval4	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval4	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name5	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval5	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval5	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name6	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval6	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval6	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name7	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval7	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval7	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name8	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval8	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval8	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name9	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval9	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval9	double	0	Upper limit of parameter.
SolverOptions.Optimization. Parameters.param_name10	string	""	Parameter name.
SolverOptions.Optimization. Parameters.param_minval10	double	0	Lower limit of parameter.
SolverOptions.Optimization. Parameters.param_maxval10	double	0	Upper limit of parameter.

SolverOptions.Optimization.Newton

SolverOptions.Optimization. Newton. random_starting_values	bool	0	set to 1 if the ’surviving_population_size’ best value(s) of shooting with ’initial_population_size’ different parameter sets should be used as starting values for Newton’s method
SolverOptions.Optimization. Newton.param_epsilon_abs	double	1e-006	Absolute value D for numerical computation of dx=K*x + D (==>f’(x) = df/dx).
SolverOptions.Optimization. Newton.param_epsilon_rel	double	0.0001	Relative value K for numerical computation of dx=K*x + D (==>f’(x) = df/dx).
SolverOptions.Optimization. Newton. max_number_of_iterations	integer	5	Maximal number of newton iterations.
SolverOptions.Optimization. Newton.absolute_accuracy	double	1e-006	Absolute accuracy.

SolverOptions.Optimization. Newton.use_param_limits	integer	0	0...no limit of optimized parameter values, 1...use param_[min|max]val, 2...assume all parameters positive, -1...assume all parameters negative.
SolverOptions.Optimization. Newton. initial_population_size	integer	1	Shooting: number of initial parameter sets initial evaluations of the cost function (randomly between range Parameters.minval and Parameters.maxval)
SolverOptions.Optimization. Newton. surviving_population_size	integer	1	Number of starting parameter sets for Newton's method. This number defines how often Newton's method should be started.

SolverOptions.Optimization.Genetic

SolverOptions.Optimization. Genetic.initial_population_size	integer	20	Size of initial trial values; also used for random Newton initialization.
SolverOptions.Optimization. Genetic. surviving_population_size	integer	10	Size of values which are further tracked; also used for random Newton initialization.
SolverOptions.Optimization. Genetic.number_of_children	integer	10	Number of children of surviving population.
SolverOptions.Optimization. Genetic. number_of_generations	integer	15	Number of generations in genetic optimization.
SolverOptions.Optimization. Genetic. range_reduction_factor	double	0.5	Reduction of range of possible mutations.
SolverOptions.Optimization. Genetic. randomizer_initialization	double	0	Initialization of random function.
SolverOptions.Optimization. Genetic. min_allowed_distance_factor	double	0.5	Set to value greater than zero (distance is allowed radius of (hyper-)sphere in the normed parameter space (min=0)). Only the best parameter in the inner of the (hyper-)sphere is furtile.

SolverOptions.Sensitivity

SolverOptions.Sensitivity. activate	integer	0	(0)1...(Don't) analyze sensitivity of sensor values with respect to parameters.
SolverOptions.Sensitivity. method	string	"Forward"	df/dx: Forward: use forward difference, Backward: use backward difference, Central: use central difference.
SolverOptions.Sensitivity. num_diff_parameter_absolute	double	0.0001	Absolute value D for computation of df/dx, dx=K*x+D.
SolverOptions.Sensitivity. num_diff_parameter_relative	double	0.0001	Relative factor K for computation of df/dx, dx=K*x+D.
SolverOptions.Sensitivity. use_final_sensor_values	bool	0	(0)1...(Don't) use final sensor values.
SolverOptions.Sensitivity. use_optimization_parameters	bool	0	1|(0) ... (Don't) get parameters from Optimization.Parameters.

SolverOptions.Sensitivity.Parameters

SolverOptions.Sensitivity. Parameters. number_of_params	integer	0	Number of parameters.
SolverOptions.Sensitivity. Parameters.param_name1	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name2	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name3	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name4	string	""	Parameter name.

SolverOptions.Sensitivity. Parameters.param_name5	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name6	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name7	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name8	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name9	string	""	Parameter name.
SolverOptions.Sensitivity. Parameters.param_name10	string	""	Parameter name.

3.13.2 LoggingOptions

Data objects of LoggingOptions:

Data name	**type**	**default**	**description**
Solver			
Solver.general_information	bool	0	Print general solver information. This includes: Newtons relative error goal, contractivity, iteration error, and qualitative information about Jacobian-updates, as well as iteration error and number of newton iterations at each post newton step, and post newton iterations at each time step.
Solver.newton_iteration_jacobi_condition	bool	0	Print condition number of Jacobi matrix in Newtons method whenever it is updated.
Solver.newton_iteration_jacobi_matrix	bool	0	Print Jacobi matrix of Newtons method whenever it is updated.
Solver.newton_iteration_residual_vector	bool	0	Print iterated residual vector at each Newton step.
Solver.newton_iteration_solution_vector	bool	0	Print iterated solution vector at each Newton step.
Solver. post_newton_iteration_data_vector	bool	0	Print data vector at each nonlinear iteration step.
Solver. step_solution_vector_increment	bool	0	Print solution increment of each step (dynamic simulation: time step, static simulation: load step).
Solver.step_solution_vector	bool	0	Print solution vector of each step (dynamic simulation: time step, static simulation: load step).
EDCParser			
EDCParser. general_information	bool	0	Print general information on parsed objects (e.g., while reading modeldata or configuration files).
output_level	integer	6	0..no output; 1..necessary output (Errors, start/end simulation); 2..almost necessary output (Warnings); 3..multiple simulation output (parameter variation/optimization); 4..simulation output (solver); 5..extended output (useful information); 6..complete information; 7..debug level 1; 8..debug level 2; 9..max output.
output_precision_double	integer	8	number of significant digits of a double in output window and logfile.
output_precision_vector	integer	6	number of significant digits of a vector in output window and logfile.
output_precision_matrix	integer	10	number of significant digits of a matrix in output window and logfile.

max_error_messages	integer	100	Number of displayed error messages.
max_warning_messages	integer	100	Number of displayed warning messages.
computation_output_every_x_sec	double	2	Write computation output every x seconds; notice: if solver logs are printed, then this option does not take effect.
write_mass_and_stiffness_matrix	bool	0	Write the initial mass and stiffness matrices in Matlab format to files Mmat.dat and Kmat.dat, in Matlab directory.
default_log_filename	string	"hotint.log"	Default filename for hotint log file.
critical_log_file_size	double	10	critical log file size, after which a warning is displayed; in megabytes.
file_output_level	integer	7	0..no output; 1..necessary output (Errors, start/end simulation); 2..almost necessary output (Warnings); 3..multiple simulation output (parameter variation/optimization); 4..simulation output (solver); 5..extended output (useful information); 6..complete information; 7..debug level 1; 8..debug level 2; 9..max output.

3.13.3 GeneralOptions

Data objects of GeneralOptions:

Data name	**type**	**default**	**description**
Application			
Application. close_application_when_finished	bool	0	1|(0) ... (Don't) automatically close application after computation.
Application. show_hotint_window	bool	1	1|(0) ... 1 .. show HOTINT window (minimized).
Application. start_computation_automatically	bool	0	immediately start computation on program start
Application.slim_menu	integer	0	0..full menu, otherwise several menu items removed.
Application.reload_last_model	bool	0	1|(0) ... (Don't) reload the last saved model on program start
Application.activate_autosave	bool	1	1|(0) ... (Don't) save the model automatically before each change of an object
Paths			
Paths.application_path	string	"d:\cpp3\HotInt_V1\hotintx64\Release\"	Path of the application.
Paths. hotint_input_data_path	string	""	Path of Hotint Input Data file.
Paths.relative_paths_relative_to_application	bool	1	1.. relative paths are relative to hotint.exe(application_path)| 0..relative to hid-file (hotint_input_data_path)
Paths.single_image_path	string	""	Path to store single images (record frame dialog)
Paths.video_image_path	string	""	Path to store video images series (recoed frame dialog)
Paths.plottool_image_path	string	""	Path to store plottool images (plottool dialog
Paths.sensor_output_path	string	"..\..\output\"	Relative or absolute path to output directory.
ModelFile			
ModelFile. hotint_input_data_filename	string	""	Name of Hotint Input Data file.
ModelFile.internal_model_function_name	string	"Generate Tex Files For Docu"	Name of internal model function (cpp).

ModelFile.recent_file1	string	""	Recent file filename 1.
ModelFile.recent_file2	string	""	Recent file filename 2.
ModelFile.recent_file3	string	""	Recent file filename 3.
ModelFile.recent_file4	string	""	Recent file filename 4.
ModelFile.recent_file5	string	""	Recent file filename 5.
ModelFile.accept_txt_file_as_model_file	bool	1	Enable this function to allow 'hotint_input_data_filename' with ending '.txt' as first argument (for drag and drop).

Measurement

Measurement.use_degrees	bool	1	1|(0) ... (Don't) use degrees instead of radiant in edit dialogs for bodies and joints.
Measurement.angle_mode	integer	0	Rotation input mode: 0=Euler angles, 1=Rotation X/Y/Z, 2=Euler parameters.
Measurement.units_of_legend	integer	0	Units of legend: 0=SI(m,N, etc.); 1=mm, N, etc.

OutputWindow

OutputWindow. max_text_length	integer	50000	Maximum text length (number of characters) for output text window, use -1 for no limit

SavedViewingOptions

SavedViewingOptions. filename_1	string	"savedViewingOptions1.txt"	Filename (including absolute path if not equal to application_path) of saved viewing options accessable via button '1' in GUI. (Use Ctrl+Click on this button to store the viewing options)
SavedViewingOptions. filename_2	string	"savedViewingOptions2.txt"	Filename (including absolute path if not equal to application_path) of saved viewing options accessable via button '2' in GUI. (Use Ctrl+Click on this button to store the viewing options)
SavedViewingOptions. filename_3	string	"savedViewingOptions3.txt"	Filename (including absolute path if not equal to application_path) of saved viewing options accessable via button '3' in GUI. (Use Ctrl+Click on this button to store the viewing options)

3.13.4 ViewingOptions

Data objects of ViewingOptions:

Data name	**type**	**default**	**description**

Animation

Animation. animate_from_beginning	bool	1	1|(0) ... (Don't) start animation from beginning.
Animation. animate_every_N_frame	integer	1	Animation frames: show every N'th frame at animation.
Animation. animate_deformation	bool	0	1|(0) ... (Don't) animate deformation scaling.
Animation. animate_deformation_once	bool	0	1 ... animate deformation (eigenmodes) only for one cycle - for recording; 0 ... endless animate

Animation.RecordSingleFrames

Animation.RecordSingleFrames. record	bool	0	1|(0) ... (Don't) record frames
Animation.RecordSingleFrames. show_frame_numbers	bool	0	1|(0) ... (Don't) show frame numbers in images);
Animation.RecordSingleFrames. record_every_x_frame	integer	1	record every x frames

Animation.RecordSingleFrames.single_file_name	string	"snapshot"	name of the single frame file without extensions
Animation.RecordSingleFrames.video_file_name	string	"frame"	name of the video frame file without extensions and number
Animation.RecordSingleFrames.default_image_format	integer	0	format of the exported file (default setting for radiobutton) 0..JPG, 1..PNG, 2..BMP
Animation.RecordSingleFrames.include_output_window	bool	0	includes the output window to the screenshot
Misc			
Misc.redraw_frequency	integer	4	Redraw frequency: 0..off, 1..draw last frame, 2..100sec, 3..20sec, 4..2sec, 5..200ms, 6..50ms, 7..20ms, 8..every 10 frames, 9..every frame.
Misc.global_line_thickness	double	1	Global_line_thickness (coord system, etc.) ****.
Misc.global_point_size	double	2	Global point size (coord system, grid, etc.) ****.
Misc.show_3D_text_in_front	bool	1	1|(0) ... (Don't) show 3D texts in front.
Misc.axes_position	integer	0	position of axes: bottom left (0), bottom right (1), top right (2), top left (3), center (4), no axes (5)
Misc.lock_rotation	bool	0	lock rotation of model (for 2D models)
GeomElements			
GeomElements.line_thickness	double	2	GeomElement (outline) line thickness ****.
Origin			
Origin.show	bool	1	1|(0) ... (Don't) draw coordinate system in origin (X0, Y0, Z0).
Origin.size_of_origin	double	0.5	Size of origin.
Grid			
Grid.show	integer	0	Show Grid and Background planes (add up), 1=XY, 2=XZ, 4=YZ.
Grid.pos_x	double	0	X-position for interesction point of planes
Grid.pos_y	double	0	Y-position for interesction point of planes
Grid.pos_z	double	0	Z-position for interesction point of planes
Grid.size_1	double	2	X-size of background plane
Grid.size_2	double	2	Y-size of background plane
Grid.size_3	double	2	Z-size of background plane
Grid.step_1	double	0.1	Grid discretization X-direction
Grid.step_2	double	0.1	Grid discretization Y-direction
Grid.step_3	double	0.1	Grid discretization Z-direction
Grid.Colors			
Grid.Colors.transparency_factor	double	0.1	Transparency factor for the backgruond planes
Grid.Colors.plane1_col_r	double	0.85	Red color channel for XY plane
Grid.Colors.plane1_col_g	double	0.85	Green color channel for XY plane
Grid.Colors.plane1_col_b	double	0.85	Blue color channel for XY plane
Grid.Colors.plane2_col_r	double	0.95	Red color channel for XZ plane
Grid.Colors.plane2_col_g	double	0.95	Green color channel for XZ plane
Grid.Colors.plane2_col_b	double	0.95	Blue color channel for XZ plane
Grid.Colors.plane3_col_r	double	0.95	Red color channel for YZ plane
Grid.Colors.plane3_col_g	double	0.95	Green color channel for YZ plane
Grid.Colors.plane3_col_b	double	0.95	Blue color channel for YZ plane
CuttingPlane			
CuttingPlane.1			
CuttingPlane.1.activate	bool	0	1|(0) ... Use (Don't use) cutting plane.
CuttingPlane.1.normal_X	double	1	Cutting plane normal-X.
CuttingPlane.1.normal_Y	double	0	Cutting plane normal-Y.
CuttingPlane.1.normal_Z	double	0	Cutting plane normal-Z.
CuttingPlane.1.distance	double	0	Cutting plane distance.
CuttingPlane.2			
CuttingPlane.2.activate	bool	0	1|(0) ... Use (Don't use) cutting plane.

CuttingPlane.2.normal_X	double	1	Cutting plane 2 normal-X.
CuttingPlane.2.normal_Y	double	0	Cutting plane 2 normal-Y.
CuttingPlane.2.normal_Z	double	0	Cutting plane 2 normal-Z.
CuttingPlane.2.distance	double	0	Cutting plane 2 distance.
CuttingPlane.cut_bodies	bool	1	1|(0) ... (Don't) cut bodies.
CuttingPlane. cut_bodies_altshapes	bool	1	1|(0) ... (Don't) cut alternative shapes of bodies.
CuttingPlane.cut_ground	bool	1	1|(0) ... (Don't) cut background.
CuttingPlane. cut_whole_scene_by_open_gl	bool	0	1|(0) ... (Don't) use OpenGL for handling cutting planes.
CuttingPlane.nosurfaceupdate	bool	0	use of cutting plane does trigger a change of drawn surfaceelements

StandardView

StandardView. angle_rot_axis_1	integer	1	Rotation axis for standard view angle_1 (rotation axis 1, 2 or 3).
StandardView. angle_rot_axis_2	integer	2	Rotation axis for standard view angle_2 (rotation axis 1, 2 or 3).
StandardView. angle_rot_axis_3	integer	3	Rotation axis for standard view angle_3 (rotation axis 1, 2 or 3).
StandardView.angle_1	double	0	Standard view angle_1.
StandardView.angle_2	double	0	Standard view angle_2.
StandardView.angle_3	double	0	Standard view angle_3.

Bodies

Bodies.Rigid

Bodies.Rigid.show_outline	bool	1	1|(0) ... (Don't) show bodies outline.
Bodies.Rigid.show_faces	bool	1	1|(0) ... (Don't) show bodies faces.
Bodies.Rigid.line_thickness	double	1	Rigid body (outline) line thickness ****.not used yet.
Bodies.Rigid. draw_center_of_gravity	bool	1	1|(0) ... (Don't) draw center of gravity
Bodies.Rigid.draw_resolution	integer	12	Draw resolution for Rigid3D.
Bodies.Rigid.COG_sizefactor	double	1	Cog_factor for Rigid3D (default: 1).
Bodies.show_element_numbers	bool	0	1|(0) ... (Don't) show element body numbers.
Bodies.show_local_frame	bool	0	1|(0) ... (Don't) show local body frame.
Bodies.transparent	bool	1	1|(0) ... (Don't) draw bodies transparent.
Bodies.local_frame_size	double	0	Body local frame size.
Bodies. deformation_scale_factor	double	1	Deformation scale factor.
Bodies. scale_rigid_body_displacements	integer	0	1|(0) ... (Don't) use deformation scale factor in animation.
Bodies.show_velocity_vector	bool	0	1|(0) ... (Don't) show velocity vector, e.g. for particles.
Bodies.velocity_vector_just_for_particles	bool	0	1|(0) ... (Don't) show velocity vector for particles only.
Bodies. velocity_vector_scaling_mode	integer	1	1: constant scaling (a), 2: linear scaling (ax), 3: exponential scaling (a(1-exp(-x/b))).
Bodies. velocity_vector_scaling_a	double	1	magnification factor; e.g., if velocity_vector_scaling_mode == 2: velocity vector length = v*velocity_vector_scaling_a.
Bodies. velocity_vector_scaling_b	double	1	knee factor; e.g., if velocity_vector_scaling_mode == 3: velocity vector length = velocity_vector_scaling_a*(1-exp(-v/velocity_vector_scaling_b)).
Bodies.velocity_vector_scaling_thickness	double	1	thickness scaling factor; independent from mode.

Bodies.Particles

Bodies.Particles. displacement_scale_factor	double	1	factor for scaling the displacements.
Bodies.Particles. draw_size_factor	double	1	factor for adjusting the size of particles while drawing.
Bodies.Particles. draw_every_nth	integer	1	draw every n-th particle only.
FiniteElements			
FiniteElements.Contour			
FiniteElements.Contour. activate	bool	1	1|(0) ... (Don't) show solution in mesh as contour plot.
FiniteElements.Contour. max_stress_active	bool	0	1|(0) ... Max. stress is (not) updated during computation.
FiniteElements.Contour. max_stress	double	0	Value of max. stress.
FiniteElements.Contour. min_stress_active	bool	0	1|(0) ... Min. stress is (not) updated during computation.
FiniteElements.Contour. min_stress	double	0	Value of min. stress.
FiniteElements.Contour. post_processing_variable_name	string	""	Name of the field variable, which is currently selected for contour plotting.
FiniteElements.Contour. variable_range_auto_update	bool	0	1|(0) ... (Don't) update the range of the variable each time a new scene is plotted.
FiniteElements.Contour. color_tiling	integer	10	Color tiling (used for FE-color texture).
FiniteElements.Contour. label_precision	integer	3	number of digits for the numbers in label.
FiniteElements.Contour. plot_interpolated	bool	0	1|(0) ... (Don't) draw Stress/strain/etc. interpolated at nodes.
FiniteElements.Contour. grey_mode	bool	0	1|(0) ... (Don't) draw grey colors for finite elements.
FiniteElements.Contour. invert_colors	bool	0	1|(0) ... (Don't) invert colors.
FiniteElements.Contour. nonlinear_color_legend	bool	0	1|(0) ... (Don't) create nonlinear distributed color legend.
FiniteElements.Contour. hide_legend	bool	0	1|(0) ... (Don't) hide color legend.
FiniteElements.Contour. axis_tiling	integer	16	Axis tiling (for element face and outline, beams and plates).
FiniteElements.Contour. resolution_axis	integer	8	Axis resolution: contour plot resolution along axis, beams and plates.
FiniteElements.Contour. resolution_cross_section	integer	4	Cross-section resolution: contour plot resolution at cross-section, beams and plates.
FiniteElements.Contour. resolution_solid_elements	integer	2	Contour plot resolution for solid finite elements.
FiniteElements.Nodes			
FiniteElements.Nodes.show	bool	1	1|(0) ... (Don't) draw nodes.
FiniteElements.Nodes. show_node_numbers	bool	0	1|(0) ... (Don't) show node numbers.
FiniteElements.Nodes. node_resolution	integer	3	Node resolution for drawing.
FiniteElements.Nodes. node_size	double	0.001	Draw node size.
FiniteElements.Mesh			
FiniteElements.Mesh.show	bool	1	1|(0) ... (Don't) show mesh of finite element.
FiniteElements.Mesh. draw_flat_elements	bool	0	1|(0) ... (Don't) draw Plate elements flat, only mid-plane (view from top only).

FiniteElements.Mesh. draw_only_surface_elements	bool	1	1|(0) ... (Don't) draw surface elements only.
FiniteElements.Mesh. element_line_thickness	double	1	Finite element line thickness (outline of 2D and 3D beam, plate).
FiniteElements.Mesh. shrinking_factor	double	1	Shrinking factor.

Connectors

Connectors.show_constraints	bool	1	1|(0) ... (Don't) show joints/connectors.
Connectors. show_control_elements	bool	0	1|(0) ... (Don't) draw control elements in 3D Window..
Connectors. show_constraint_numbers	bool	0	1|(0) ... (Don't) show constraint number.
Connectors.show_faces	bool	1	1|(0) ... (Don't) show constraint faces –> show constraint faces.
Connectors.transparent	bool	1	1|(0) ... (Don't) draw constraints transparent.
Connectors.draw_outline	bool	0	1|(0) ... (Don't) draw constraints outline **** –> faces is IOption 114.not used yet.
Connectors.line_thickness	double	1	Constraint (outline) line thickness ****.not used yet.

Connectors.Contact

Connectors.Contact. show_contact_as_circle	bool	1	1|(0) ... (Don't) draw circles at contact of bodies.
Connectors.Contact. show_contact_points	bool	1	1|(0) ... (Don't) show contact points.
Connectors. global_draw_scalar_size	double	0.1	global scalar constraint draw size (e.g.radius)
Connectors. global_draw_resolution	double	16	global constraint draw resolution
Connectors.Autosize	bool	0	1|(0) Autogenerate a global scalar constraint draw size

Loads

Loads.show_loads	bool	0	1|(0) ... (Don't) show loads.
Loads.arrow_size	double	0.1	Size of arrow for drawing of loads.
Loads.color_red	double	0.6	Red-value for drawing of loads (use values between 0. and 1.).
Loads.color_green	double	0.6	Green-value for drawing of loads (use values between 0. and 1.).
Loads.color_blue	double	0	Blue-value for drawing of loads (use values between 0. and 1.).

Sensors

Sensors.show_sensors	bool	0	1|(0) ... (Don't) show sensors.
Sensors.transparent	bool	1	1|(0) ... (Don't) draw sensors transparent.
Sensors.sensor_origin_size	double	0.2	Sensor origin size.

OpenGL

OpenGL.enable_lighting	bool	1	OpenGL lighting.
OpenGL.smooth_model	bool	1	OpenGL SMOOTH ShadeModel smooth.
OpenGL. immediate_apply_dialog	bool	1	Immediate apply in openGL dialog.
OpenGL.global_culling	integer	0	OpenGL cull (means: exclude) 1=front 2=back or 3=both views on faces of polygons; 0=don't cull any view.
OpenGL.global_transparency	double	0.8	Global transparency for SetColor, 1=no translucency, 0=fully transparent.
OpenGL.material_shininess	double	60	Material shininess (0..128).
OpenGL. material_color_intensity	double	1	Material specular color intensity.

OpenGL.Light1

OpenGL.Light1.enable	bool	1	OpenGL enable light1.
OpenGL.Light1. use_light_position	bool	0	OpenGL light1 mode (0=standard, 1=use light position).
OpenGL.Light1.ambient	double	0.25	Light1 ambient parameter.
OpenGL.Light1.diffuse	double	0.4	Light1 diffuse parameter.
OpenGL.Light1.specular	double	0.4	Light1 specular parameter.
OpenGL.Light1.pos_x	double	1	Light1 posx.
OpenGL.Light1.pos_y	double	1	Light1 posy.
OpenGL.Light1.pos_z	double	-1	Light1 posz.

OpenGL.Light2

OpenGL.Light2.enable	bool	1	OpenGL enable light2.
OpenGL.Light2. use_light_position	bool	0	OpenGL light2 mode (0=standard, 1=use light position).
OpenGL.Light2.ambient	double	0.25	Light2 ambient parameter.
OpenGL.Light2.diffuse	double	0.4	Light2 diffuse parameter.
OpenGL.Light2.specular	double	0	Light2 specular parameter.
OpenGL.Light2.pos_x	double	0	Light2 posx.
OpenGL.Light2.pos_y	double	3	Light2 posy.
OpenGL.Light2.pos_z	double	2	Light2 posz.

ApplicationWindow

ApplicationWindow.rect_left	integer	795	left coordinate of application window
ApplicationWindow.rect_top	integer	595	top coordinate of application window
ApplicationWindow.rect_width	integer	700	width of application window
ApplicationWindow. rect_height	integer	700	left coordinate of application window

DataManager

DataManager.dialog_open	bool	1	open data manager on startup
DataManager. store_data_to_files	bool	0	if checked, then solution data (for data manager) is stored in files, instead of memory; these files are located in subdirectory solution_data of 'GeneralOptions.Paths.sensor_output_path'.
DataManager. store_data_every	double	0.01	Store data with data-manager, redraw and create animations: # -2 == always, -1 == at max stepsize, 0 = never, x.x = at every time x.x.

OutputWindow

OutputWindow.dialog_open	bool	1	open output dialog on startup
OutputWindow.stored_width	integer	200	stored width of output dialog
OutputWindow. enable_output_text	bool	1	enable output text in output dialog

View3D

View3D.Center_point

View3D.Center_point.xpos	double	0	Centerpoint x-position
View3D.Center_point.ypos	double	0	Centerpoint y-position

View3D.Center_offset

View3D.Center_offset.xpos	double	0	Centeroffset x-position
View3D.Center_offset.ypos	double	0	Centeroffset y-position
View3D.Center_offset.zpos	double	0	Centeroffset z-position
View3D.scene_offset	double	1.5	Scene offset
View3D.zoom_factor	double	2.8	zoom factor of 3D view
View3D.aspect_ratio	double	1	current aspect ratio in OpenGL view
View3D. maximum_scene_coordinates	double	1	stored maximum scene coordinates in OpenGL view

View3D.Mouse

View3D.Mouse. zoom_factor_mouse_inc	double	0.005	increment of zoom factor per mousemove of 3D view on mousemove
View3D.Mouse. translation_mouse_inc	double	0.01	increment of translation per mousemove of 3D view on mousemove

View3D.Mouse. rotation_mouse_inc	double	0.5	increment of rotation per mousemove of 3D view on mousemove
View3D.Mouse. perspective_mouse_inc	double	0.01	increment of perspective per mousemove of 3D view on mousemove
View3D.ModelViewMatrix			
View3D.ModelViewMatrix.a11	double	1	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a12	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a13	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a21	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a22	double	1	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a23	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a31	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a32	double	0	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.a33	double	-1	rotation parameter in open GL model view matrix
View3D.ModelViewMatrix.tx	double	0	translation parameter in open GL model view matrix
View3D.ModelViewMatrix.ty	double	0	translation parameter in open GL model view matrix
View3D.ModelViewMatrix.tz	double	0	translation parameter in open GL model view matrix

3.13.5 PlotToolOptions

Data objects of PlotToolOptions:

Data name	**type**	**default**	**description**
auto_redraw	bool	0	1|(0) .. (Don't) redraw in regular intervals
auto_redraw_interval	double	30	redraw every x seconds
auto_rescale	bool	1	1|(0) .. (Don't) rescale to fully fit the whole data when updated
title_size_factor	double	1.25	factor to in-/decrease font size of title in respect to axis font
ticks_size_factor	double	0.7	factor to in-/decrease font size of ticks in respect to axis font
line_thickness_border	integer	4	line thickness (border) in logical points
line_thickness_factor	double	1	scaling factor for all plotted lines
status_bar_info	bool	0	1|(0) .. (Don't) show status bar information.
DataPoints			
DataPoints. flag_draw_every_nth	bool	0	1|(0) .. (Don't) skip several datapoints in draw routine)
DataPoints.draw_every_nth	integer	100	draw every nth datapoint (0 for every)
DataPoints. flag_mark_every_nth	bool	0	1|(0) .. (Don't) skip marking of several datapoints in draw routine)
DataPoints.mark_every_nth	integer	100	mark every nth datapoint (0 for every)
DataPoints.vertical_marker	bool	0	1|(0) .. (Don't)mark current time in plot with a special marker
DataPoints. draw_only_to_time	bool	0	1|(0) .. (Don't) draw the data only up to the time from datamanager
DataPoints.use_time_interval	bool	0	1|(0) .. (Don't) use t_min and t_max as boundaries for drawing in plottool
DataPoints.t_min	double	0	Lower boundary for time interval plot. Only used if use_time_interval = 1.
DataPoints.t_max	double	0	Upper boundary for time interval plot. Only used if use_time_interval = 1.
View			

View.initial_size_horizontal	integer	640	initial size of the CView holding the plot
View.initial_size_vertical	integer	480	initial size of the CView holding the plot
View.plot_horizontal	integer	3000	size in logical units for the plot - fixed aspect ratio
View.plot_vertical	integer	2000	size in logical units for the plot - fixed aspect ratio
View.distance_left	double	15	surplus in %plotwidth from left border of the plot to left border of the window
View.distance_top	double	15	surplus in %plotheight from upper border of the plot to the upper border of the window
View.distance_bottom	double	20	surplus in %plotheight from lower border of the plot to the lower border of the window
View.distance_right	double	15	surplus in %plotheight from lower border of the plot to the lower border of the window
Watches			
Watches.initial_size_horizontal	integer	300	initial size of the CView holding the plot
Watches.initial_size_vertical	integer	200	initial size of the CView holding the plot
Axis			
Axis.draw_at_origin	bool	0	1|(0) .. (Don't) draw axis at origin
Axis.label_major	bool	1	1|(0) .. (Don't) write lables for major ticks
Axis.label_minor	bool	1	1|(0) .. (Don't) write lables for minor ticks
Axis.overdraw	double	3	percentage the axis are longer than the graph
Axis.ticksize	double	2	size in percent of major ticks, minor are half size
Axis.minor_ticks_x	integer	0	minor ticks for x-axis
Axis.minor_ticks_y	integer	0	minor ticks for y-axis
Axis.digits_x_labels	integer	3	maximum digits for x-axis labels
Axis.digits_y_labels	integer	3	maximum digits for y-axis labels
Grid			
Grid.shading	double	0.5	Linecolor of grid lines: black if 0, white if 1, and grey scales in between
Grid.linetype_major_x	integer	2	Linetype for major gridlines, x axis (0 = no line, 1 = solid, 2 = dash, 3 = dot)
Grid.linetype_minor_x	integer	3	Linetype for minor gridlines, x axis (0 = no line, 1 = solid, 2 = dash, 3 = dot)
Grid.linetype_major_y	integer	2	Linetype for major gridlines, y axis (0 = no line, 1 = solid, 2 = dash, 3 = dot)
Grid.linetype_minor_y	integer	3	Linetype for minor gridlines, y axis (0 = no line, 1 = solid, 2 = dash, 3 = dot)
Legend			
Legend.show	bool	0	1|(0) .. (Don't) draw axis at origin
Legend.left	double	75	position in % of the legend's left border
Legend.right	double	100	position in % of the legend's right border
Legend.top	double	100	position in % of the legend's upper border
Legend.bottom	double	75	position in % of the legend's lower border
SavePicture			
SavePicture.filename	string	"snap"	filename for the picture without extensions
SavePicture.size_horizontal	integer	1600	size in pixels of the saved BMP
SavePicture.size_vertical	integer	1200	size in pixels of the saved BMP
SavePicture.jpg_quality	integer	10	quality setting for the JPG encoder
SavePicture.store_jpg	bool	1	1|(0) .. (Don't) store image as jpg
SavePicture.store_png	bool	0	1|(0) .. (Don't) store image as png
SavePicture.store_bmp	bool	0	1|(0) .. (Don't) store image as bmp
SavePicture.store_emf	bool	1	1|(0) .. (Don't) store image as emf

Bibliography

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, 1996.

[2] E. Eich-Soellner, C. Führer, *Numerical Methods in Multibody Dynamics*, Teubner, Stuttgart, 1998.

[3] J. Gerstmayr, M. Stangl, *High-Order Implicit Runge-Kutta Methods for Discontinuous Multibody Systems*, Proceedings of the APM 2004, St. Petersburg, Russia, submitted.

[4] J. Gerstmayr, J. Schöberl, *An Implicit Runge-Kutta Based Solver for 3-Dimensional Multibody Systems*, PAMM, Volume 3(1), 2003, pp. 154-155.

[5] E. Hairer and G. Wanner, *Stiff differential equations solved by Radau methods* or *the RADAU5-code*, available via WWW at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau5.f` (1996)

[6] E. Hairer, (Nørsett) and G. Wanner, *Solving ordinary differential equations I (II)*, Springer Verlag Berlin Heidelberg, 1991.

[7] E. Hairer and Ch. Lubich, and M. Roche, *The numerical solution of differential-algebraic systems by Runge-Kutta methods*, Lecture Notes in Math. 1409, Springer–Verlag, (1989).

[8] A. Shabana *Dynamics of Multibody Systems*, Third Edition, Cambridge University Press, 2005.

[9] R. R. Craig Jr. and M. C. C. Bampton, *Coupling of substructures for dynamic analyses*, AIAA Journal, 6(7), pp. 1313Ű1319, 1968

[10] J. Gerstmayr and A. Pechstein, *A generalized component mode synthesis approach for multibody system dynamics leading to constant mass and stiffness matrices*, Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011, Washington, DC, USA, 2011. Paper No. DETC2011/MSNDC-47826, submitted.

[11] Masarati, P, *Direct eigenanalysis of constrained system dynamics*, Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics, 2009.

[12] R. Ludwig and J. Gerstmayr, *Automatic Parameter Identification for Generic Robot Models*, Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics, 2011.

[13] H. Bremer, Elastic Multibody Dynamics, Springer, 2008

[14] Denavit, Jacques; Hartenberg, Richard Scheunemann (1955). A kinematic notation for lower-pair mechanisms based on matrices. Trans ASME J. Appl. Mech 23: 215Ű221.

[15] K. Nachbagauer, P. Gruber, J. Gerstmayr. Structural and Continuum Mechanics Approaches for a 3D Shear Deformable ANCF Beam Finite Element: Application to static and linearized dynamic examples. Journal for Computational and Nonlinear Dynamics, 8, 021004, DOI:10.1115/1.4006787, 2012.

[16] K. Nachbagauer. Development of shear and cross section deformable beam finite elements applied to large deformation and dynamics problems, Johannes Kepler University Linz, 2012.

[17] K. Nachbagauer, P. Gruber, Yu. Vetyukov, J. Gerstmayr. A spatial thin beam finite element based on the absolute nodal coordinate formulation without singularities. Proceedings of the ASME 2011 International Design Engineering Technical Conferences, Computers and Information in Engineering Conference IDETC/CIE 2011, Paper No. DETC2011/MSNDC-47732, Washington, DC, USA, 2011.

[18] P. Gruber, K. Nachbagauer, Yu. Vetyukov, J. Gerstmayr. A novel director-based Bernoulli-Euler beam finite element in absolute nodal coordinate formulation free of geometric singularities. Mechanical Science, 2013 (to appear).

[19] R. Schneiders, *Algorithms for Quadrilateral and Hexahedral Mesh Generation*, www.robertschneiders.de/papers/vki.pdf - Proceedings of the VKI Lecture Series on Computational Fluid Dynamics , 2000.