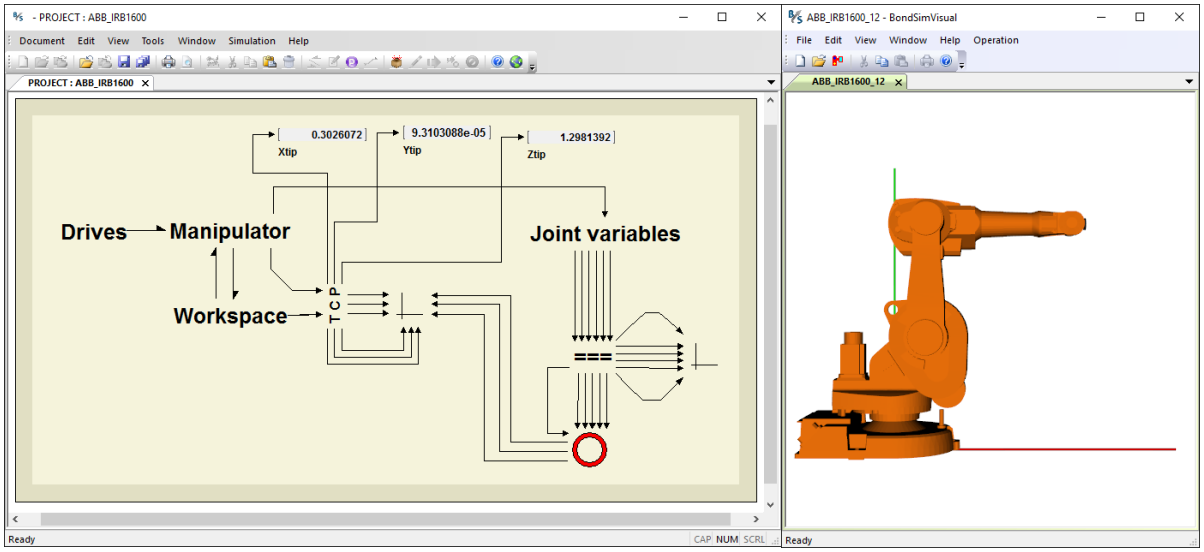
BondSim

.



Prof.dr.sc.Vjekoslav Damic and the BondSim team
University of Dubrovnik, Dubrovnik, Croatia

Table of Contents

Introduction

Welcome to BondSim

Please note that BondSim implements our ideas of Bond Graphs and modeling mechatronic systems by Bond Graphs. Thus, it possible, that it differs from how the other authors treat the Bond Graphs. It is described in much more details in the reference part of this manual and in particular in our books.

The main difference concerns the causality issue. In common Bond Graphs the modeling and the causality assignment is closely related. As is well known the causality assignment defines what are in the Bond Graphs the inputs and what are the outputs. If it is not possible to achieve this then the conventional Bond Graph modeling fails. Often the models are changed to archive this. Our approach is quite different. We separate the modelling of the model solving. We use the power and signal flow through the system and the physical reasoning to lead us during the model development. We try to develop the models as similar to the physical systems as is possible or reasonable. But, we do not use the causalities. Hence, the resulting models are not abstract state space models, but robust models composed of differential and algebraic equations. We do not try to eliminate the algebraic part of the models but treat them as a unit. Thus, our models are very general and robust. Of course it goes with a price. We have also to develop a very powerful solving engine. This enables e.g. to generate models of the robots in very general Lagrangian form and successfully solve them efficiently as reader can assure himself (by using e.g. Puma 560 robot project from the BondSim Library).

Another point by which our approach differs from the others is that we do not use any modelling language such as Modelica or some other. Instead, by using the object oriented modelling approach, we code the model as a tree of the modeling components. The models are created by drag and drop technique. Only, at the level of elementary components we use the mathematic expressions, but there are basically simple ones. All the other is left to the program. This way, the modeling of pretty complex system can be done and the problem solved without going into much mathematical details. This opens possibility to play with the models to rather wide user population. Using BondSim we may explore the physics of the problem and learn modeling. All the modeling problems that we as developers encountered so far we solved successfully by BondSim. The program library contains a large collection of such projects from different fields of engineering. We invite the users if they find a real engineering problem (not a pathological one) that they failed to solve by BondSim, to report to us at the address given in the manual. We will appreciate this very much.

You are really welcome to the BondSim a powerful modeling and simulation framework. Using

BondSim you can solve different problems in engineering ranging from simple mechanical, electrical, thermal system to complex solid-state electronic circuits, multibody systems and continuous system models. It offers in particular a convenient environment for dealing with mechatronics systems. It uses a unified approach based on hierarchical Bond Graphs and continuous and discrete time block diagram models.

The BondSim framework supports systematic top-down model decomposition and/or model building using library components. This way the complex models for solving different real life problems can be developed efficiently. Every effort was taken to make you feel comfortable even if you use it for the first time.

The models are developed using simple drag and drop techniques. There is no a specific simulation language that user must learn in order to develop a model. The physics of the problem can drive the user. Thus, the physical components in the problem may be represented by the corresponding model components. They are interconnected in similar way as the physical components are. These components are "opened" next and modeling continued. In this way the physical models are developed that has a similar structure as the corresponding physical system.

-o-

BondSim Features

BondSim is an integrated modeling and simulation environment that lets you develop a model for a given problem by system decomposition. At every stage of decomposition the model is described by components interconnected by bond lines. Equations representing the model are generated automatically. Numerical and computational algebra methods then are used to simulate the model behavior.

Characteristics

- Visual modeling;
- Top-down and bottom-up hierarchical model development,
- Bond Graph elementary components that support power and control interactions;
- The electrical and mechanical components can be represented using common electrical or mechanical symbols;
- Block diagram input-output components;
- A simple language for definition of constitutive relations at the elementary components ports. Most algebraic, relational and logical operations are supported including if-else-then relations;
- Hierarchical definition of model parameters;
- A tool for analyzing ports connections over model hierarchy;
- B-spline interpolation of tabular data;
- Symbolically user defined functions;
- Support for project and component repositories. The model repositories already contain many projects and components from mechatronics applications;
- Windows like complex copy insert and delete operations on the projects and the components;
- Drag operations on single and group of components;
- Automatic building of the mathematical models equations in symbolic form;
- Symbolic generation of Jacobian matrices used for model solving;
- Decompiling of model equations to a human readable form for inspection;
- Just-in time compiled approach to solving model equations during simulation using a state of art differential-algebraic equation solver;
- Generation of *x*-*t* and *x*-*y* output plots during the simulation;
- Frequency spectra and response analysis based on FFT;
- Printing screen images to file (bond graphs and data plots) in emf (enhanced metafile format), e.g. for documentation purposes;
- Collaboration support by exporting and importing component models and complete projects;
- Discretization of continuous signals into their discrete equivalents and operation on discrete components;
- Online Help;

Benefits

- Representing models of engineering systems or devices as trees of component models. These ways modeling of really complex systems can be easily developed. Both the top-down and bottom-up

approaches are supported;

- Bond Graphs offer high capabilities for modeling complex multi-domain mechatronics problems in a unified way;
- Modeling is independent of model solving during the simulation. Causality relations are not used; the underlying mathematical models are implicit algebraic and differential equations;
- Use of combination of symbolic and numerical model solving enables high level of flexibility in model development and use;
- Robust differential-algebraic equation solver (based on BDF methods and analytic Jacobian matrices);
- The program is self-contained – does not need a separate compiler;
- Projects and components stored in the program libraries can help to master object-oriented approach to problem solving;
- Collaboration support enables organizing of complex modeling projects as interacting component subprojects and exchange of developed models by exporting and importing.

-o-

Installing and Uninstalling BondSim

Installing BondSim

- **Before you install BondSim, make sure your computers meet the following minimum system requirements**:

  - Processor — 1 gigahertz or faster
  - Memory — 1 gigabyte(GB) for 32 bit or 2 GB for 64 bit OS
  - Disc Space — 500 MB or above
  - Operating systems — Windows 7, Windows 8.1, Windows 10
  - .NET Framework version 4.0
  - Recommended screen area setting 1024 by 768 pixels or higher.

- **To install BondSim:**

1. Search for the installation file SetupBondSim*.exe, stored on a memory stick or any other storage medium.
2. Double-click the installation file. The BondSim Setup wizard starts. Follow the instructions that appear on your screen.

The setup program first checks if a more recent program was already installed on your system. If it is not, the setup program will install the program. BondSim also needs .NET Framework 4.0 or higher on your machine. Note that you will need to have the administrator privileges to install the program (or otherwise ask the system administrator to do it for you). However, for running the BondSim you do not need the elevated privileges.

-o-

Uninstalling BondSim

To uninstall BondSim:

- **Windows 8.1 and 10**

1. Right click the **Start button** and then select the **Programs and Features**.
2. Select the program.
3. Click the **Uninstall.**

- **Windows 7**

1. Click the **Start button**.
2. Select **Control Panel** and then **Uninstall a program**.
3. Select the program.
4. Click **Uninstall.**

You will need to have the administrator privileges to uninstall the program (or otherwise ask the system administrator to do it for you).

-o-

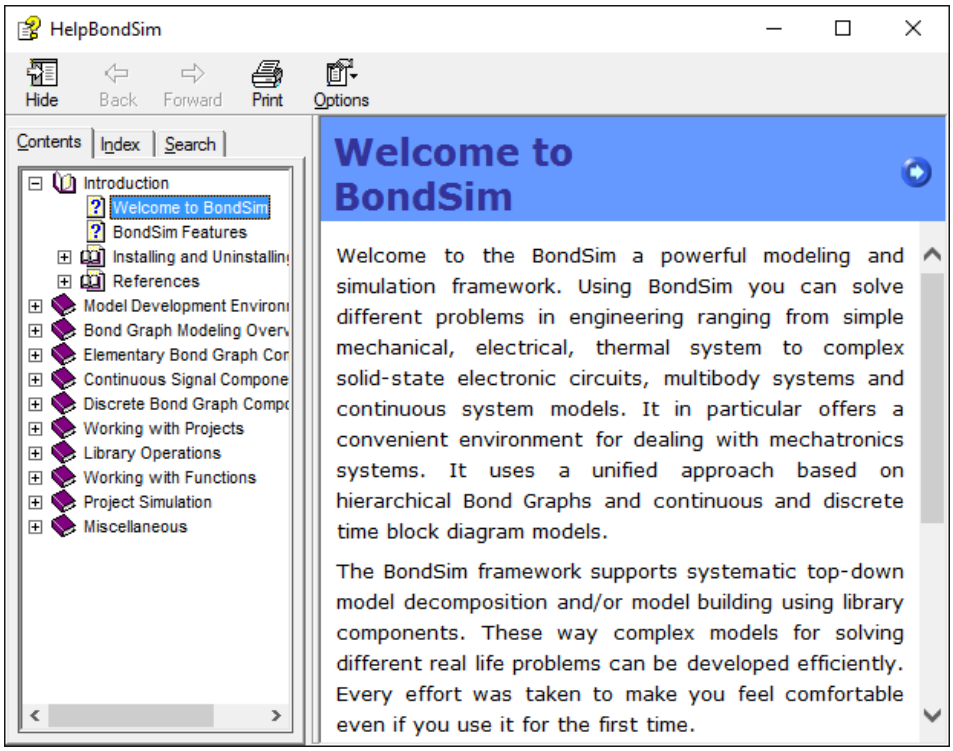
Getting Help

BondSim Help is your source of information about BondSim. Help describes how to perform a wide variety of tasks - from installing BondSim to run simulation.

- **To open Help:**

1. On the **Help** menu, click **Contents** command.
   The **HelpBondSim** dialog box appears.

BondSim Help



2. Each tab helps you to locate information in a different way. Use the *Contents* tab to locate the topics, the *Index* tab to look up for the keywords and the *Search* tab to search for a text.

-o-

Contact

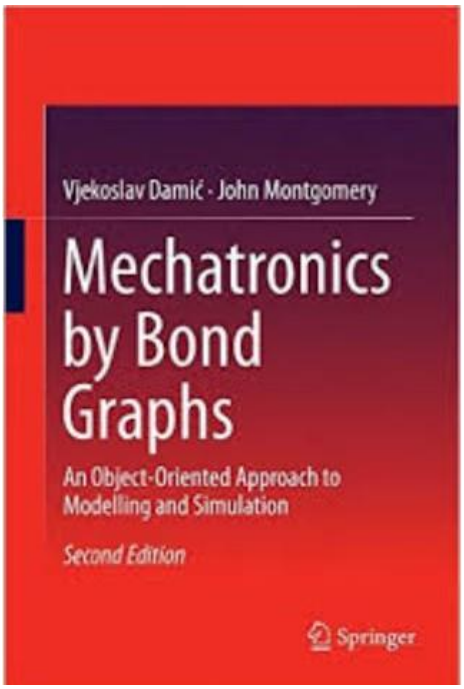
If you have questions concerning BondSim please contact at the address given below. Any feedback, advice or support for further work, as well as suggestion on other mechatronic or other relevant problems is truly welcome.

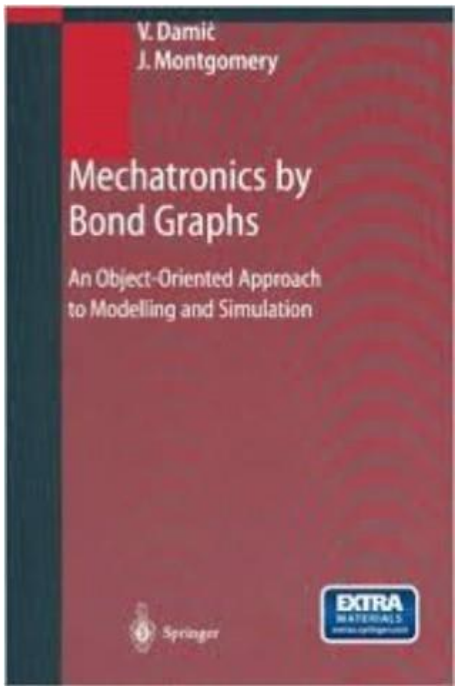
Prof. dr. sc. Vjekoslav Damic and the BondSim team
Engineering Simulations, ltd., Dubrovnik, Croatia
University of Dubrovnik, Dubrovnik, Croatia
Contact: info@bondsimulation.com
vdamic@unidu.hr
http://www.bondsimulation.com

-o-

References

Books





- Damic, V. and J. Montgomery, *Mechatronics by Bond Graphs*, 2nd ed. Springer-Verlag, 2016.
- Damic, V. and J. Montgomery, *Mechatronics by Bond Graphs*, 1st ed. Springer-Verlag, 2003.
- Damic, V. and M. Cohodar, Chapter 17, Multibody System Modeling, Simulation, and 3D Visualization, pp.627-671, in book © Springer International Publishing Switzerland 2017, W. Borutzky (ed.), *Bond Graphs for Modelling, Control and Fault Diagnosis of Engineering Systems*, DOI 10.1007/978-3-319-47434-2_17

-o-

Journal and Conference Papers

1. V. Damic, Modelling flexible body systems: a bond graph component model approach, Mathematical and Computer Modelling of Dynamical Systems (2006) 12, N0.2-3, pp. 175-187.
2. M. Cohodar, W. Borutzky, V. Damic, Comparison of different formulations of 2D beam elements based on Bond Graph technique, *Simulation Modelling Practice and Theory* 17 pp. 107–124, doi:10.1016/j.simpat.2008.02.014, 2009.
3. V. Damic, M.Cohodar, Dynamic Analysis and Visualization of Spatial Manipulators with Closed Structure, Proceedings of the 26th DAAAM International Symposium, pp.0109-0117, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-07-5, ISSN 1726-9679, Vienna, Austria DOI: 10.2507/26th.daaam.proceedings.016, 2016.
4. V.Damic, M.Cohodar, Dynamic analysis of Stewart platform by bond graphs, *Procedia Engineering,* Vol.100, pp.226-233, 2015.
5. V.Damic, M.Cohodar, M. Kulenovic, Modeling and Simulation of Hydraulic Actuated Multibody Systems by Bond Graphs, *Procedia Engineering* 69, 203-209, 2014.
6. V.Damic, M.Cohodar, Dynamic analysis and 3D visualization of multibody systems, IMAACA: The International Conference on Integrated Modeling and Analysis in Applied Control and Automation, 8th edition, part of I3M (The 12th International Multidisciplinary Modeling&Simulation Multiconference,) September 21-23, 2015. Bergeggi, Italy
7. V.Damic, M.Cohodar, D.Damic, Multibody Systems Dynamical Modeling and Visualization based on IPC technique, *The proceedings of the 2014 International Conference on Bond Graph Modeling and Simulation – ICBGM'2014*, Edited by Jose J. Granda, Dean C Karnopp, Simulation Series, Vol.46, No.8, pp. 773-778, ISBN: 978-1-63266-700-7, The Society of Modeling & Simulation, Monterey, USA, July 6-10, 2014,
8. V.Damic, M.Cohodar, Multibody systems with closed structure: Dynamic analysis and visualization of slider crank mechanism, *Journal of Trends in the Development of Machinery and Associated Technology*, Vol.19, No.1, ISSN 2303-4009, pp. 173-176. Editors: Dr. Sabahudin Ekinovic, Dr. Joan Vivancos Calvet, Dr. Senay Yalcin, 2015.
9. V.Damic, M.Cohodar, Procedure for Visualization and Dynamic Analysis of Robot Manipulator, *Journal of Trends in the Development of Machinery and Associated Technology*, Vol.18, No.1, ISSN 2303-4009 (online), pp. 127-130. Editors: Dr. Sabahudin Ekinovic, Dr. Senay Yalcin, Dr. Joan Vivancos Calvet, 2014.
10. V.Damic, M.Cohodar, D.Damic, Bond Graph Formulation of Impact with Friction in Multibody Systems, *Proceedings of the IASTED International Conference, Modelling, Identification and Control (MIC 2013),* DOI: 10.2316/P.2013.794-085, pp. 298-302, ISBN: 978-0-88986-943-1, 794MIC, Acta Press, February 11-15, 2013, Innsbruck, Austria, 2013.
11. V.Damic, M.Cohodar, A.Omerspahic, Dynamic analysis of an omni-directional mobile robot, pp. 153-156, *Journal of Trends in the Development of Machinery and Associated Technology*, Vol.17, No.1, ISSN 2303-4009, Editors: S. Ekinovic, J.Vivancos, S.Yalcin, 2013.
12. V.Damic, M.Cohodar, D.Damic, Discontinuities in Physical Modeling: Bond Graph Models of Impact in Multibody Systems, *Proceedings of the International Simulation Multi-Conference – ICBGM*, ISBN #: 1-56555-348-9, pp.250-255, 8-11 July 2012, Genoa, Italy, 2012.

13. V.Damic, M.Cohodar, M. Kulenovic, Modeling and Simulation of Hydraulic Systems by Bond Graphs, *Annals of DAAAM for 2012 & Proceedings of the 23rd International DAAAM Symposium*, Volume 23, No.1, pp. 0591-0594, ISSN 2304-1382, ISBN 978-3-901509-91-9, Ed. B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2012.
14. V.Damic, M.Cohodar, M. Kulenovic, Physical Modeling and Simulation of Mechatronics Systems by Acausal Bond Graphs, pp. 247-248, *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium*, Volume 22, No. 1, ISBN 978-3-901509-83-4, ISSN 1726-9679, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria 2011.
15. V.Damic, M.Cohodar, Study of Thermal Effects in 2D Beams Using Co-rotational Framework in Bond Graphs Settings, *Proceedings of* the 2010, pp.158-163, *International Conference on Bond Graph Modeling, Society for Modeling and Simulation International, ICBGM* 2010, Edited by Jose Granda & Francois Cellier, Orlando, Florida, U.S.A, 2010.
16. V.Damic, M.Cohodar, M. Kulenovic, An object oriented approach to modelling of flexible multibody system: Focus on joint constrains, pp.323-324, *Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium*, Vol.21, No.1, ISBN 978-3-901509-73-5, ISSN 1726-9679, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria 2010.
17. V. Damic, M. Cohodar, Modeling thermal effects in high speed rotating elastic manipulator by bond graphs, *6th International Congress of Croatian Society of Mechanics*, Edited by Ivica Smojver&Jurica Soric, ISBN 978-953-7539-10-8, Dubrovnik, Croatia, 2009.
18. M. Cohodar, W. Borutzky, V. Damic, The application of a co-rotational approach in bond graph settings to the modeling of general spatial mechanisms undergoing large motions, *Proceedings of the 2007, International Conference on Bond Graph Modeling, Society for Modeling and Simulation International, ICBGM 2007*, ISBN:1-56555-310-1,148-155, Edited by Jose Granda & Francois Cellier, San Diego, California 2007.
19. V. Damic, M. Cohodar, Bond Graph Based Modelling and Simulation of Flexible Robotic Manipulators, *20th European Conference on Modelling And Simulation ECMS*, Bonn 2006.
20. V. Damic, M. Cohodar, Dynamics of Flexible Multibody Systems using Co-rotational Approach, *5th International Congress of Croatian Society of Mechanics*, Edited by F.Matejiček, ISBN 953-96243-8-X, Trogir, Croatia, 2006.
21. V. Damic, M. Cohodar, A Bond Graph approach to Modelling of Spatial Flexible Multibody System Based on Co-rotational Formulation, *International Conference on Bond Graph Modeling and Simulation (ICBGM'05)*, January 23 - 27, 2005, New Orleans, USA, Proceedings of the 2005 International Conference on Bond Graph Modeling, pp. 213-218, 2005.
22. V. Damic, M. Cohodar, Modelling flexible multibody systems using bond graph technique, *4th International Congress of Croatian Society of Mechanics*, Bizovac, Croatia, 2003.

-o-

Model Development Environment

Starting BondSim

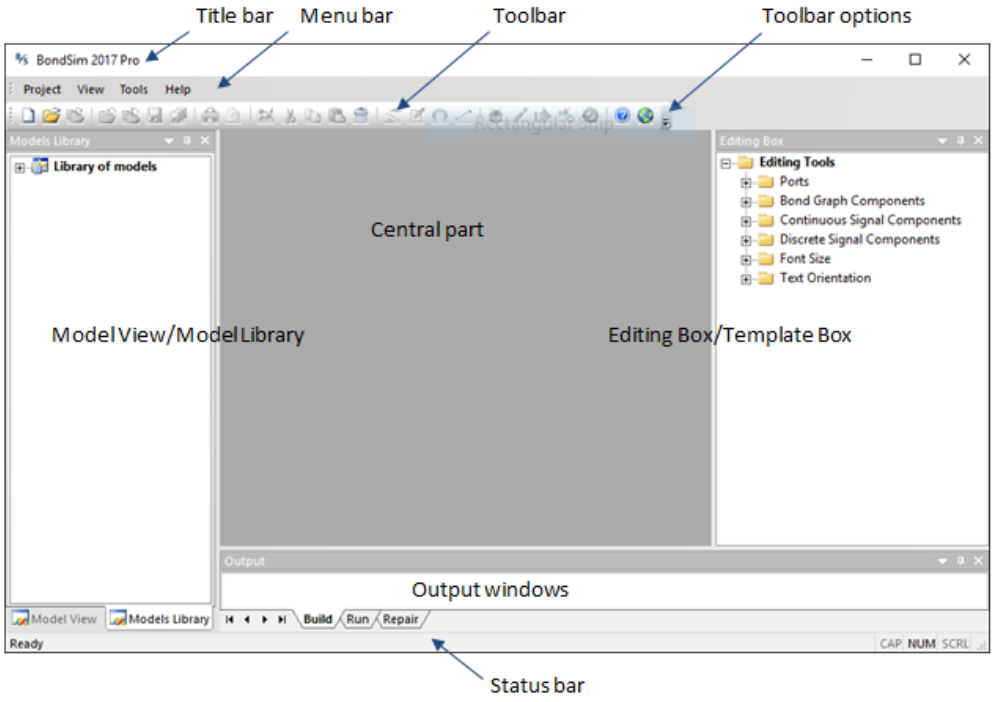
To start BondSim:

- Double click the BondSim* shortcut icon  on your Windows desktop, or
- Click **Start** button,
    1. Select All apps (All programs in Windows 7)
    2. Select BondSim startup folder
    3. Select BondSim* application

-o-

BondSim Main Frame

The first windows that opens when BondSim starts is the *Main Frame window* (see below). It contains the other windows. There are familiar *Title bar*, *Menu bar* and Tool bar at the top, and *Status bar* at the bottom of the main frame. The others such *as Model View* and *Models Library* on the left, *Editing Box* and *Template Box* on the right, and Output with Build, Run and Repair at the bottom are designed to simplify using BondSim. In the central part of the Main Frame Windows there is a free space that is used to open document windows in which the models for the current modeling and simulation project are developed.



Main Frame

- Title bar — The area displaying the title of the open project, component, etc. Initially it contains BondSim program name
- Menu bar — The area containing the pull-down menu options. There are two kinds of the menu. If there is no opened a project there is *Default* menu, which contains the basic *Project*, *View*, *Tools* and *Help* commands.

  When a modeling project is opened the *BondSim* menu appears that contains the commands for operations on the project.
- Toolbar — The area containing buttons that are shortcuts of often used menu commands.
  - Options — It contains Add or Remove Button command to select standard or customize toolbar, menu and other commands.
- Central Part — The central frame area where the project or component document windows are opened.

- Model View — A window that displays the hierarchical structure of the currently opened project in the form of an Explorer type tree.
- Models Library — A tabbed document window, which contains the list of the complete projects and component models stored in the library segment of the workspace.
- Editing Box — A window which contains the groups of the tools for editing the models. Generally the tools are dragged from the Editing box and dropped into the document window in the central part.
- Template Box — A window that contains predefined mechanical and electrical symbols. These symbols are used to construct the component represented by the common engineering schemas instead of textual titles (word models). It also supports the drag and drop technique.
- Output — The bottom part of the frame window contains three tabbed windows: *Build*, *Run* and *Repair*. These windows shows the messages generated during the corresponding operations.
- Status bar — The area on the bottom of main application windows that shows information about the current operation.

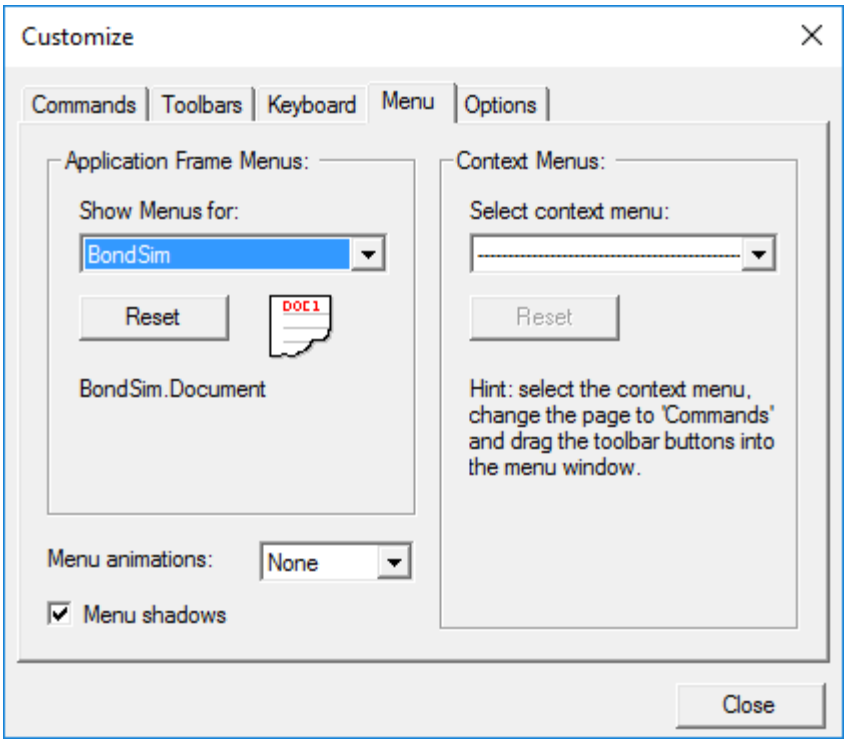
Note: Some commands described in this documents are not accessible in the *basic* version but only in the *professional* version of BondSim.

-o-

How to Reset Default Menu

- If default *menu* didn't appear when we started the BondSim program to reset it we can:

1. Click **Toolbars Option** button,
2. Select **Customize** command from **Add or Remove Buttons** dropped down menu. The dialog shown below appears.
3. Select **Menu** tab.
4. Select **Default Menu** in menu bar list.
5. Click **Reset** button, and then **Close** the dialog window



Toolbar customize dialog

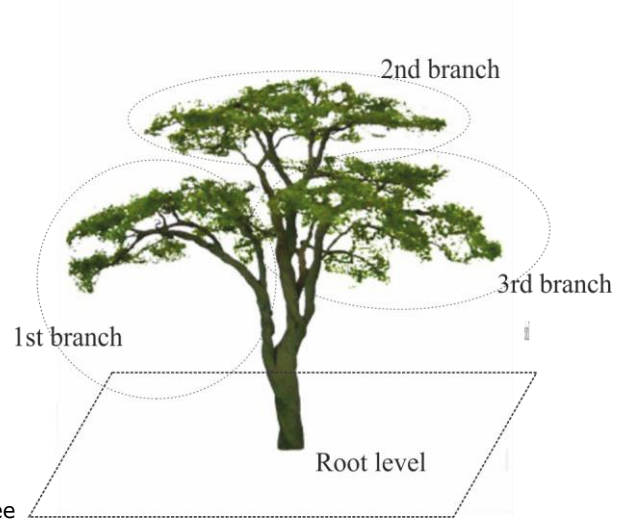
-o-

Multilevel Component Modeling Approach

Practically any engineering system has a complex structure consisting of other components, these ones of the still others usually simpler components, etc. Thus e.g., an electric motor is composed of a body, which houses a rotor fixed onto a shaft that rotates in the bearings mounted in the body, the connection box, the feet for fixing the body to ground, etc.

In order to build the models that describe as closely as possible the behavior of the real systems they should have a similar structure as well. Thus, they should consist of the component models that represent the basic parts (components) and which are connected in the similar way. These ones on other side are built also of the other component models, etc. The model of every component, thus, has the form of a tree (see the figure below).

The root of the tree corresponds to the level of the basic component. At that level we see the component as seen from the outside. It contains the *ports*, which serve for its connection to the other components. To observe its structure, however, we need to *open* it. It may contain the branches, which represent the sub-component models, e.g. the models of the body, rotor, shaft, etc. These constitute a higher level the component models.



Component model tree

We can proceed further by opening every component branch. They may also consist of the smaller branches - the simpler components. Finally, we reach to the levels of leaves, which cannot be open further. These are the simplest components, the *elementary* components, which describe the physics of the problem.

To build the model of a system we generally use *Bond Graphs*. They represent a unified approach to modeling of engineering systems covering different physical domains. This is typical of mechatronics, where we have closely interacting mechanics, electricity and electronics, thermodynamics, control and other domains. A model can be developed in two fundamental

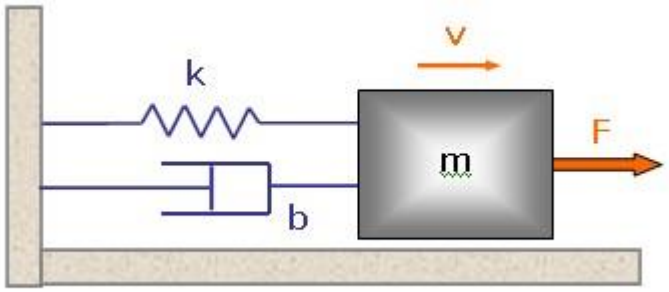
different ways - as a flat, single-level model, or as a complex multi-level model structure. By applying component model approach we normally construct a complex multi-level model structure. During the model development we may eliminate all branches and reduce it to a simpler model consisting of the leaves only. However, with exception of very simple problems, such an approach is not convenient one and is prone to errors. Therefore, the models in BondSim are generated as multi-level structures, where numbers of the levels are not limited. BondSim knows how to traverse the model tree and generate a corresponding mathematical model. This model is solved using suitable routines during simulation and generates corresponding plots. The procedure is in general automatic with minimal intervention of the user.

The storing of the models in the storage medium (e.g. disk) follows also the same philosophy. It is based on the concept of the structured storage files. Thus, when the project is closed it is removed from the memory and stored in practically same form in the storage medium. In addition to the model some other data are stored as well such as the generated plots, simulation statistics, and other data.

-o-

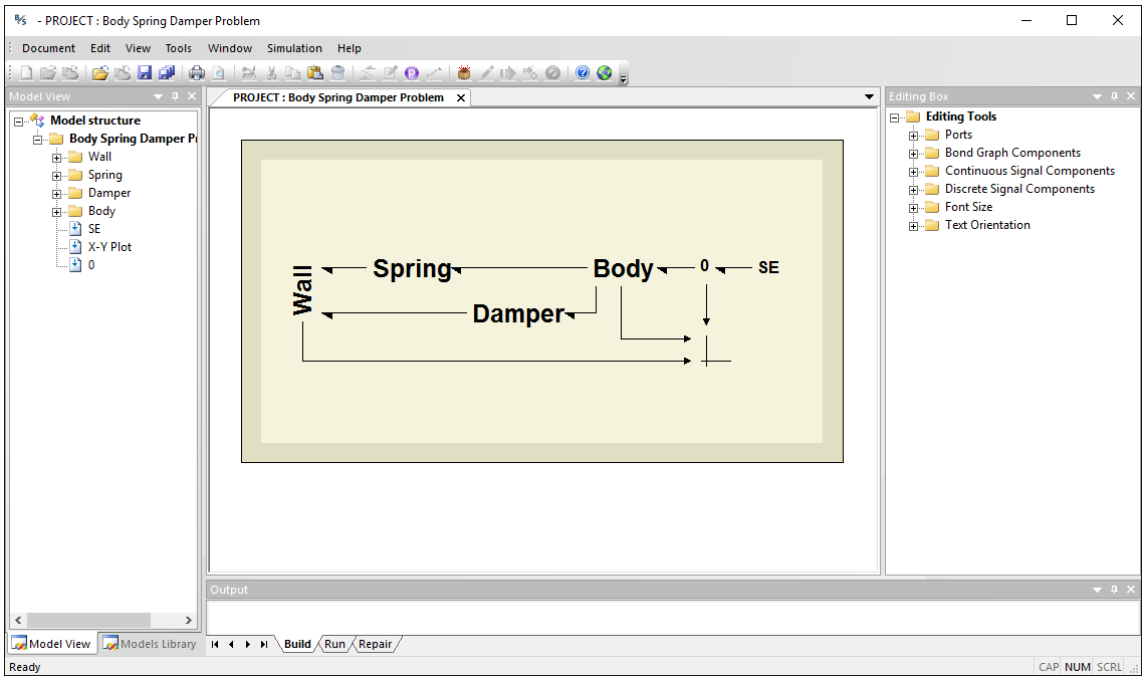
Example of Multilevel Model Structure

BondSim supports developing the project models as interacting components structured by several levels of the hierarchy, similarly as the real systems are structured. As an example consider a simple system consisting of a body of mass *m*, which can move over a flat floor by action of a force *F*. It is connected by a spring of stiffness *k*, and mechanical damper with the linear friction coefficient *b*, which are fixed to the vertical wall.



Body Spring Damper Problem

The model of the system can be found in the library under the project titled **Body Spring Damper Problem**. We can open the library project by expanding *Library of Model*, and then *Project* branches. A list of projects held in the library opens. We select the project and right click it and from dropped down menu select *Open* command. The project root document opens as shown below. Also the menu changed showing the available commands.



Library project

Note that the title bar now reads **PROJECT: View Project: Body Spring Damper Problem** indicating that it is the read only project, i.e. it cannot be edited and serves for reviewing only. A corresponding document window having the same title opens in the central part of the main frame. The window contains the central editing area enclosed by a gray strip. This area is used for placing the components that the model at the current level contains.

The document window shows Bond Graph model of the system. If we compare it with the previous figure we can see that it has similar structure as the physical system we consider. Note, however, that the corresponding components are denoted by the titles: **Body**, **Spring**, **Damper**, **Wall**. They also contain the ports denoted by half-arrows, which serves for their connection. Such forms of the models are in Bond Graphs known as *word models*. In BondSim they are used to define the basic structure of the project model, i.e. the components it consist of and how they are interconnected. They are simply called the *component models*. This is the root level of the model. The components contain the documents that describe their internal structure in form of Bond Graphs. They, thus, constitute higher levels of the model hierarchy.

The hierarchical structure of the model is shown in **Model View** windows on the left. It contains the **Model structure** tree. It is the currently expanded, which is indicated by – box. We can collapse it by clicking, and the box changes to +. The root component in the tree is the project. It contains the components shown in the central window. The first four of these are denoted by the *folder* icons. These indicate that they are the complex components, which contains the simpler ones. They are the branches of the model tree.

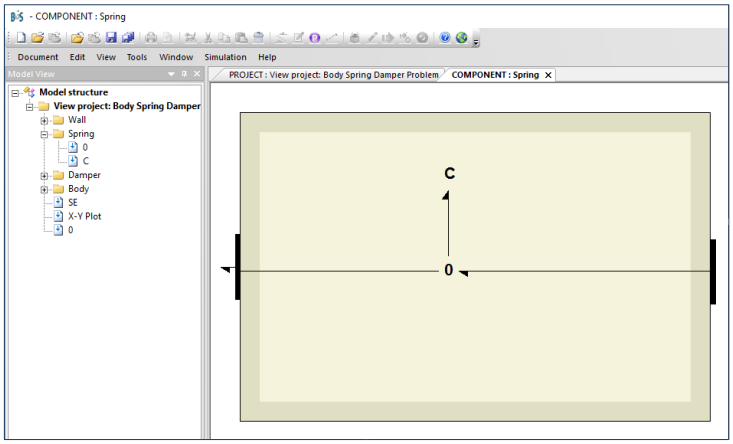
The components in BondSim are the containers (storages) of the documents that define theirs models. The + symbols indicate that the components are closed. Note also, that the last three components are denoted by the *leaf* icons. These mean that they are simple, elementary, components. In the current case these are **SE** (Source Effort) describing the external force acting on the body, a signals branch **0**, and an **X-Y plot**. These are the leaves of the model tree.

We can open a component by expanding the corresponding branch in the model three, or by double clicking its title (name) in the document window. We can also click the component title to select it. A red rectangle appears around the component indicating visually that it is selected. We may then click on **Document** menu, and select **Open Next** command. By opening a component a separate document window appears in the central part of the window as shown below.

Thus, if we open e.g. **Spring** component, its model is shown in the document window titled **COMPONENT: Spring**. Note also that Program window title changes accordingly. The component document contains now two other components **C** and **0.** Note also that model tree changes by expanding Spring branch. It now contains two simple components, which belong to the elementary Bond Graph components **C** and **0**. In general, the document may also contain the

other components, and thus a multilevel model structure can be easily built.

Observe please that the component document has at its left and right sides two rectangular strips. These are the document ports, which serve for internal connection of the components, in this case to the **0 junction**, by bond lines. Comparing with **Spring** component on the previous figure, we see that they correspond to ports, which serves for its external connections. Thus a pair of the (external) component and (internal) document ports enables communication of the external and internal components. The document ports are in form of a strip to enable connection of several components to the same port. Only restriction is that they are of the same type.



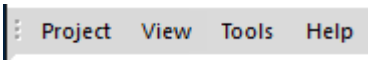
Model of the spring

To close a component we can click the control **x** button in the document tab, or open the **Document** menu and select **Close Component**, or collapse the branch in the model tree. In the similar way we can close the project, this time using **Close Project** command, control **x** button or collapse the project branch.

-o-

Default Menu

The default menu bar appears when BondSim is launched. It contains groups of commands, which are used to invoke different operations at the program level.



There are four commands:

- Project
- View
- Tools and
- Help.

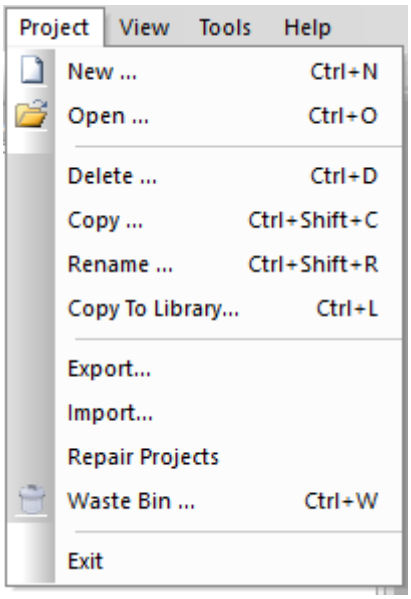
If a different menu appears reset the menu as described in How to Reset Default Menu.

To start a new project a suitable command from Project menu is chosen. A project window opens and it serves as starting point of project model development. The View menu defines the layout of BondSim Development Environment. The Tools menu offers different useful commands of which most often used are the component buffer commands copy, cut and insert, which parallel Windows clipboard commands. The Help menu enables access to the program manual and some of other commands.

-o-

Project Menu Commands

- The **Project** menu offers the following commands:

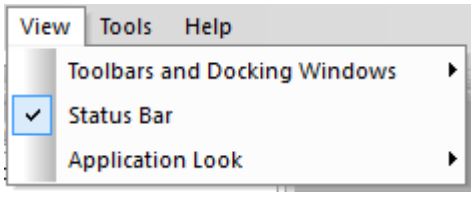


New	Creates a new project.
Open	Opens a selected existing project.
Delete	Removes a selected project to Waste Bin.
Copy	Copies a selected project and save it under a different name.
Rename	Change the name of a selected project.
Copy To Library	Inserts a copy of a selected project into the Projects library.
Export	Exports a project from the model workspace. The extension is 'ept'.
Import	Imports an *.ept project, or (*.fun, *.tbl) functions into the project workspace.
Repair Projects	Repairs the projects contained in the model workspace.
Waste Bin	Manages items placed in Waste Bin buffer. Available operations are permanent deleting or restoring of the separate items, or a group of the items, emptying of the complete buffer.
Exit	Quits BondSim.

-o-

View Menu Commands

The **View** menu defines the layout of BondSim Development Environment and offers following commands:

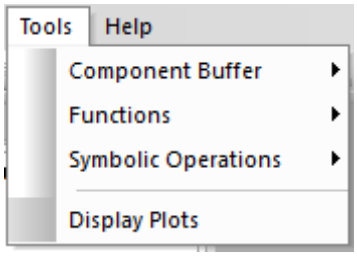


Toolbars and Docking Windows	Displays or hides the Toolbar, Models Library, Output window, Editing and Template boxes.
Status Bar	Displays or hides the Status Bar.
Application Look	Selects the kind of Model Development Environment view.

-o-

Tools Menu Commands

- The **Tools Menu Command** menu offers the following commands:

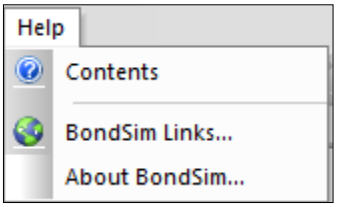


Component Buffer	Views the component placed in Component Buffer or remove the component from the buffer to Waste Bin.
Functions	Creates a new function, open an existing, remove to Waste Bin or export a function.
Symbolic Operations	Differentiates the symbolically defined function.
Display Plots	Displays the plots of external data.

-o-

Help Menu Commands

- The **Help** menu offers the following commands:

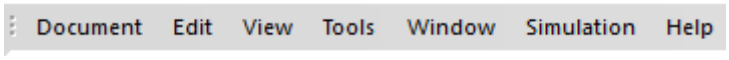


- Contents — Displays the contents for online documentation.
- BondSim links — Gets web links.
- About BondSim — Displays program information, version number, and copyright notice.

-o-

BondSim Menu

- This form of menu bar corresponds to opened projects. It consists of the following groups of commands:

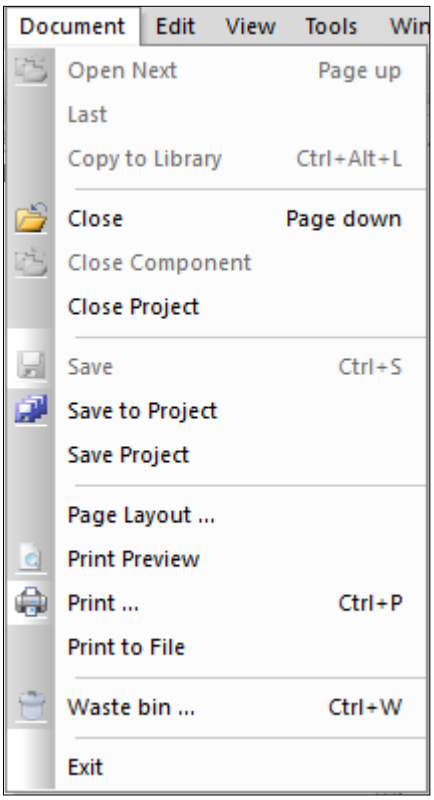


- Document
- Edit
- View
- Tools
- Window
- Simulation
- Help.

-o-

Document Menu Commands

- The **Document** menu offers the following commands:



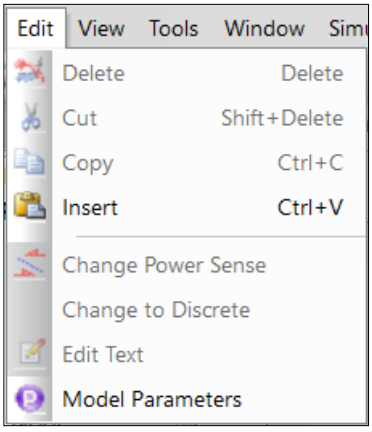
- Open Next — Opens the selected component.
- Last — Switches to the last document.
- Copy to Library — Copies a selected component into the component library.
- Close — Closes the active document and all open upper-level documents.
- Close Component — Closes the selected component.
- Close Project — Closes the current project.
- Save — Saves the current document to the corresponding component storage.
- Save to Project — Saves the current document down to the project file
- Page Layout — Defines the page orientation.
- Print Preview — Displays the current document as it would appear printed.

- Print — Prints the current document.
- Print to File — Prints the current document to a selected file in emf format (extended metafile).
- Waste bin — Manages items placed in Waste Bin buffer. Available operations are permanent deleting or restoring of the separate items, or a group of items, emptying of the complete buffer.
- Exit — Quits BondSim.

-o-

Edit Menu Commands

- The **Edit** menu offers the following commands:

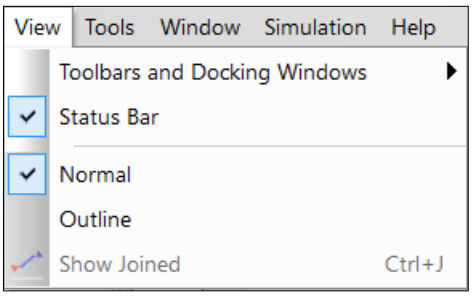


- Delete — Deletes a selected object (component, port, bond, etc.).
- Cut — Removes selected component from the document and moves it into the component buffer.
- Copy — Copies a selected component to the component buffer.
- Insert — Inserts a selected component from the component buffer into the document.
- Change Power Sense — Changes power flow direction of the selected port.
- Change to Discrete — Changes a continuous control port to a discrete one.
- Edit Text — Switches to selected component text editing mode.
- Model Parameters — Manages the parameters at the level of the current document. The parameter can be inserted, edited or removed.

-o-

View Menu Commands

- The **View** menu offers the following commands:

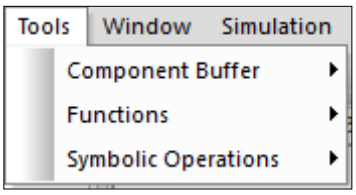


Toolbars and Docking Windows	Displays or hides the Toolbar, Models Library, Output window, Editing and Template box.
Status Bar	Shows or hides the status bar.
Normal	Switches to the normal editing mode.
Outline	Switches to outline (non-editing) view used for document overviewing.
Show Joined	Shows the joined ports.

-o-

Tools Menu Commands

- The **Tools** menu offers the following commands:

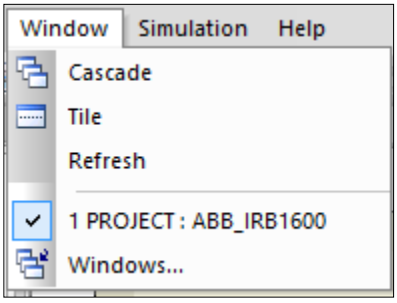


Component Buffer	Displays a component contained in the buffer, or delete a component to Waste Bin.
Functions	Creates a new or open an existing function.
Symbolic Operations	The currently only differentiation of symbolically defined function is implemented.

-o-

Window Menu Commands

- The **Window** menu offers the following commands:

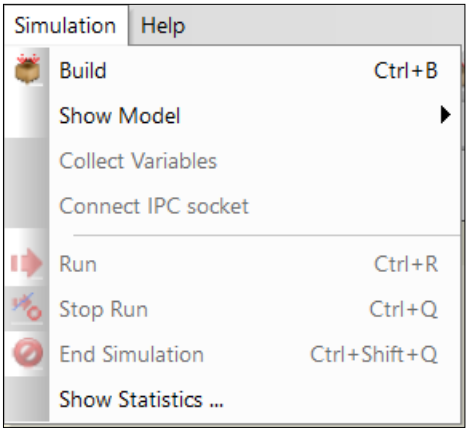


- Cascade — Arranges windows so that they overlap.
- Tile — Arranges windows as non-overlapped tiles.
- Refresh — Refreshes the active window.
- Windows… — Selects a window to activate, save, close, …

-o-

Simulation Menu Commands

- The **Simulation** menu offers the following commands:

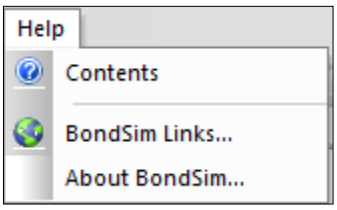


Build Model	Builds the simulation model.
Show Model	Shows the model size, equations of the model, partial derivative matrices.
Collect Variables	Collects the indexes of the continuous variables whose values will be collected during the simulation run, and then exported out.
Connect IPC socket	Creates and connect the IPC socket.
Run	Executes the simulation run.
Stop Run	Stops the simulation run.
End Simulation	Ends the simulation session.
Show statistics	Shows the statistics of the last simulation run.

-o-

Help Menu Commands

- This is the **Help** menu as in the default menu and offers the following commands:



- Contents — Displays the contents for online documentation.
- BondSim links — Gets web links.
- About BondSim — Displays program information, version number, and copyright notice.

-o-

Toolbar

The standard toolbar consists of buttons that are the shortcuts to many often used commands. It normally appears below the Title Bar. However, if there is no **Toolbar**, open the **View** menu, select **Toolbars and Docking Window**, and then check **Standard**. In same way you can hide the **Toolbar**.

	Creates a new modeling project.
	Opens an existing project.
	Opens the next level model document corresponding to the selected component. The same command can be used to open a dialog showing properties of elementary (inertial, capacitive, resistive, sources, transformers, gyrator) or block diagram (input, output, function, integrator, differentiator, summator) components at theirs ports.
	Closes the active document.
	Closes document corresponding to the selected component.
	Saves the active document.
	Saves down to the project file.
	Prints the active document.
	Prints preview of the active document.
	Deletes the selected object (word model, elementary bond graph or block-diagram component, port or bond line) to Waste Bin.
	Cuts the selected object (word model, elementary or block-diagram component) from the document into the component buffer.
	Copies the selected object (word model, elementary bond graph or block-diagram component) and put it into the buffer.
	Inserts a component (word model, elementary bond graph or block-diagram component) from the buffer into open document.
	Opens waste bin buffer.
	Changes power flow direction of the selected port.

	Switches to text editing mode of the selected component.
	Manage model parameters at level of the active document (insert, delete or edit).
	Shows a port joined to the selected port.
	Builds the simulation model.
	Shows equations of the model in the Output window.
	Executes a simulation run.
	Stops the simulation run.
	Ends the simulation session.
	Displays the table of contents for the online documentation.
	Gets web links.

When you move mouse over a button a tool tip windows opens containing the name of corresponding command and a short information what the command do. Some of commands are active only under the appropriate conditions.

-o-

Status Bar

- The status bar is placed at the bottom of the main window and shows information on the current status of the operations. To display/hide the **Status Bar** click on **View** menu and select **Status Bar**.

  When you move mouse pointer over toolbar buttons or navigate through menus, the short description of its action appears in the left area of the status bar.

  During the lengthy operations such as build and simulation run a progress control bar appears on the right end of the status bar and shows percentage of completion of the operation.

-o-

Output window

- The Output window is commonly located at the right bottom part of the main window. However, if there is no **Output** window, to open it open the **View** menu, select **Toolbars and Docking Window**, and then click on **Output**. It now appears at the right bottom corner of the main window and is activated. In the same way you hide the **Output** window, or alternatively by clicking **x** button in its title bar.

  The output windows serve to display different information for users during the building simulation model, the simulation runs or repair operation. For every of these operation a separate tabbed document activates. It is also used for displaying information on the simulation model generated during the build, e.g. the underlying mathematical model equations, the parameters, or elements of Jacobian matrices.

-o-

Editing Box

- **Editing Box** can be normally found at the right side of the Main Frame Window. However, if there is no the **Editing Box**, open the **View menu**, select **Toolbars and Docking Window**, and then click on **Editing Box**. It now appears at the right side of the Main Frame end is activated. In the same way you can hide it, or alternatively by clicking **x** button in its title bar. As with any docked window you can move it to another position within the main frame if you wish.

  It contains the tools for development bond graph and block diagram models. They are organized in the groups of the tools as shown below.

  - Ports — Tools for creating power-in, power-out, control-in, control-out, activation and triggering ports.
  - Bond Graph Components — Tools for creating the component model, inertial, capacitive and resistive components, source effort and source flows, switch, transformer and gyrator, 1 and 0 junctions.
  - Continuous Signal Components — Tools for creating input generator, X-Y and Numeric Displays, function, integrator, summator, differentiator, node, pseudo random number generator, IPC client side pipe.
  - Discrete Signal Components — Tools for creating discrete component model, A/D and D/A converters, input generator, function, unit delay, summator, clock, node, table lookup function, pseudo random number generator, and different array buffer tools.
  - Font Size — Tools for setting normal, small, medium and large font used when editing the component titles.
  - Text Orientation — Tools for setting the direction of running texts of the component titles: Horizontal and Vertical.

These tools are used to construct different component model objects in the active document window by drag and drop technique.

-o-

Template Box

- **Template Box** can be usually found at the right side of the Main Frame Window. However, if there is no **Template Box,** then open the **View** menu, select **Toolbars and Docking Window**, and then click on **Template Box**. It appears at the right side of the Main Frame end is activated. In the same way you can hide it, or alternatively by clicking **x** button in its title bar.

  It contains tools for creating word model components in the form of common schemas. There are two main groups of the template tools as shown below. The third group defines the orientation of the components.

  - Electrical Components — Tools for creating component models in form of a device, current or voltage sources, resistor, capacitor, inductor, voltmeter and ammeter, node, ground, switch, transmission line, diode, BJT, JFET, MOSFET and Opamp components.
  - Mechanical Components — Tools for creating body, spring, damper, dry friction, connector, and walls components.
  - Component Orientation — Tools for selection of the orientation of the component: Horizontal or Vertical.

  The components are created by dragging and dropping the tools onto the current active document window and are empty. It is the same type of the component as created by dragging and dropping **Component Model** from **Bond Graph Component** branch in Edit Box; they are only the containers and do not contain the model documents. The corresponding models documents are developed in the usual way.

-o-

Models Library

Models Library can be usually found at the left side of the Main Frame Window. However, if there is no **Models Library,** then open **View** menu and select **Toolbars and Docking Window**; click on **Models Library**. In the same way we can hide the **Models Library**, or alternatively, by clicking **x** button in its title bar.

The library contains complete modeling projects, the some of basic component models, electrical and mechanical components. The main parts of the library are shown below.

- Projects — Contains a list of the complete modeling projects from different branches of engineering. By expanding the project branch, selecting and right clicking to a project, and then choosing the appropriate command, it is possible to open the project for reviewing or copying it into the modeling environment for the further use.
- Components — Contains the list of component models, which can be opened for reviewing or dragged and dropped into the active model document in the central window.
- Electrical Components — Contains the list of the basic electrical component such as voltage and current sources, resistors, capacitors, inductors, semiconductors, etc. They can be opened for reviewing or dragged and dropped into the active model document in the central window.
- Mechanical Components — Contains the list of the basic mechanical components such as bodies, dampers, springs, dry friction, etc. They can be opened for reviewing or dragged and dropped into the active model document in the central window.

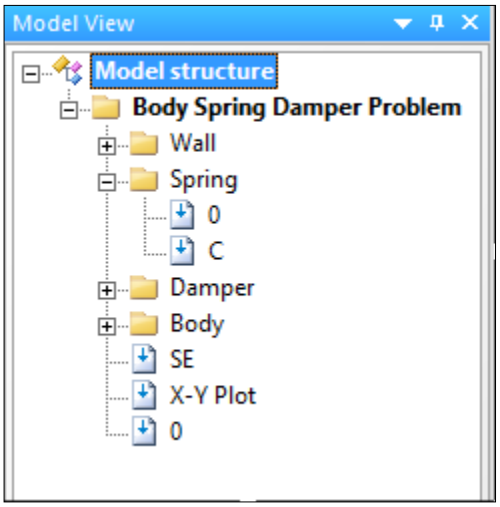
Note that component dragged and dropped onto the active document are the complete models. The system creates the component model and makes in the background a deep copy of all underlying documents and stores them into the appropriate component storages.

-o-

Model View

- **Models View** can be usually found at the left side of the Main Frame Window. However, if there is no **Model View,** open **View** menu and select **Toolbars and Docking Window**; click on **Model View**. It appears by showing the structure of the actually opened project if any. In the same way you can hide it, or alternatively by clicking **x** button in its title bar.

  The Model structure has form of the Explorer type tree, as depicted in figure below for the Body Spring Damper Problem.



- By expanding a branch the corresponding component opens and its model document opens as a separate window in the central part of the main frame.
- Similarly, by collapsing a branch the corresponding document in the central part closes, and the component closes as well. However, if it was changed the user is asked if she or he wishes to save the changes in the document to the component storage, or ultimately to the project on the disk, or to reject the changes. Note that if the documents are stored in their components and not to the project the complete data which are not in the memory is saved in a temporary storage on disk. Thus, finally when the project is saved the temporary storage is stored into a disk file. If the project is closed without storing it the temporary changes are rejected.

-o-

Bond Graph Modeling Overview

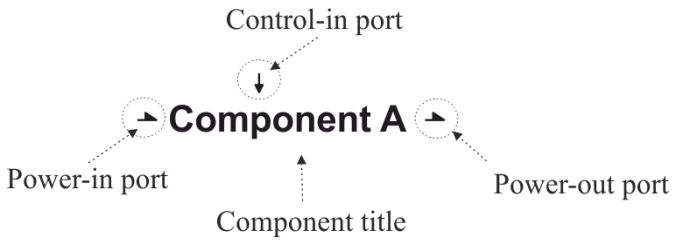
General Modeling Approach

To analyze dynamic behavior of engineering (and other as well) systems we usually start by developing the corresponding mathematical model. This model is subsequently used to find answers to question concerning the problems we are interested in. Instead of models in form of mathematical equations it is possible, however, to develop models in form of Bond Graphs.

Using Bond Graphs the mathematical model of the system can be equivalently represented in graphical form, which is easier to comprehend than the equation form. One of great advantages of Bond Graphs is that they offer a unified approach to modeling of different physical processes taking places in engineering systems. Thus e.g. typical mechatronic system consist of mechanical, electrical and solid-state components, which are under microprocessor control and in which thermal and chemical processes are also important. They can be readily modeled by Bond Graphs.

In BondSim the Bond Graph models consist of separate components that depict dynamical behavior of different parts of the system. These components are interconnected by the bond lines (or bonds for short), which describe interactions within the system. A systematic approach to modeling of a system is based on the system decomposition.

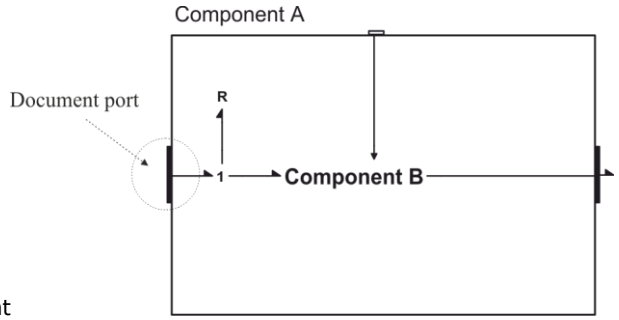
We define models in term of the general component models defined by their titles (names) and ports placed around the component periphery (see figure below). The ports are places were power flows into the component, or out of it, and are known as the *power ports*. They are represented by half arrows. There may also be the information exchange with its environment. These ports are termed the *control ports*; they are represented by common arrows and serve for control signals input or output.



Component model

Such abstract models are known as *word component models or* simply the *component models*. However, in addition to titles, it is possible to use the schemes often found in engineering, e.g. electrical or mechanical schemas.

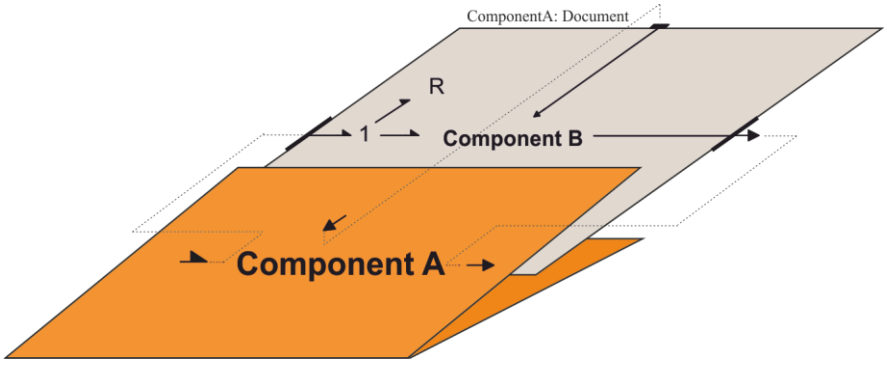
The component model does not define its complete model, only how it could be connecting to the other component models using bonds. Thus, it represents the component as seen from the outside. To define its internal model structure a separate document is used (see below).



Component model document

It has a title that corresponds to that of the component whose model it represents. There is a rectangular area within which the model is edited. On the outside of it there are thin rectangular strips, which correspond to the ports of the corresponding component model and represent the **document ports**. The model of the component may be described in terms of the other components. These components are interconnected by the bonds, but some of them are also connected to the document ports, and thus to the outside of the component as well; e.g. junction **1** in the figure above is connected to the left power-in port, and thus to the outside component connected the corresponding component port. Similarly the right port of **Component B** is connected to the right power-out port, and its control-in port to the corresponding control-in port of **Component A**.

The component model and its document jointly define the model of the component. In BondSim the components are implemented as the containers (storages) of their model documents (see figure below). The component defines its interface to the outside components and its document defines its internal model structure.



Component as a container

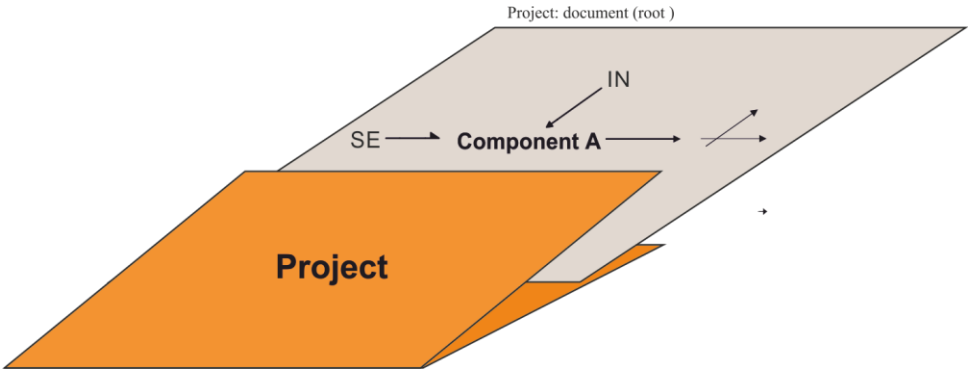
The simple components that do not need the documents are known as the *elementary* components. Their ports are used to define and store the constitutive relations of the processes seen at these ports. Bond Graph theory defines nine elementary components. Beauty of Bond Graphs is that these components are sufficient for modeling the fundamental physical processes.

These are:

- Effort **SE** and flow **SF** sources,
- Inertial **I**, capacitive **C** and resistive **R** components,
- Transformers **TF** and gyrators **GY**,
- Efforts **1** and flows **0** branches.

A complex model consisting of many general components (the model's tree branches) and elementary components (the model's tree leaves) can be, thus, systematically constructed. However, we must start from somewhere. To that purpose we introduce a component without the ports. It is used to represents the system model for the given modeling project. This root component is called the **Project** (see figure below).

The project component is created in the memory when a new project is created. The new project document, which is created and shown in the central part of the main frame, is an empty document which has the same title as the project, but has no document ports. It is used to define the structure of the corresponding system. It contains the components, which constitutes the system for the given project and its interaction with the environment. We may use Editing Box to develop the system level model for the project.



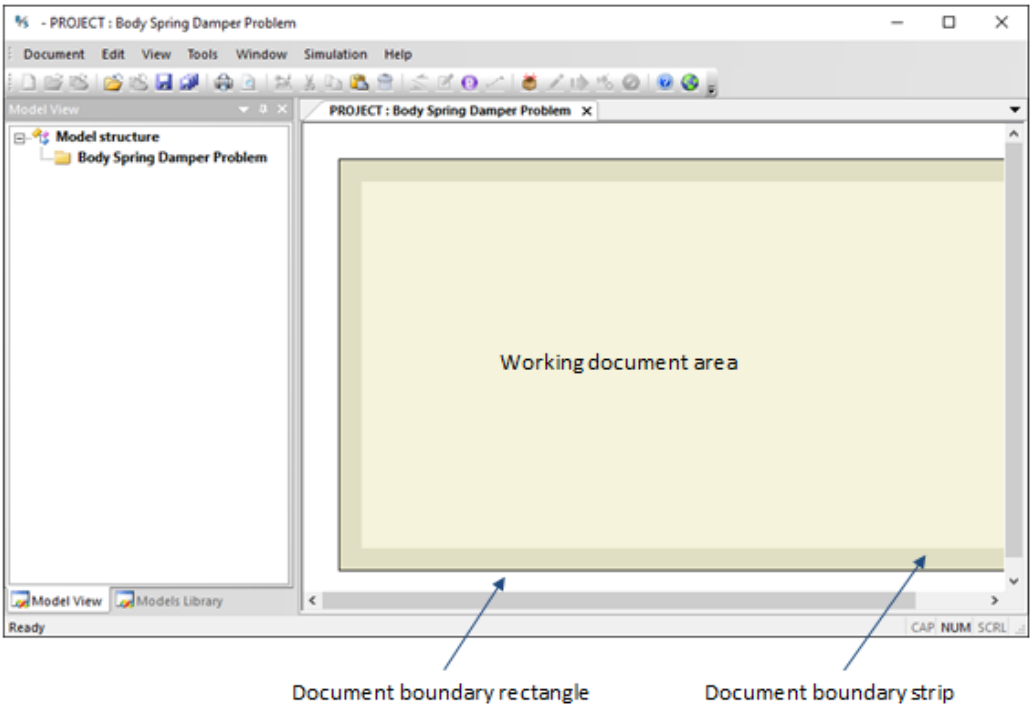
Project as component

To illustrate use of the component models to build a project model we consider again Body Spring Damper Problem. Thus to create such a project we do the following:

- Start BondSim.
- Put mouse over the first bitmap on the left in the toolbar. A tooltip appears displaying **New Project** command with information what this command do. Click on the bitmap.
- In the dialog Create New Project that opens type-in the name of the project **Body Spring Damper Problem** and click OK button.

The new project document window opens in the central part of the frame as is shown below. It is divided into the separate parts. Inside it there is a rectangle, which bounds the part of the document where bond graph model is edited. Within it there is a bounding strip (shown in a

darker color), which enclose the central part of the document that serves as the working document area. Thus, all components have to be placed inside this area, but as shown later, the bonds can be drawn across or within this bounding strip as well.



Project document

The document is empty because this is a new project. When the project is created the Model View tab is activated as well, as seen on the left side of the figure above. If we expand **Model Structure** root (click **+** box) the **Project** branch appears indicating that the root component (project) was created in the memory, but there is no **+** or **–** box ahead of the project branch because the project storage is empty.

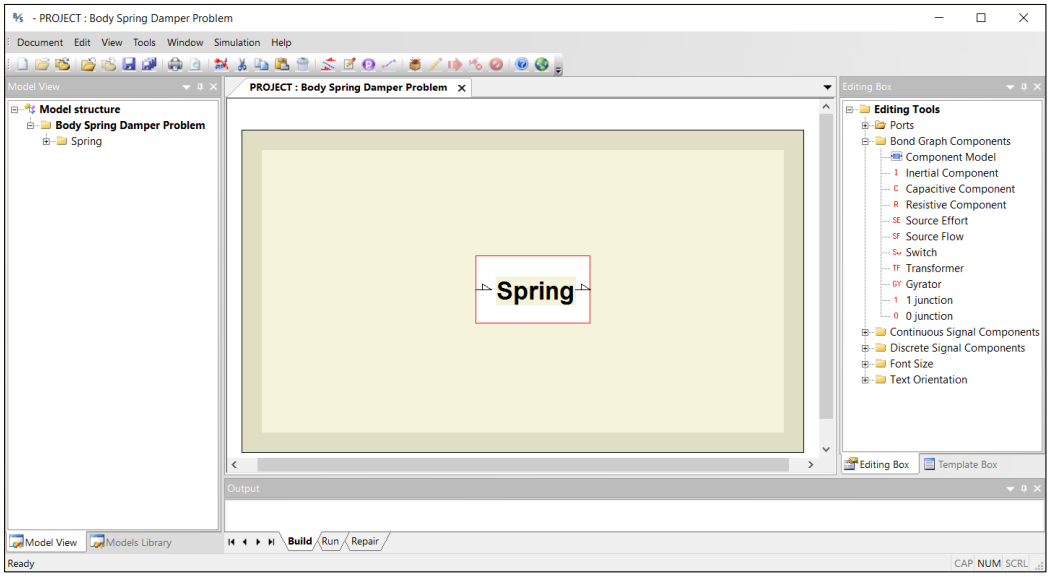
To develop system level model of the project we will create the components we already met. To that end we activate **Edit Box** tab on the right side of the frame window (if it is not already active), and expand **Bond Graph Components** branch. The list of tools for creating Bond Graph components now appears. To create a component we need to put mouse cursor over the corresponding tool, press left key, drag the tool with key pressed to the active document; as we drag it the cursor change the shape to cross. We place it within the central working area, and release the mouse. This operation is commonly called the *drag and drop* operation. It is used in BondSim to insert the components and other objects (e.g. ports) into the documents.

When we drop the tool at a place in the document window the component object is created in the memory and the vertical text cursor (the caret) appears indicating that we need to edit the component title in the current place. Hence we start typing the component title using the keyboard. We may end editing by simple clicking outside of the title text. This completes the

component creation and it appears in the document. It already has pre-created power ports. We must take into account that component boundary rectangle must be inside the working area. Therefore, we must drop the tool not too close to the working area boundary. We can move it later to some other place.

To create the **Spring** component model we drag **Component Model** tool and drop it in the center of the document as shown below. When the caret appears we type *Spring* and click outside of the text entered. This ends in-place component title editing and the component appears on the screen. The created component already contains two pre created power ports. One is the power-in port showing power transfer into the component, and the other is the power-out port showing power transfer out of the component.

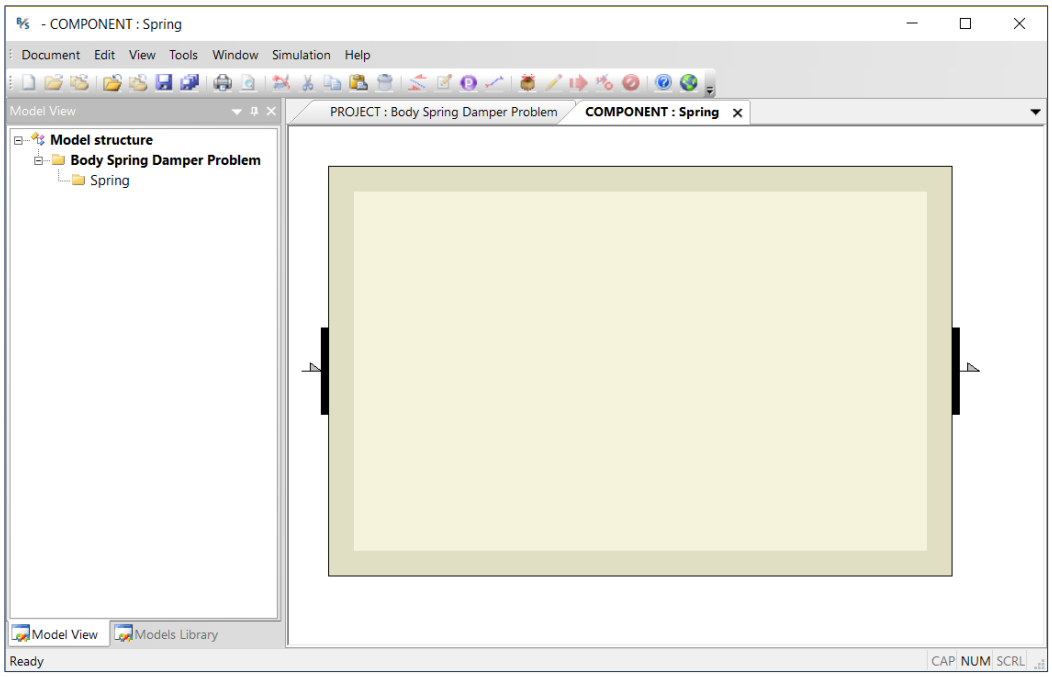
Note a red rectangle around the component. It indicates that the component is selected. On the selected components, or other objects such as ports or bonds as well, we may apply different menu commands. By clicking outside of the component it is deselected and the red rectangle disappears.



Spring component

After the **Spring** is created a corresponding branch appears under **Body Spring Damper Problem** branch. However it is has the "leaf" icon because it is an empty component model.

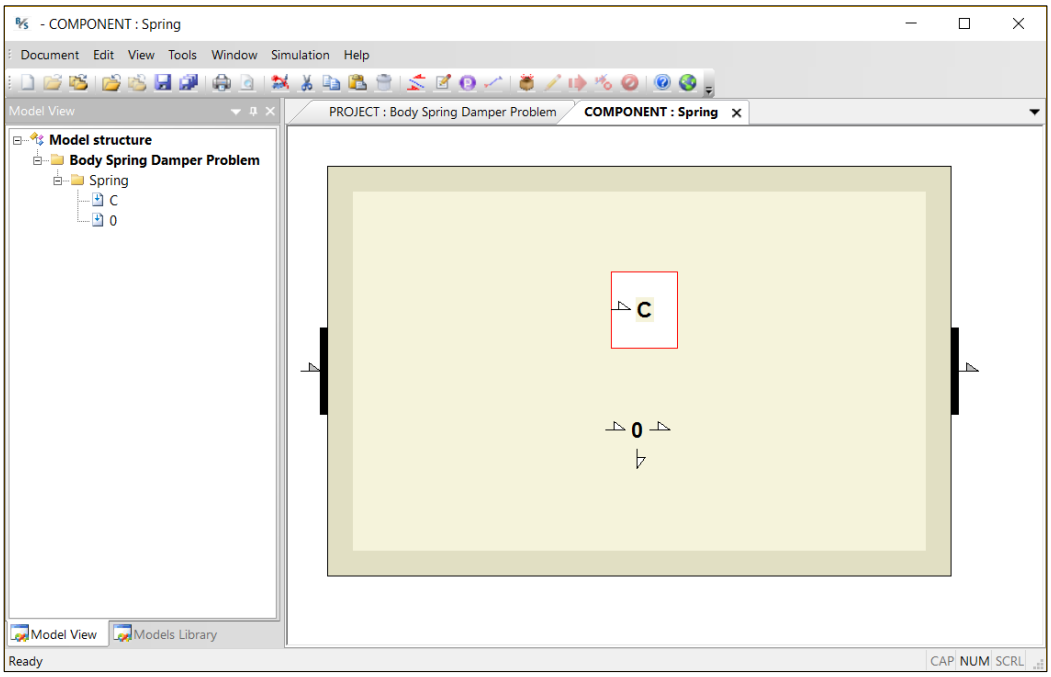
To define model of **Spring** we need to open its document. To this end we cannot use the model tree because it is not a branch yet. Because the component is already selected we may apply the **Open Next** command from **Document** menu, or **Open Component** bitmap shortcut in toolbar (third form the left), or **pgup** shortcut key. We may also simply double click the **Spring** title. A message appears asking if we wish to edit model of Spring. If we accept it a new empty document window appears titled **Component: Spring**, as shown in the next figure. The model tree didn't change because the document is empty. Note the document has two document ports, which corresponds to the Spring component ports (see figure Spring component above).



New spring document

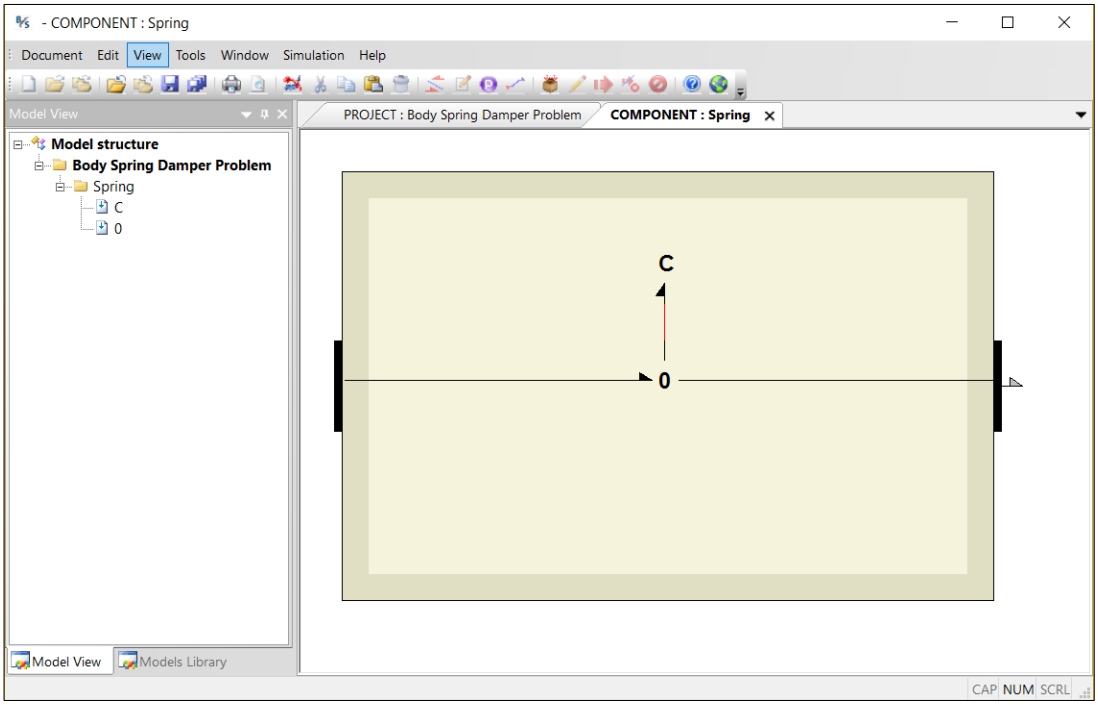
Now we may drag **0 junction** from **Edit Box** and drop it in the middle of the **Spring** document, and then drag the **Capacitive component C** and drop it above the **0 junction** (see the figure below). Thus, we have created two elementary Bond Graph components. Now the Spring icon in the model tree changes to the "folder" because the **Spring** document now contains two elementary components.

Note that **0 junction** has predefined three power ports. The power sense of the left port corresponds to that of the left document ports and thus can be connected by a bond. Hence, move the cursor over the left **0 junction** port, press mouse and then drag it towards the left document port. As we drag the mouse a bond line appears. When the mouse is over the document port it becomes selected and when we drop the port (releases the mouse) the bond is created and is in the selected state (drawn in red color). We may break the drawing of the bond any time simply by double clicking. In the similar way we may connect the right junction port and the right document port by a bond (see figure Spring model below).



Populated Spring

Note that as we drew the bonds we crossed the document bounding strip and rectangle in order to connect to the document ports, which lies just outside of the document bounding rectangle.



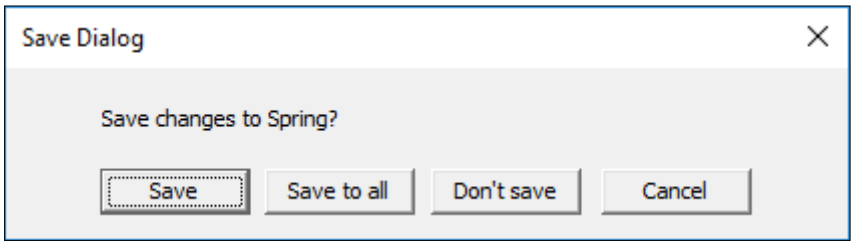
Spring model

It is now left to connect the third junction port and power port of C component. But, first we rearrange the ports. Thus, we drag the bottom junction port around the component boundary and drop it at the top. Similarly, we drag the **C** port around the component boundary and drop it at the bottom. Finally, we may drag **C** by title and move it until its port is exactly above the top **0 junction** port (we may use also the cursor keys). Now we can connect the ports by a bond.

This completes the development of the **Spring** model. The constitutive relation of the 0 junction and C component are predefined. The first one is defined by the power flow senses of the junction ports. The second one, however, is default relation which can be modified.

We can now close the document window. It is possible to do it in different ways. The simplest one is just to click the **x** button in the document title bar. We can also collapse the Spring branch in the model tree (click the **–** box). We can also select **Close** command in **Document** menu, or click **Close Document** bitmap shortcut in toolbar (the fourth from the left), or click **pgdn** key shortcut.

Because we have changed the document when closing it a dialog appears asking if we would like to save the changes:



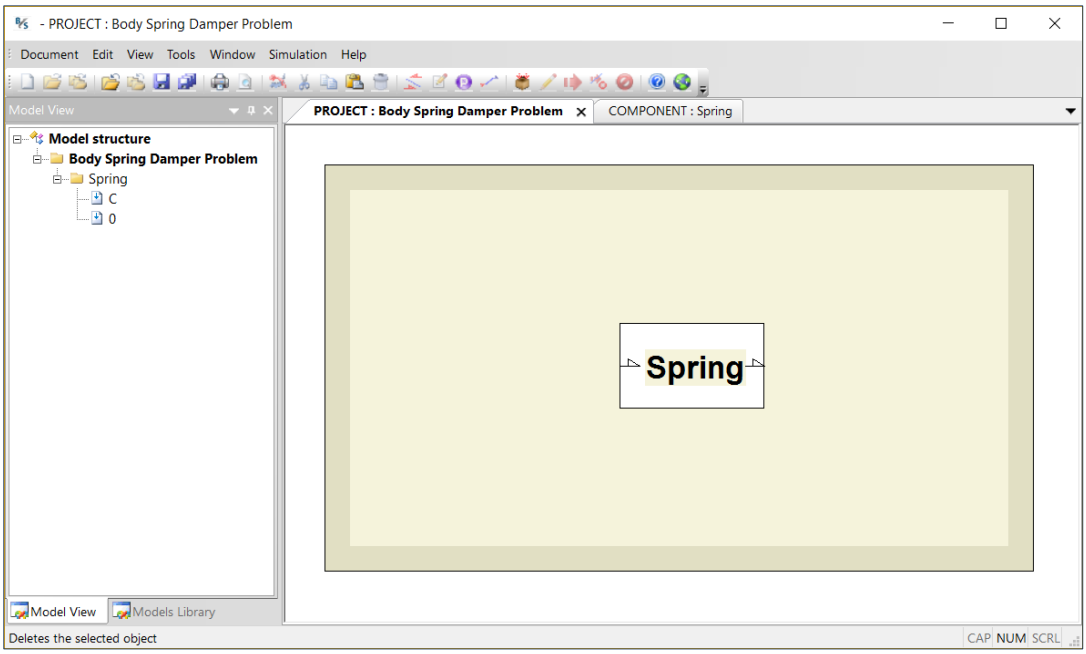
Save dialog

If we select **Save** button, the document window closes and is stored in the Spring storage. It is still not stored to the project file, and we can reject the created model later.

Selecting **Save to all** means that the document is first stored into the **Spring** storage, and then the project document is stored into the **Project** file.

Selecting **Don't save** button the document is closed but the changes made are rejected. Thus, we rejected the **Spring** model and the component is left empty. Finally selecting the **Cancel** the document closing operation is rejected and the document remains open.

We may close the Spring document by activating first to the previous document (by clicking on the Project document title bar). Thus, we obtain the picture shown below. We can now see that the **Spring** is enclosed by a rectangle, which indicates that it is opened. We can close it by selecting it (by clicking) and then using **Document**'s menu **Close Component** command or the corresponding bitmap shortcut in toolbar (the fifth from the left). The Spring document **x** button is not available now, but we can still collapse the Spring branch in the model tree to close it.



Previous document

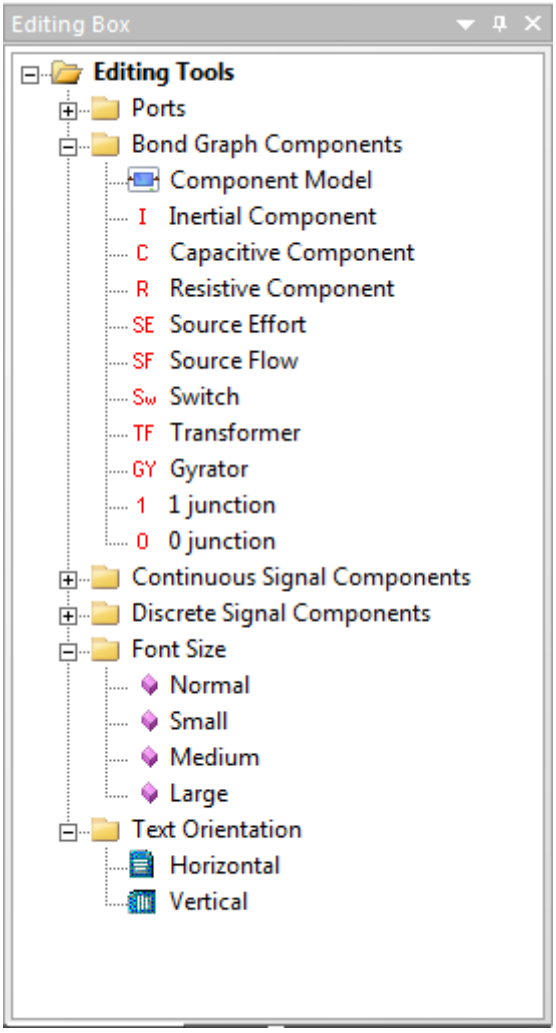
Now we have developed Spring model. In the similar ways we can develop models of other components and the complete model. The model of a project is represented in Model view as a tree, whose branches represent the components models, and leaves are the models of the elementary processes.

We can close the project by selecting **Close** command from the Document menu similarly as we did earlier. This command closes the currently active window, in this case the project document. We can also close the project by collapsing the project root branch (**Body Spring Damper Problem**). Before closing the program we need to close all opened components and save their documents one by one until the project level is reached and then finally to the project file to the disk. We can simplify these operations by selecting **Save to all** buttons when asked.

-o-

Creating, Editing and Deleting Components

Bond Graph components (but also the others) are created by dragging a suitable tool from **Edit Box** and dropping it into the active document. The figure below shows enlarged **Edit box** with expanded **Bond Graph Components**, **Font Size** and **Text Orientation** branches. The first tool under **Bond Graph Components** serves to create a component model and the others are for creation of the corresponding elementary components.



Editing Box Expanded

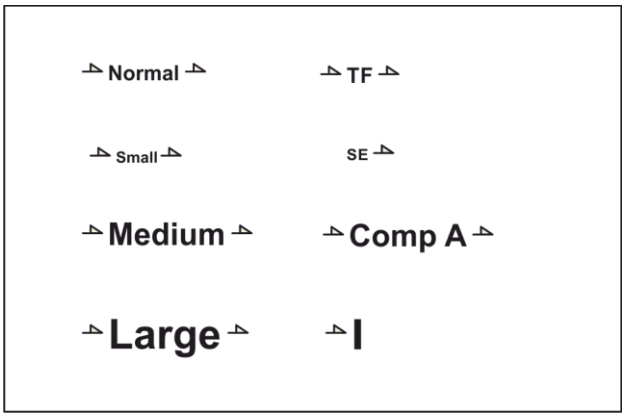
When we start dragging the tool the cursor changes its form to the "no parking" sign indicating we have not to drop it. But when we enter inside the document boundary it changes to the cross. We place the cross in the document working area, but not too close to the surrounding strip, and drop the tool. It is the best to drop it near the center of the widow. Later we may drag the created component to a suitable place. When the drop operation completes the corresponding component is created in the memory, but is still invisible. We must first define its title (name). The caret appears and we may in-place edit its title. We start typing the letters using the keyboard. All basic

editing operations are supported such as small and large letters, spaces, delete and backspace operation, use of the cursor keys, opening a new row by Enter key (if we wish a multiline title), etc. However, when we click outside of the edited text the editing operation ends, and the component appears with edited title on the screen. If we click outside of the (invisible) component, before editing its name and thus having empty title, the component appears under default **NONAME** title. We can change it later. The component has also pre-created ports. The number and type of the port depends on type of the component created.

When we drag and drop the elementary components such as *Inertial*, *Capacitive*, *Resistive* or other components, the predefined symbols appear first such as **I** for the *Inertial*, **C** for *Capacitive*, **SE** for *Source Efforts* etc., and then the caret. We still can change these symbols if we wish, but usually we accept them by clicking outside of the component. The predefined symbols correspond to the standard Bond Graph elementary component symbols.

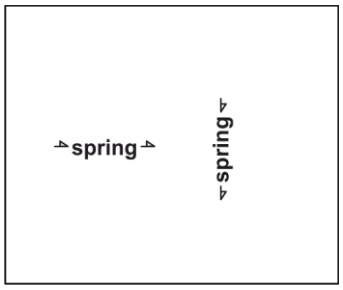
Every component created is a unique object in the memory. Even if two or more components have the same title (name), as is often the case with the elementary components, they are different objects. The component title (name) is used for the user convenience only. Internally, the program identifies every component by a unique identifier, which is based on GUID (Globally Unique Identifier).

When a component is created the program uses predefined font and the current font sizes and the orientation of the text. The font is fixed (typically Arial). The default font size is **Normal** or **Medium** depending on type of the component being created, but can be changed. The text orientation is horizontal by default, but also can be changed. The currently there are four sizes of the fonts that can be used. They are called: **Normal**, **Small**, **Medium** and **Large**. If we wish to use e.g. the small font size, we expand the **Font Size** branch in **Editing Box** (see the figure above), and select the **Small** size. The program sets the new font size for the active document and every next created component use this font size (see figure Different font sizes below).

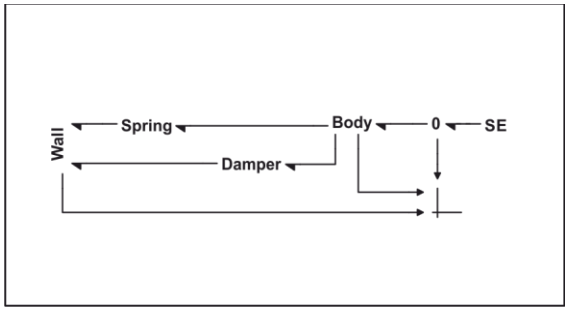


Different font sizes

Expanding the **Text Orientation** branch (see the figure above) we may choose between **Horizontal** (running from the left to right), or **Vertical** (running from below to up). When we select e.g. **Vertical** the text orientation for the active document changes to **Vertical** and every component that we create later on uses this text orientation for its title. To return to default (horizontal) orientation we need to click on **Vertical** text orientation in **Edit Box** before continuing with work. The figure Different titles running directions, below, shows a component whose titles were edited using horizontal and vertical text orientation, respectively.

Different titles running directions 

The component title can be reedited any time we wish. However, for this it is necessary that the component ports are free, i.e. not connected to other components. Thus, to change title of **Body** component in the figure below we must disconnect its bonds first. Hence, we click every of the bonds by which the component is connected to the other components to select it and then delete it (by pressing **Delete** key or selecting **Delete** command from **Edit** menu).

Spring Body Damper Problem model 

Next we select the component (by clicking on it) and either

- Select **Edit Text** command from **Edit** menu, or
- Click **Edit Text** bitmap button in the form of a page with pencil, or
- Right click on the selected component and from drop-down menu select command **Edit Text**.

The component disappears and the selected title appears with a caret within the text. Now the title can be modified as we wish, in a similar way as was done when creating the component. To finish the editing we can simply click somewhere outside the text.

To delete a component it should be disconnected first. It must be closed also. Next, we need to select it by clicking on it. Finally, we apply delete operation, which is similar to those applied to the other objects. That is, we press **delete** key or select **Delete** command from the **Edit** menu.

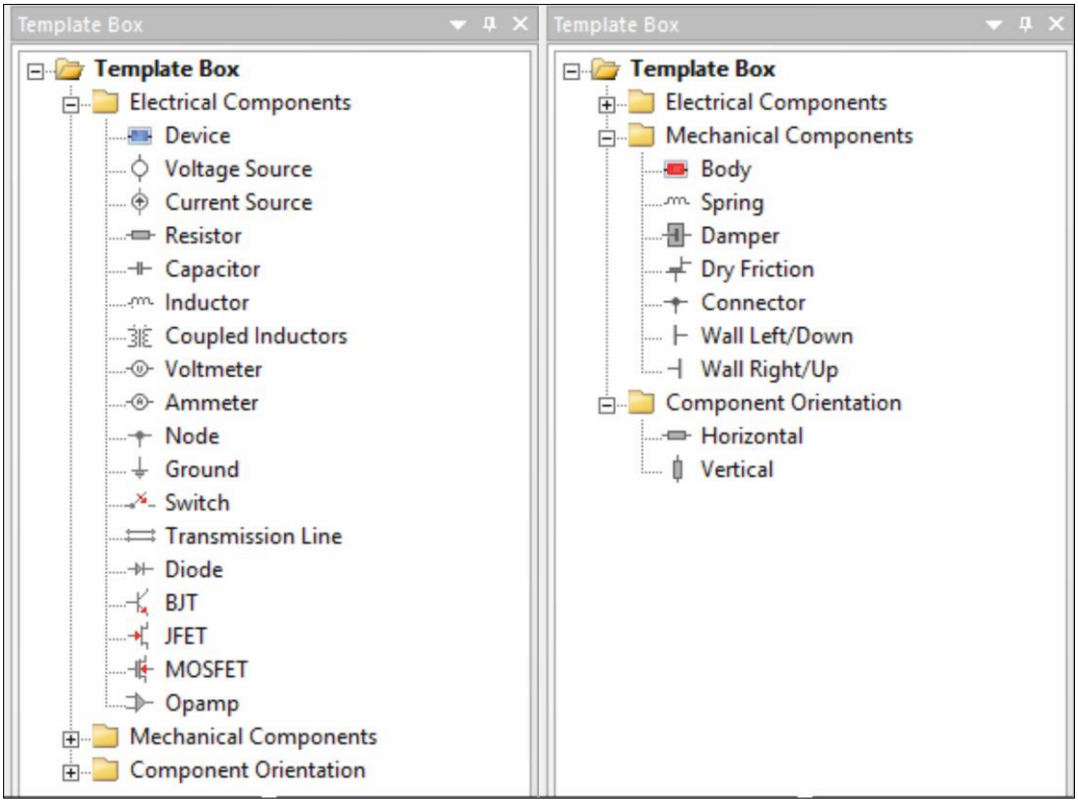
However, the component is not deleted right away but moved into **Waste Bin** buffer.

The **Waste Bin** plays similar roles in BondSim as *Recycle Bin* do in Windows. To invoke it there is command **Waste Bin** in both Project and Document menus. Using it we can delete the component completely or eventually restore it. However, the restoring does not move the component back into the document, but into a special component buffer that the program supports. This buffer has similar functions as Windows **Clipboard** (copy to, cut, insert components or group of components).

-o-

Creating Component Represented by Schemas

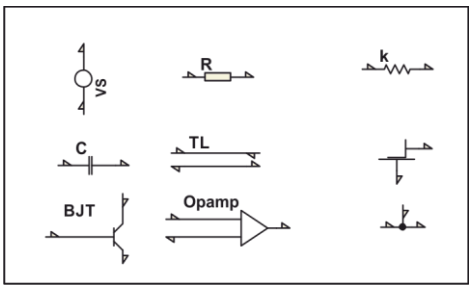
It is not necessary to create general component model as the word model component, i.e. as a component represented only by a title. In the main frame there is also tabbed window **Template Box** on the right side (usually nearby Editing Box tab). The expanded form of this box is shown in the figure below. The box contains template branches **Electrical** and **Mechanical Components**. They are shown expanded and show number of tools that enable creating of general modeling components that use common electrical or mechanical symbols.



Template Box

They are used in same way as **Edit Box** component tools. First we need to activate the box by clicking on the **Template Box** tab, and expand the corresponding branch – **Electrical Components** or **Mechanical Components**. After that we drag the corresponding tool and drop it into the working area of the active document window.

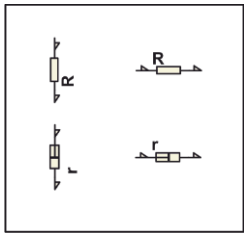
The figure below shows typical electrical components such as Voltage Source (VS), Resistor (R), Capacitor (C), Transmission line (TL), Bipolar Junction Transistor, Operational amplifier. Similarly, there are mechanical component models (on the right side) such as spring (k), Dry Friction, mechanical junction. Note that the components in addition to the schemas contain the titles too. The title is edited in usual way and can serve to specify the component specific data such as the component number, resistor value, or spring stiffness, or similar.



Electrical and mechanical component models

The components differ from the common symbols used in Electrical or Mechanical Engineering by presence of the power ports. These indicate that these are the Bond Graph component models. After their creation they are empty as any created model word components are. We can develop their model in the usual way.

In **Template Box** there is also the third branch – **Component Orientation**. As figure Template Box above shows it contains two tools, which enable creating the component having either **Horizontal** (default) or **Vertical** orientations. When a document window opens the component orientation is preset to **Horizontal**. Thus, any component created using Template Box has the horizontal orientation as the figure above shows. To change to **Vertical** we need to expand the **Component Orientation** and click to **Vertical**. The next components that are created will have the vertical orientation (see the figure below, on the left side).



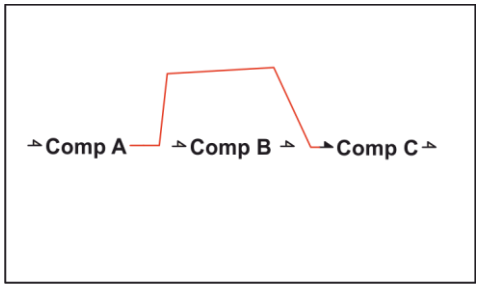
Different components orientations

However, it is advised to return to default orientation (**Horizontal**) before continuing with the model development.

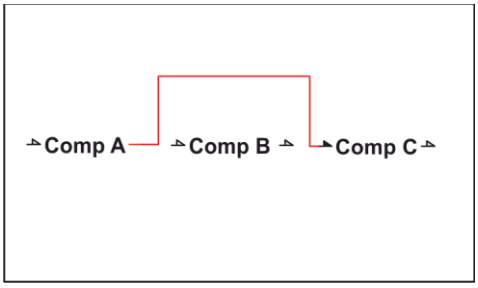
-o-

Connecting Components by Bonds

The component ports can be connected directly as shown in General Modeling Approach. But, sometimes we need the intermediate points, e.g. if we wish to go around a component (see the figure below). These points must be outside of the other components. Thus, to draw a bond we move mouse until it is above the port we wish to connect, press it and drag it until some intermediate point is reached and release the key. As we move the mouse the bond line is drawn. We continue moving the mouse to possible another intermediate point and click and the key. We may repeat drawing to other points. Finally, we move mouse until it is over a port we wish to connect to. When it changes the color to red we click the mouse key. The bond is created now and it appears as zigzag line and is selected.

Drawing bond 

We can also reshape the bond (see the figure below). Just put the mouse cursor on the bond, press the key and drag it. The bond reshapes as an elastic band fixed at its ends. Drop it when we are satisfied. You may repeat this operation several time until the bond has the suitable form.

Reshaping bond line 

We can remove a bond by selecting it (by clicking to it) and then pressing **delete** key, or use **Delete** command from **Edit** menu.

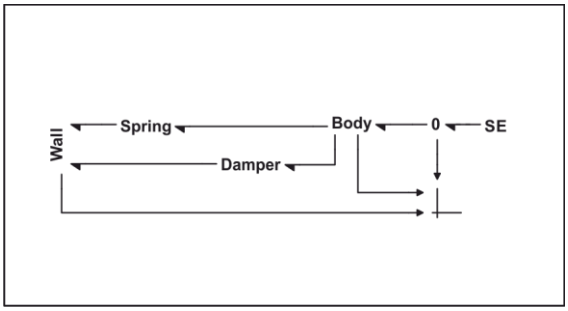
There are some connections between the elementary components that are permitted however (see the table below).

Components	Permitted connection to
I, C, R, SE, SF, Switch	1-junction, 0-junction, TF, GY
1-junction, 0-junction	All
TF, GY	all but TF and GY

-o-

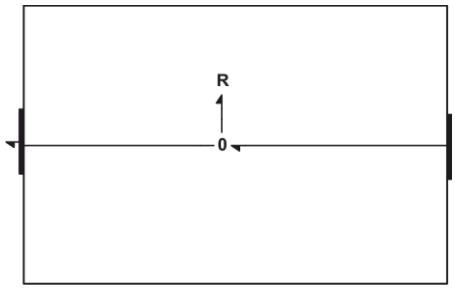
Changing Document Size

Consider **Body Spring Damper Problem** shown in the figure below.



Body Spring Damper Problem

If we open e.g. **Damper** component its document window appear as shown below.



Damper model

The document contains the Damper model. However, the size of the document that was created by the program is perhaps too large. We can change it if we wish. However, to do it we have first to disconnect the document ports. Thus, we click the left and right bonds and delete them. We will make the width of the documents smaller. To do it

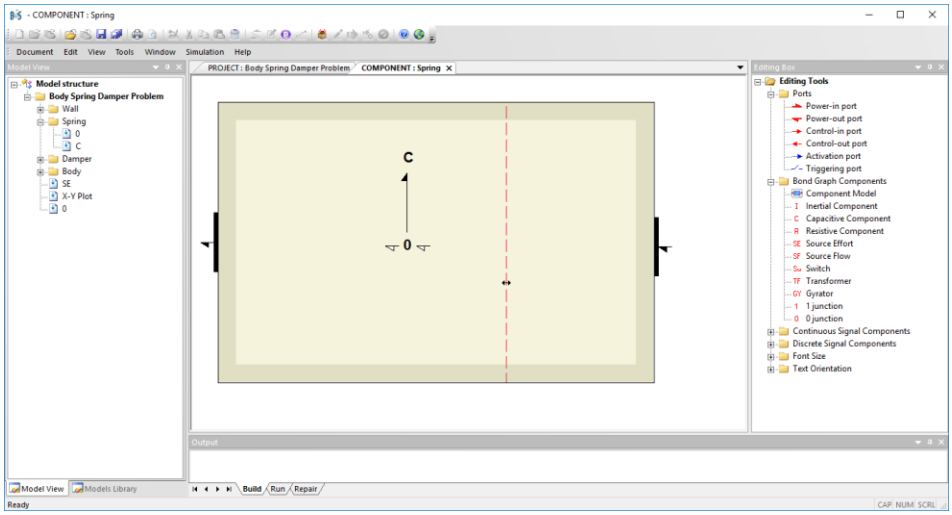
- We move the cursor over the right the document border edge. When it changes the form into the horizontal double arrow we press the mouse key and drag the right document edge to the left (as shown in the figure Dragging the right edge below), or right.
- When, the width of the document is OK we release the mouse. Now, the document window redraws and shows in the new size (see figure Final form of document below).

In a similar way we can change the document height.

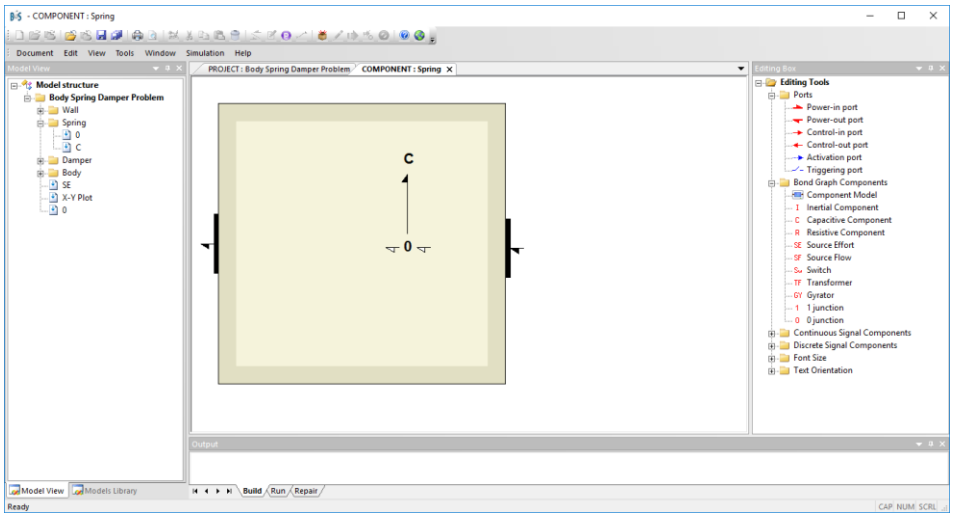
- We put the cursor over the bottom document edge and, when it changes to the vertical double arrow, we press the mouse key and drag the document edge up or down.
- When we are satisfied with the size we release the mouse and the document redraws in the new size.

We can also simultaneously change both the width and height of the document.

Dragging the right edge



The final form



In that case:

- We move cursor over the bottom-right document corner and when it changes to double arrow that makes angle of 45° with the horizontal, we press the mouse key and drag the document corner in north-west or south-east direction.
- When we achieve the suitable size we release the mouse and the document redraws in the new size.

After the document was resized we may reconnect the components whose ports were disconnected from the document ports by drawing the bonds again. We can also close the document and save the changes made (to its component storage or down to the project and disk).

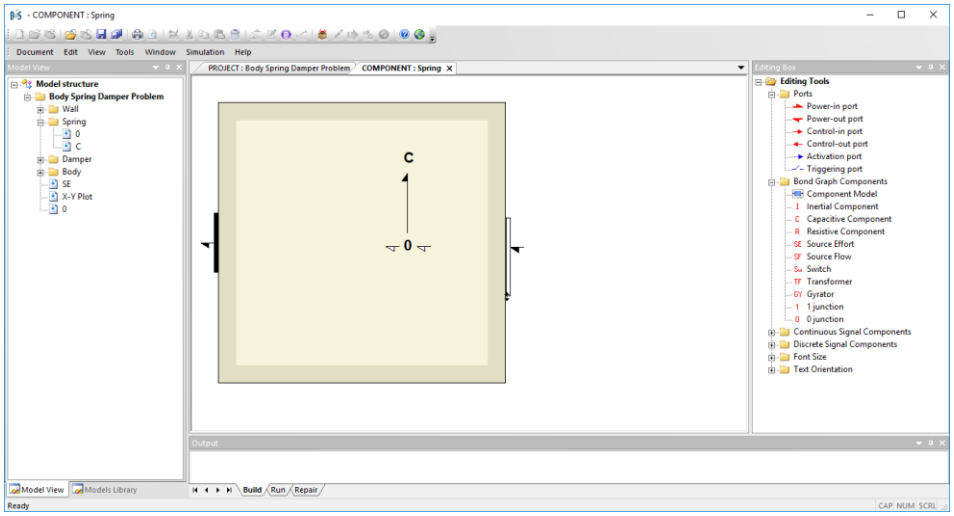
-o-

Changing Document Ports Sizes

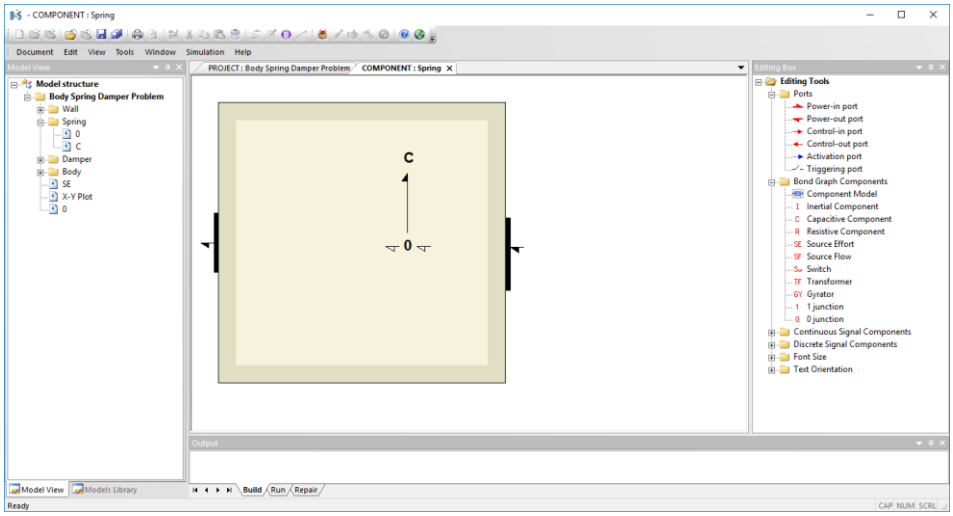
Similarly as we changed the document size we can change the width or height of the document port strip. Thus, e.g. to change the width of the right document port strip

- We put the cursor over the (narrow) bottom edge of the right document port, and when the cursor changes into vertical double-arrow form, we drag the edge downward or upward (as shown in the figure below).
- When we are satisfied we release the mouse key and the port redraws in the new size (see the second figure below).

Changing document port



The final form of the port

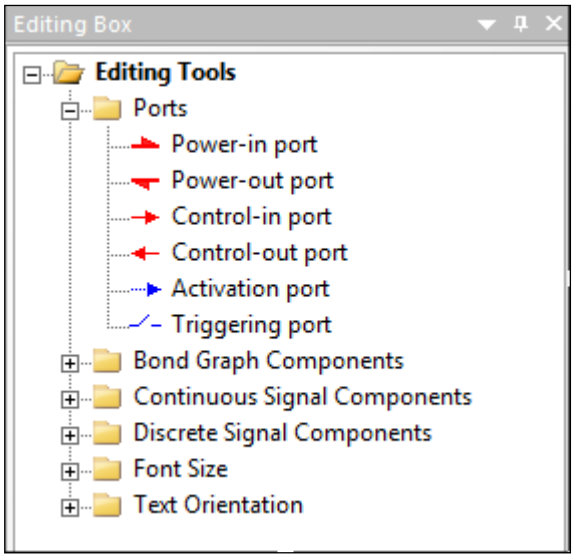


In the similar way we can change the size of the horizontal or corner document ports.

-o-

Inserting, Moving and Deleting Component Ports

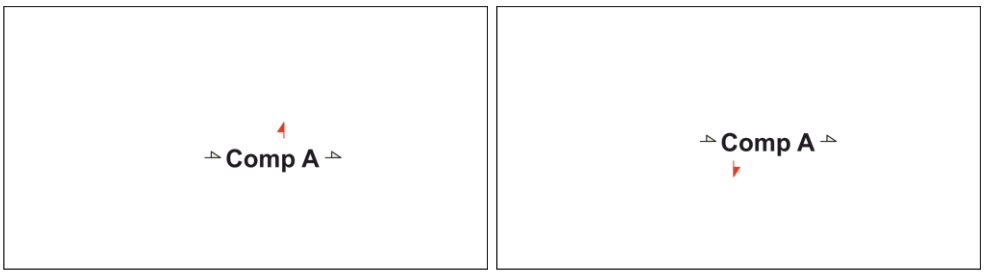
When a component is created by dragging and dropping Component Model from Edit Box (or some of the tools from Template Box) the component is created with already pre created ports. The same holds when creating the elementary component. We may add an additional port by dragging and dropping the corresponding port tool from **Editing Box**. In the figure below is shown enlarged a part of **Editing Box** with expanded **Ports** branch.



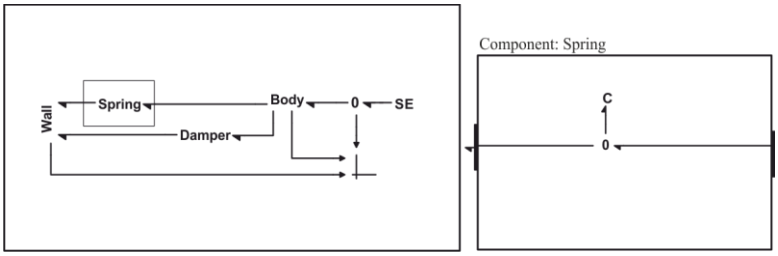
Expanded port tool branch

There are six tools of which first four are used in *Bond Graph* and *Continuous signals models* and the last two in *Discrete signals* models. If we wish to add e.g. power-in port we

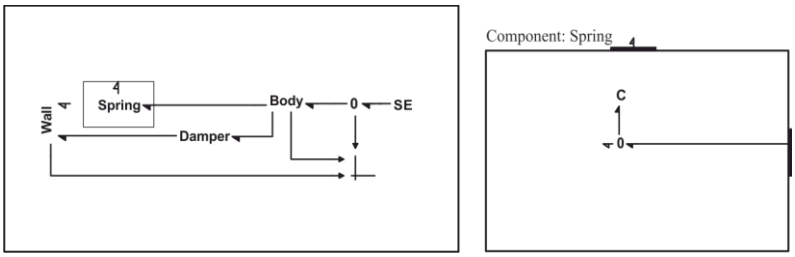
- Expand **Ports** branch in Editing Box and drag the corresponding tool. As we start dragging the cursor changes to "no parking" sign. When the cursor enters into the active document it changes into the cross. As we are over the component we wish the port to insert to a red bounding rectangle appears around the component. We place the cursor "cross" at the suitable place within the bordering strip around the component title and drop the port tool.
- The component redraws and the new port is shown in the selected state (in red color) as can be seen in the figure below on the left.
- We may drag the port around the component title (see figure below). As we start dragging the port the mouse cursor changes the shape so that it resembles the port we are dragging. The dragging is, however, confined to the strip around the component. As we moved the cursor to a suitable position we drop it. We must not place it near another port, otherwise operation fails.
- When the operation is completed the component is redrawn showing the port in the new position. It is selected (see figure on the right).

Inserting and moving the port 

When the component is connected, we may add the ports in the same way, i.e. without disconnecting it first. But to move an existing port around the component boundary we need first to disconnect it both internally and externally. As an example consider model of the spring in project **Body Spring Damper Problem** shown in the figure below. We open **Spring** and its document is shown on the right. Now to move the left port of **Spring** component we need to disconnect it from the **Wall**. But also we need to disconnect its left document port from the **0 junction**.

Model of the spring 

When we did this, we may drag the left **Spring** port around the component boundary and drop it somewhere, but not near the right port (see the figure below).

Move left spring port 

Similarly, to delete a port we need first to disconnect it both internally and externally. After that we select it (by clicking) and apply delete operation as with the other objects (by **delete** key or applying **Delete** command from **Edit** menu).

Note that some ports cannot be deleted or even moved. These are the ports created as fixed, e.g. when using the template tools.

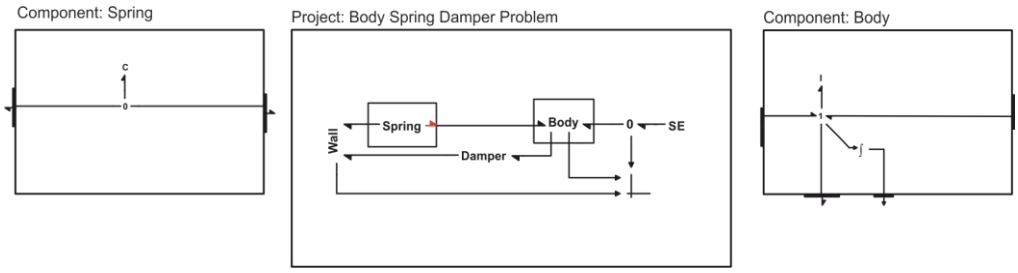
-o-

Changing Port Power Flow Sense

The power sense of the component port can be easily changed. For this it is even not necessary to disconnect it. Thus, to change power of the right **Spring** port in the figure below,

- Select the port (by clicking on it) and
- Select command **Change Power Sense** form **Edit** menu, or click on the corresponding bitmap button  in Toolbar.

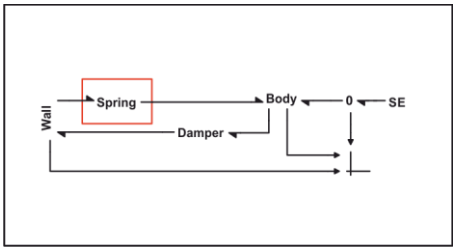
When we apply the command on the selected port the changes transmit on the both side of the port connections, on outside and inside, until an elementary component port is reached. Thus, in the central figure below when changing power sense of the right Spring port from power-in to the power-out, the changes are transmitted to the connected Body port, but also inside the Body document until **1 junction** port was reached (see figure on the right). Similarly, in the **Spring** document in the figure on left the corresponding document port changes, but also the sense of the connected **0 junction** port.



Changing port sense

Therefore, the power senses of all connected ports on both sides of the bond changes insuring that the port connection remains the proper, i.e. power-out ports remain connected to another power-in ports. This means that sometime the change must transmit even over some of the elementary components, e.g. transformers and gyrators.

We can also change the power sense of the all component ports simultaneously. To this end we need only to select the component and then apply command **Change Power Sense** as above.



Changing senses of the component ports

Note the changes apply to the ports connected to the component model ports on inside and outside as well. The other ports are not changed. Thus in the figures above the **C**'s and **I**'s ports are not affected. The control ports are not affected by this command, only the power ports.

-o-

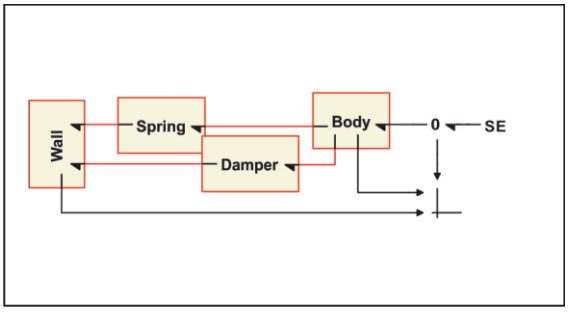
Selecting and Moving the Components

We can freely move a component within the document borders if it is not connected to the other components. Thus, to move a component we need to

- Disconnect it of the other components by deleting all connection bonds.
- Click on the component to select it; a component bounding rectangle appears in red indicating that the component is selected.
- To move the selected component drag it and drop somewhere, but not close to another component, nor near the document boundary strip. We can move the component using also the cursor keys.
- We can remove the selection by clicking outside of the component and the selection rectangle disappears.

We can also select and move a group of the components and their interconnecting bonds. Thus in the example below to select a group of the component

- Put mouse cursor to empty space in the document window and press and hold the key.
- Drag the mouse diagonally downward or upward; as we drag the mouse a selection rectangle appears. Drag it until it covers the component we wish to select.
- Release the mouse. The all components within the selection rectangle become selected, as well as the bonds connecting the selected components. If there are the components that are not included in the selection we may add them by pressing and holding ctrl key and clicking by mouse on them. In the same way may remove a component from the selection.



Selecting a group of the components

Note that in the figure above there are components that are not selected. The bonds connecting these components to the selected ones are not selected as well. Thus, the selected group of the components and interconnecting bonds are not free to move. To do it we have first to disconnect them and then repeat the selection again. Now the group behaves as a single component. We may put the cursor anywhere within a rectangle enclosing the group and press it and hold. As we drag the selected group a bounding rectangle appears that encloses the entire group. We have to

take care that when dragging it is within the document bounding strip. When we release the mouse the selected group appears in the new position and the bounding rectangle disappears. The component and the connecting bonds contained remain selected. Thus, we can move it again, or apply some other operation. To cancel the selection we click outside of the bounding rectangle.

-o-

Copying and Cutting to Component Buffer

BondSim supports the **Component Buffer**, which has a similar function as Windows **Clipboard**. It serves to store the components and group of the components that we copy, or cut, into it. (Really no components are moved, only their links.)

When we select a component we may copy it into the **Component Buffer** even if it is connected to the other components. The operation makes a deep copy of the component in the memory and store the link to the component into the buffer. The deep copy means that the complete component tree is copied, not only the component object. Thus, to copy a component to the buffer

- Click on the component to select it; the component bounding rectangle appears in red indicating that the component is selected.
- Select **Copy** command from **Edit** menu, or click **Ctrl+C** key combinations (press and hold the **ctrl** key and click on **C** key), or click on Copy bitmap in the toolbar.
- A deep copy operation was silently made in the background and the link to the copy of the component object in the memory is moved into **Component Buffer** under the component title (name). If there is already a component with same name in the buffer the program asks to change the name.

  We can also copy a group of the components to **Component Buffer**. We need first to select a group of the components, as was described before. Now, when we apply the **Copy** operation as described above deep copies of all encountered components and bonds are made silently in the memory. Also a temporary root component is created, which contains the links to the copies of the selected components and interconnected bonds. In this way a tree containing the selected group is created silently in computer memory. The program asks the modeler for the name under which the link to the selection copy root will be stored in **Component Buffer**. When the name is accepted the operation completes. Thus, the **Copy** operation is pretty involved, but the modeler basically is not aware of this. After selecting the components and the connecting bonds group, **Copy** operation is applied, and she or he only needs to supply the copy group (a block) name. All others were done automatically in the background.

  We can also **Cut** a component into the **Component Buffer** and remove it from the document. However, to apply this operation the component must be disconnected from the other components in the document. Thus to cut a component and remove it into the buffer
- Disconnect the component by deleting all bonds connecting it to the other components ports including the active document ports.

- Click on the component to select it; a component bounding rectangle appears in red indicating that the component is selected.
- Select **Cut** command from **Edit** menu, or press **Shift+Delete** key combinations (press and hold the **shift** key and click on **delete** key), or click on **Cut** bitmap on the toolbar.
- The link to the component in the memory is moved into **Component Buffer** under the component title (name) and the component is removed from the document and it is redrawn. If there is already a component in the buffer under the same name the program asks for another name.

  In similar way it is possible to remove a group of the components and the connecting bonds into **Component Buffer**. However, as with move operation, the selected components should be disconnected from the other components or the document ports. The operation is, however, simpler then the **Copy** operation. When we select a group of components and the connected bonds and apply **Cut** operation as described above, the program asks for the name under which the selection will be hold in the buffer. When the name is accepted, similarly as in **Copy** operation, the selection root component is created silently in the background, and the links to all selected components and bonds are moved into this component. The link to this component is stored in the **Component Buffer** and now all selected components and bonds can be removed from the document and document is redrawn. The modeler again is not much involved in this complete operation. She or he just needs to made selection, apply the operation, and supply the name.

-o-

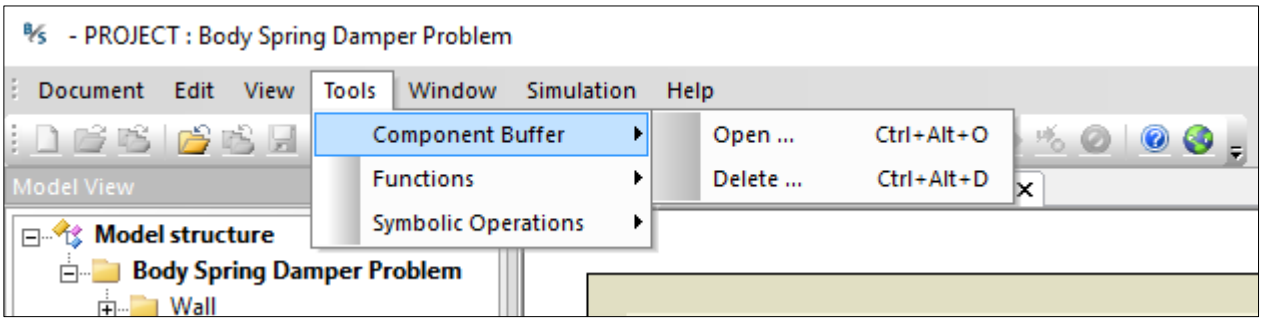
Reviewing and Deleting from Component Buffer

The components stored in **Component Buffer** can be used any time the user wishes. The buffer is saved between BondSim sessions. Thus, it can be used as a source of the component models when developing a modeling project. This is different from Windows **Clipboard**, which generally is not saved between the Windows sessions.

To review the component in **Component Buffer**

- Click on **Tools** menu and in the menu that drops down put mouse on **Component Buffer** submenu and select **Open** command (see the figure below).
- The **Component Buffer** dialog windows open with the list of component that buffer contains. To review a component from the list select it and click on **View** button, or double click the component.
- The component root document appears in the central part of the program window. We can walk now through the component tree. When finished we can close its document window.

Component Buffer menu



In similar way we can remove a component from **Component Buffer**:

- Click on **Tools** menu and in the menu that drops down put mouse on **Component Buffer** submenu and select **Delete** command (see the figure above).
- The **Component Buffer** dialog windows open with the list of component that buffer contains. To delete a component from the list select it and click on **Delete** button.
- The component is not removed right now, but is moved into **Waste Bin** buffer.

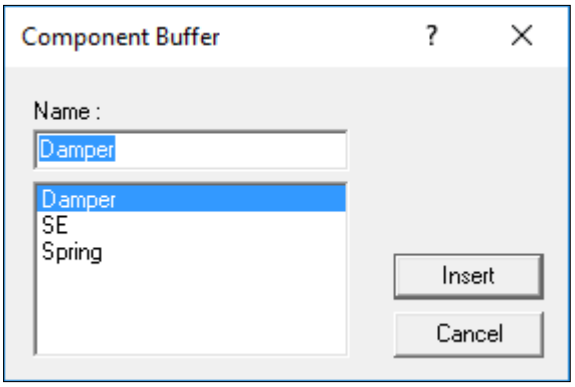
-o-

Inserting from Component Buffer

The components, which are stored in **Component Buffer**, can be inserted back into the same document from which they are copied /cut to the buffer, or into some other document, or even into a different project. The **Component Buffer** is saved between different BondSim sessions. A component or group of the components (block) can be inserted into the current active document windows. It is similar to Windows **Paste** command and uses the same shortcut command (Ctrl+V).

To insert a component or a group of components:

- Select **Insert** command from **Edit** menu, or press **Ctrl+V** key combinations (press and hold the **ctrl** key and click on **V** key), or click on **Insert** bitmap on the toolbar. A similar Component Buffer dialog as with previous commands appears, but using **Insert** button (see below).
- Select the component and click to **Insert** button. The dialog closes and we need to move the cursor into the current active document.
- At beginning cursor changes into "no parking" sign, but as soon as the cursor enters the document bounding rectangle it changes the form into the "cross" and around it appears a rectangle which corresponds to the bounding rectangle of the inserting component or the group of the components. We need to move the cursor into the document so that it is not close the inside boundary strip. When we click by the mouse the rectangle is replaced by the selected component or group of components.

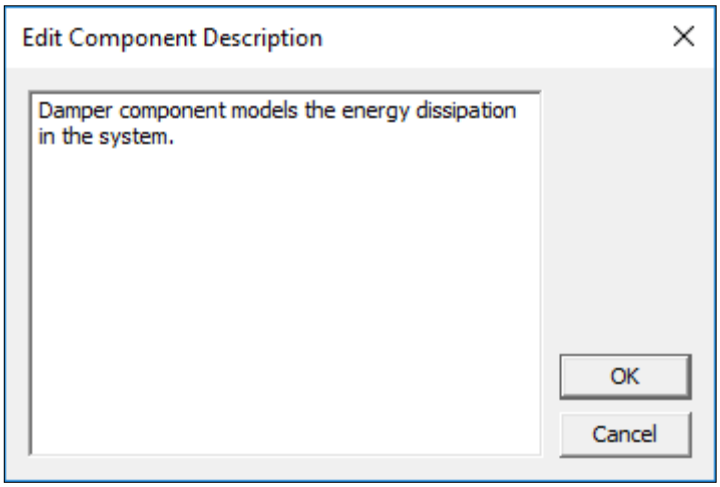


Insert Component Buffer dialog

-o-

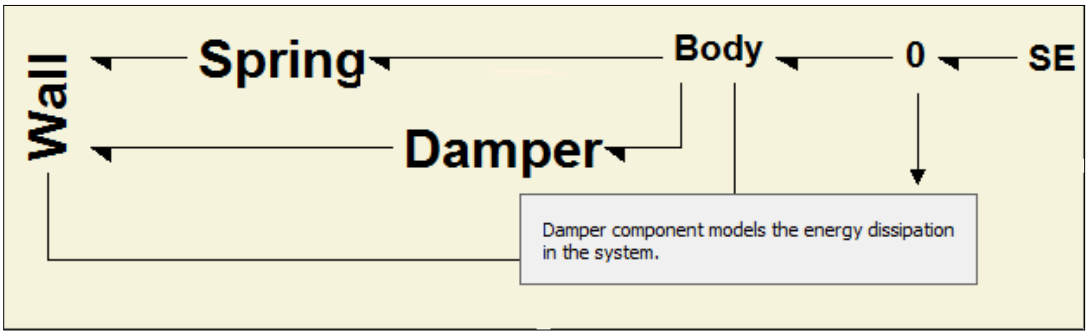
Editing Description Text

BondSim program supports also the component tooltips. This is a small window that appears when mouse is moved across a component title. To edit the contents of the tooltip widow we select the component, right click on it, and select **Edit Description**. The component may be connected this operation to apply. A widow appears in which we can edit a text, e.g. a short description of the component function (see below).



Edit component description

When editing is finished we accept it by clicking on **OK** button. The next time we move mouse over the component's title the tooltip appears as shown below.

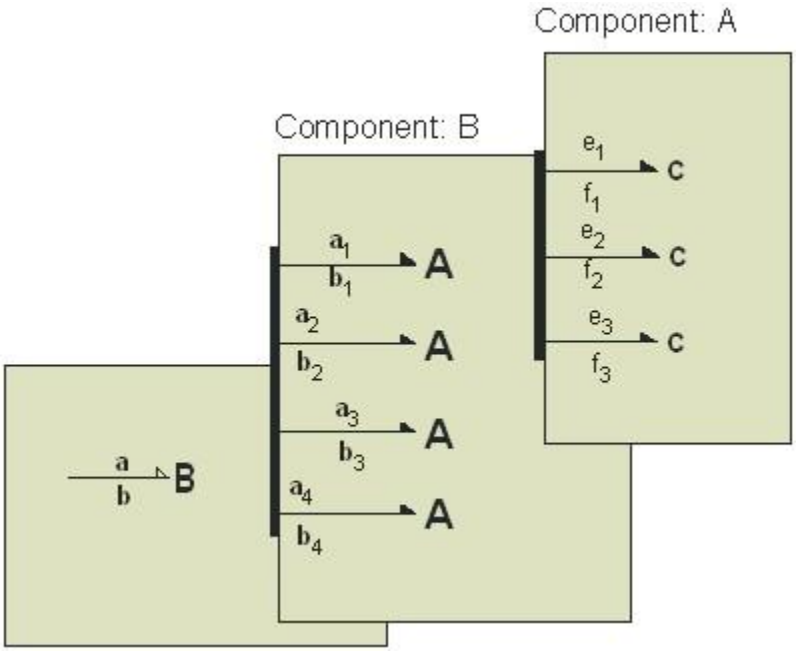


Component toolbox

-o-

Vector and Higher-Dimensional Quantities

Using concept of simple (scalar) effort and flow variable associated with the ports of elementary components it is easy to construct the higher-dimensional variables associated with external ports of the corresponding components. As an example consider model of component **B** as shown in the figure below. This component consists of four components **A** connected internally to its port (the middle document). Every component **A**, on other hand, consists of three elementary (capacitive) components **C**, which are connected internally to its port as shown in the right document.



Compound ports structure

Starting from the document **A** we associate with the ports of components **C** the pairs of effort e_i and flows f_i (i = 1, 2, 3). Because the document port of **A** is internally connected by the bonds to three ports of the components **C**, we take that the port of **A** has dimension 3. Thus, we treat the port of components **A** as compounded, each consisting of three sub ports. The sub ports are ordered (from the top to bottom or from left to right ones). We can, thus, associate each port of the components **A** with a pair of 3 dimensional effort and flow (row or column) vectors. Hence,

$$\mathbf{a}_i = \begin{pmatrix} e_{i1} & e_{i2} & e_{i3} \end{pmatrix}, \quad \mathbf{b}_i = \begin{pmatrix} f_{i1} & f_{i2} & f_{i3} \end{pmatrix}, \ (i = 1, ...,4)$$

or

$$\mathbf{a}_i = \begin{pmatrix} e_{i1} \\ e_{i2} \\ e_{i3} \end{pmatrix}, \quad \mathbf{b}_i = \begin{pmatrix} f_{i1} \\ f_{i2} \\ f_{i3} \end{pmatrix}, \ (i = 1,...,4)$$

In same way, the port of component **B** is also compounded having dimension 4. We, thus, associate this port with a pair of higher dimensional effort and flow vector quantities

$$\mathbf{a} = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 & b_2 & b_3 & b_4 \end{pmatrix}$$

or, alternatively,

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

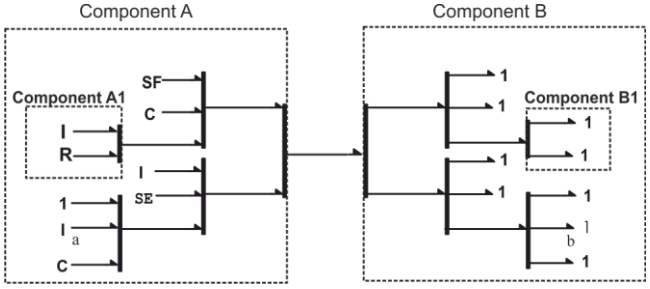
Now if we make substitutions in the last relationships we find that pair of variables that describes the processes seen at the port of element **B** can be interpreted as 12 dimensional row or column vectors, or either 3x4 or 4x3 matrices of the elementary components efforts or flows, e.g.

$$\mathbf{a} = \begin{pmatrix} e_{11} & e_{12} & e_{13} \mid e_{21} & e_{22} & e_{23} \mid e_{31} & e_{32} & e_{33} \mid e_{41} & e_{42} & e_{43} \end{pmatrix},$$
$$\mathbf{b} = \begin{pmatrix} f_{11} & f_{12} & e_{13} \mid f_{21} & f_{22} & f_{23} \mid f_{31} & f_{32} & f_{33} \mid f_{41} & f_{42} & f_{43} \end{pmatrix}$$

or, alternatively,

$$\mathbf{a} = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \\ e_{41} & e_{42} & e_{43} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \\ f_{41} & f_{42} & f_{43} \end{pmatrix}$$

Thus, the components ports are compound, consisting of the sub ports. It is associated with the ports the complex structure of the effort and flow variables. This however comes with price. The higher-dimensional ports can be connected by the bonds only if they have the same structure (see figure below).



Joining the compound ports

Thus the connected components ports, must have not only the same dimensions (two in this case above), but the dimensions of the internally connected ports should be the same. Thus, the internal connection of the connected ports should be the symmetrical. With some experience it is not difficult to achieve this.

This requirement is mathematically logical. The higher dimensional mathematical constructs (such as vectors, matrices, tensors) could be equated only if they have the same structure. Thus we may equate a m×n matrix **A** to a m×n matrix **B**, etc.

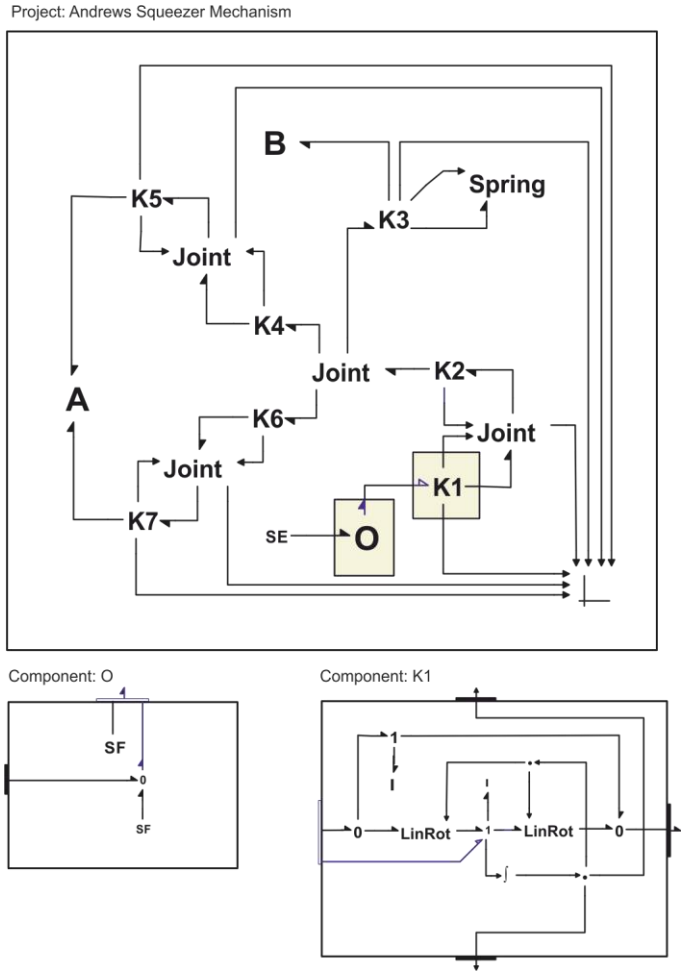
-o-

Searching for Connected Port

The ports are often connected over several components and their documents. To simplify determining the port which is connected to a port we may select command **Show Joined** from **View** menu. The same command can be applied using the corresponding **Toolbar** bitmap button  . To illustrate use of this command consider project **Andrews Squeezer Mechanism** from the **Project library**.

To apply to this project

- Start BondSim and activate Library widow.
- Expand Project and right click on **Andrews Squeezer Mechanism** and select Open command. The project document opens as shown below.
- Open component **O** and select the upper port of **0 junction** (see Component: O in the figure below). Apply command *Show Joined* as explained above.



Joined ports in Andrew's mechanism

We wish to find the port which is connected to the selected port. The component **O** is model of a

shaft, which can rotate in the bearings. What we are really trying to find is the component the torque developed by the shaft is the applied to. Search procedure is applied and the connected bonds and ports are shown in blue color. The search procedure is:

- When we apply the command the search starts at the selected port (Figure Component: O), then continues by the connecting bond to the right sub-part of document port of **O** and goes out of the component (see the project document in the upper figure).
- It continue by the bond to the connected port of component **K1** and goes inside it (Component: K1 in the figure above);
- Component **K1** is opened and the connection goes out form the bottom sub-port of the left document port, and by the bond ends at lower-left **1 junction** port.

Note that the component **I** is connected to the same **1 junction**. This represents the rotation inertia of body **K1**. Thus, the shaft torque is applied to body **K1**, which rotates about the shaft **O**. Note also that connected ports of bodies **O** and **K1** have the same structure, i.e. they have two connected bonds each. Thus, the right bond on inside of **O** is the same that goes out form the bottom of the left document port of **K1**.

-o-

Elementary Bond Graph Components

The Fundamental Types

The elementary Bond Graph components are the fundamental components used to define models of the basic physical processes. There are nine such components as listed below.

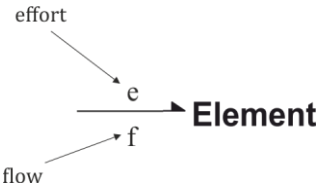
Inertial Component	It is used to model the inertial processes in mechanics or inductivity in electrical circuits.
Capacitive Component	It is used to model elastic deformation such as in mechanical springs or accumulation of the charges such as in electrical capacitors.
Resistive Component	This component is used to model friction in mechanical system or dissipation processes in electrical resistors.
Source Effort and Flow	Models power sources or sinks, such as voltage sources, gravity force, pumps, etc.
Transformer and Gyrator	Model two basic forms of transformations appearing in the physical world.
Effort and Flow Junctions	Describes two basic forms of the branching – of efforts and flows.

The elementary components are created in a model document by dragging and dropping from **Edit Box** as described in Creating, Editing and Deleting Components.

-o-

Power Variables and Physical Analogies

Processes seen at the ports of elementary components are described by a pair of external variables: *effort* e and *flow* f (see figure below). These variables are called *power* variables because their product is power transferred across the port.



Elementary Component

In addition to these external port variables, in some components there are also the internal variables, which represents the state of the process. Such variables are generalized *momentum* (in inertial components), and *generalized displacement* (capacitive components). These variables represent accumulation of the efforts and flows respectively over time. The typical association of variables in various domains is given in table below:

Domain	Effort	Flow	Momentum	Displacement
Mechanical translation	Force	Velocity	Momentum	Displacement
Mechanical rotation	Torque	Angular velocity	Angular momentum	Angle
Electrical	Voltage	Current	Flux linkage	Charge
Hydraulic	Pressure	Volume flow rate	Pressure momentum	Volume
Thermal	Temperature	Heat flow	-	Heat energy

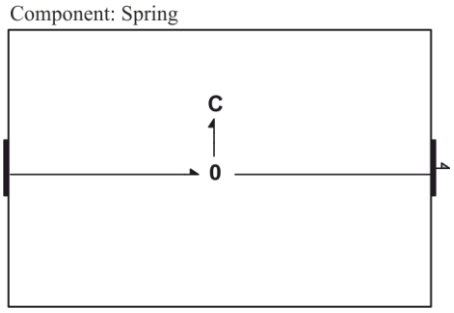
It should be noted that thermal effort and flow variables, as defined in the last row, are not the true power variables because their product is not power. The Bond graphs corresponding to variables having this property are usually termed pseudo-bond graphs. It is also possible to define true thermodynamic power variables (using concept of entropy).

Association given by the above table is one of possible physical analogies, and corresponds to electromechanical analogy in which force in mechanical systems corresponds to voltage in electrical systems (force-voltage analogy). In Bond Graph approach the above analogy, in which efforts are *intensities* and flows are *extensities*, is usually preferred.

The processes in the elementary components are seen at their ports and are described by suitable relations between the port variables. These are the element *constitutive* relations and are stored in their ports. The elementary components, thus, do not need a separate document to define its model and represent the separate entity. In the model tree they are its leaves.

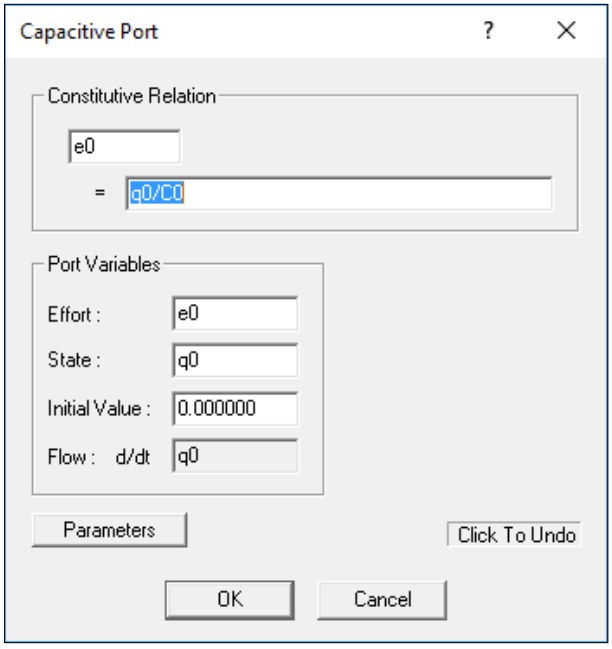
To see this we will open **Spring** component in the previously considered Body Spring Damper problem:

- Start BondSim.
- Put mouse over the second bitmap on the left in toolbar. A tooltip appears displaying **Open Project** command with information what this command does. Click on the bitmap.
- In the dialog Open Project in dialog that opens double click the project **Body Spring Damper Problem** in the list of existing programs, or select the project and click OK button.
- In project document that opens in the central part of the frame double click the **Spring** component. The **Spring** document opens and has the form shown below.



Model of Spring

To access the constitutive relations describing processes in **C** component at its port we *open* the port by double clicking, or select the port and choose command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). A dialog appears as shown below.



Capacitive port dialog

The constitutive relations depend on the type of the component and its port. Here we see that there are three ports variables, two external power variables – *effort* and *flow*, and the internal one – *state*. The constitutive relation for this component relates the effort (the spring force) and state (the spring deformation). C0 is a parameter, which is defined here as the reciprocal of the spring stiffness. The parameter can be defined using Parameter button in the component. The parameters are defined at the level of the component, not in the port, and thus are acceptable by the other component ports if they exist. But, it is seen in the component also if it is defined at any model document downwards from the component until the project document, i.e. along the branches from the component leaf down to the project root.

When we edit the constitutive relation and click OK button the program first parses the constitutive relation to find if it is well defined. All symbols must be either port symbols shown in the dialog, time *t* or the parameters. The parameters are searched for as described in Define Parameters. Also only supported operations should be used (see Description of Constitutive Relations). After the parsing is successfully completed the dialog closes and the port variables and constitutive relation are stored in the port. Note that we may use any appropriate names for the port variables. They are used only locally and serves to define the constitutive relation in a user friendly way. (The program internally creates the corresponding globally unique variables; they are hidden because they are very long and not user friendly, and are used only internally.)

-o-

Description of Constitutive Relations

The constitutive relations of the elementary components generally have the form:

Variable = algebraic expression

where the algebraic expression is formed of the variables, constants, parameters and operators.

The accepted variables are the port power and control variables and time *t* (which is the reserved symbol). The lower and upper case symbols differ; thus, *m* and *M* are not the same variable or parameter. All standard forms of the constants are acceptable, e.g. -120, .058, 3.5609, -2.85e-12. The supported operators are given in the table below. The table also defines the precedence order of the operators. Thus, the function call has the highest and the arithmetic if the lowest precedence. The operator precedence is similar as in C language. The supported elementary functions are given in the next table.

Table of supported precedence and their precedence

Operator	Meaning
( )	Function call
!	Logical not
+ -	Unary + or -
^	Exponentiation
*** / %**	Product, division and modulus
< > <= >=	Relational operators
+ -	Addition and subtraction
& |	Logical AND and OR
? :	Arithmetic if

Table of supported elementary functions

Name	Form	Domain	Range
Sine	sin(x)		
cosine	cos(x)		
tangent	tan(x)		
arcsine	asin(x)	[-1, 1]	[-π/2, π/2]
arccosine	acos(x)	[-1, 1]	[0, π]
arctangent	atan(x)	(-∞,∞)	[-π/2, π/2]
hyperbolic sine	sinh(x)		
hyperbolic cosine	cosh(x)		
hyperbolic tangent	tanh(x)		
inverse hyperbolic sine	argsh(x)	(-∞,∞)	(-∞,∞)
inverse hyperbolic cosine	argch(x)	[1,∞)	[0,∞)
inverse hyperbolic tangent	argth(x)	(-1,1)	(-∞,∞)

absolute value	abs(x)		
e raised to a specified power	exp(x)		
natural (base **e**) logarithm of a specified number	log(x)		
base 10 logarithm of a specified number	log10(x)		
square root of a specified number	sqrt(x)		
Sign function	sgn(x)		
the largest integer ≤ the specified number	floor(x)		
the smallest integer ≥ the specified number	ceil(x)		
Rounds a real value to the nearest integral value	round(x)		

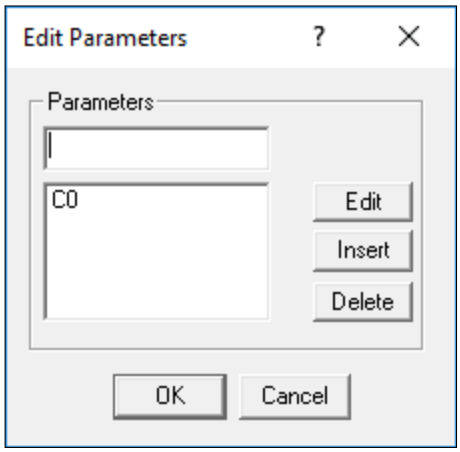
The common arithmetic, relational and logical operators are permitted. The exponentiation operator is defined also (as in Fortran). If-else conditional statements can be created using C language like „question" operators ?:.

The most of common elementary mathematical functions are supported (see table of elementary functions below), such as sin, cos, log, asin, acos, etc. Also the tabular and symbolically defined functions are supported.

-o-

Define Parameters

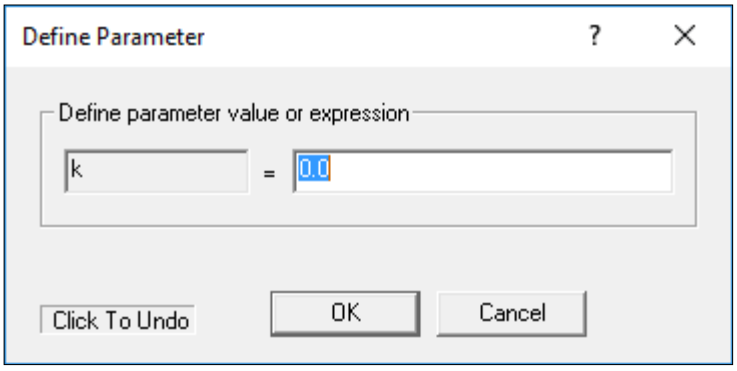
The parameters can be defined at the level of the component by the port dialog as discussed. By clicking the **Parameter** button parameter edit dialog appears as shown below.



Parameter dialog window

- To create a new parameter type in the name of the parameter in text editing box and click on **Insert** button.
- To edit an existing parameter select the parameter in the list and click **Edit** button.
- To delete the parameter select it and click **Delete** button.

To insert a parameter k, we need thus to write k in the edit box and click **Insert**. A following dialog for editing the parameter definition opens.



Parameter definition dialog

Now we can define the parameter value. It can be done by typing its value, or an expression in terms of already defined parameters and constants. When finished we click **OK**. Before its closing the parameter definition is parsed first to find if it is well defined. Next we need to click **OK** in the previous dialogs. After that the new parameter or reedited already defined parameter is stored in the component in the corresponding parameter list.

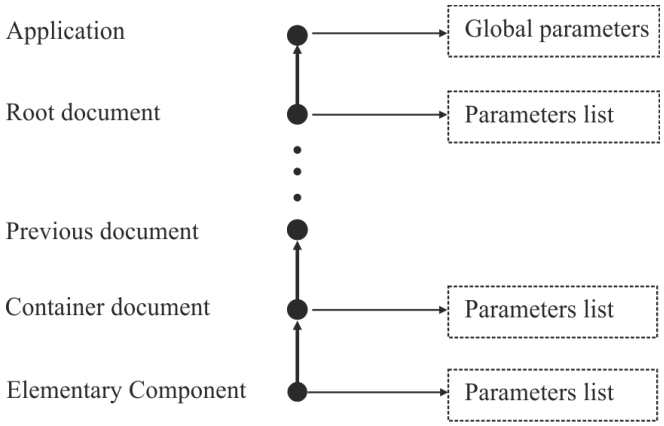
The parameters can be also defined in the project or component document by applying command **Model Parameters** form **Edit** menu, or clicking on short-cut bitmap **P** in **Toolbar**. The same dialogs appear as described above.

It is important to understand how parameters are designed in BondSim. As described previously the parameters can be defined in the elementary components or in the components or project documents. In addition there are the parameters defined at the level of BondSim application. These are the global parameters (see table below) and are visible in any open project.

Global parameters

Parameter	Value	Meaning
g	9.80665	Gravity constant (m/s^2)
PI	3.141592653589793	π constant
BOLTZMANN	1.380662e-23	Boltzmann constant (J/K)
ECHARGE	1.6021892e-19	Electron charge (C)
EGSI	1.12	Energy gap in Si (V)
TNOM	300.0	Normal working temperature (K)
BG_EPSILON	4 DBL_EPSILON	BondSim epsilon
BG_MAX	DBL_MAX	BondSim maximum value
MEXP	50.0	BondSim maximum exponent
BG_e	2.718281828459045	Basis of natural logarithm
DBL_EPSILON	2.2204460492503131e-016	Double epsilon
DBL_MAX	1.7976931348623158e+308	Maximum representable double value

When a parameter is used in a constitutive to find its definition the following search path is used (see below).



The parameter is searched for first in the component where it is used, then in the document, which contains the component, then the previous documents until the project root document is reached, and finally in the application itself. If there are two or more definitions only the first one is taken that is the closest to its use. This definition, thus, hides all lower level definitions.

-o-

Inertial Component

Model:

Inertial components can have one or more power ports.

Function:

This component is used to describe inertia of a body in translation or rotation, or inductivity of an electrical coil. In addition to effort and flow at the ports there is an internal variable, *generalized momentum* p, defined by the relationship

$$e = \frac{dp}{dt}$$

The generalized momentum can be viewed as the accumulation of the effort over time:

$$p(t) = p(0) + \int_0^t e dt$$

This way the momentum expresses previous history of the effort at the port considered and plays the role of a state variable, i.e. a variable defining the state of the process.

The constitutive relation for the element defines relationships between momentum and flow. In the most applications it has the linear form

$$p = I \cdot f$$

where *I* is a parameter. In body translation it is the body mass *m*, in rotation about a fixed axis it is moment of inertia of the body *J* with respect to this axis, and in electrical coils it is coefficient of inductance *L*. The constitutive relation can also have a nonlinear form.

Processes represented by the inertial elements are characterized by accumulation of power flowing into the element in form of dynamic (kinetic) energy

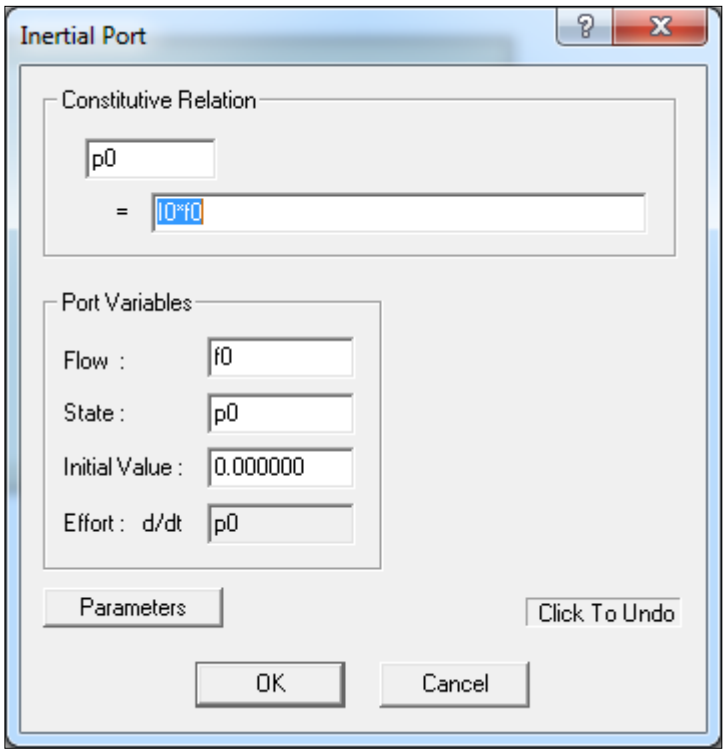
$$E(t) = E(0) + \int_0^t e f dt$$

After substitution of effort as given by the first equation above we find

$$E(p) = E(p_0) + \int_{p_0}^{p} f dp$$

Editing the Constitutive Relation

To edit the port constitutive relation we need to open its port , i.e. either by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The inertial dialog appears as shown below.



Inertial port dialog

The constitutive relation is already predefined in the linear form using predefined *flow* and *state* (momentum) variables names and parameter I0. Also the initial state is set to 0.0. We may change these as we wish. The constitutive relation can be defined by the state (momentum) as function of the flow, or the flow as function of the state (momentum). If the changes were made they may be accepted by clicking OK. Otherwise click Cancel.

-o-

Capacitive Component

Model:

Capacitive component can have one or more power ports.

Function:

This component is used to model mechanical springs, electric capacitors and similar processes. In addition to power variables at the component port, there is also an internal variable, *generalized displacement q*, defined by relationship

$$f = \frac{dq}{dt}$$

The generalized displacement can be viewed as the accumulation of the flow over time:

$$q(t) = q(0) + \int_0^t f dt$$

This way the displacement expresses previous history of the flow at the port considered and plays the role of a state variable, i.e. a variable defining the state of the process.

The constitutive relation for the element defines relationships between displacement and effort. In many applications it has the linear form

$$q = C \cdot e$$

where *C* is a parameter. In electric capacitors this is common capacitance parameter. However, in mechanical spring elements the constitutive relation typically has the form

$$e = K \cdot q$$

where *K* is the spring stiffness. The constitutive relation can also have a nonlinear form.

Processes represented by capacitive elements are characterized by accumulation of the static (potential) energy

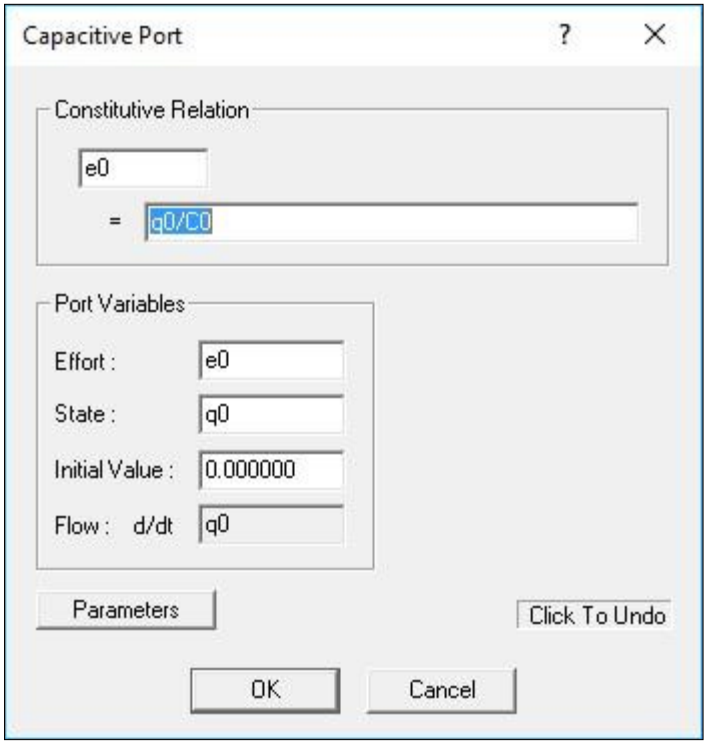
$$E(t) = E(0) + \int_0^t e f dt$$

After substitution of the flow as given by the first equation above we find

$$E(q) = E(q_0) + \int_{q_0}^{q} e dq$$

Editing the Constitutive Relation

To edit the port constitutive relation we need to open its port , i.e. either by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The inertial dialog appears as shown below.



Capacitive port dialog

The constitutive relation is already predefined in the linear form using predefined *effort* and *state* (displacement) variables names and parameter C0. Also the initial state is set to 0.0. We may change these as we wish. The constitutive relation can be defined by the effort as function of the state (displacement), or the state (displacement) as function of the effort. If the changes were made you may accept them by clicking OK. Otherwise click Cancel.

-o-

Resistive Component

Model:

Resistive component can have one or more power ports.

Function:

This component is used for modeling the energy dissipation by friction in mechanical systems and resistors in the electrical. The constitutive relation for the element defines relationships between effort and flow. In many applications a linear form suffice

$$e = R \cdot f$$

where *R* is a parameter. Such relationship implies that power flowing into the element

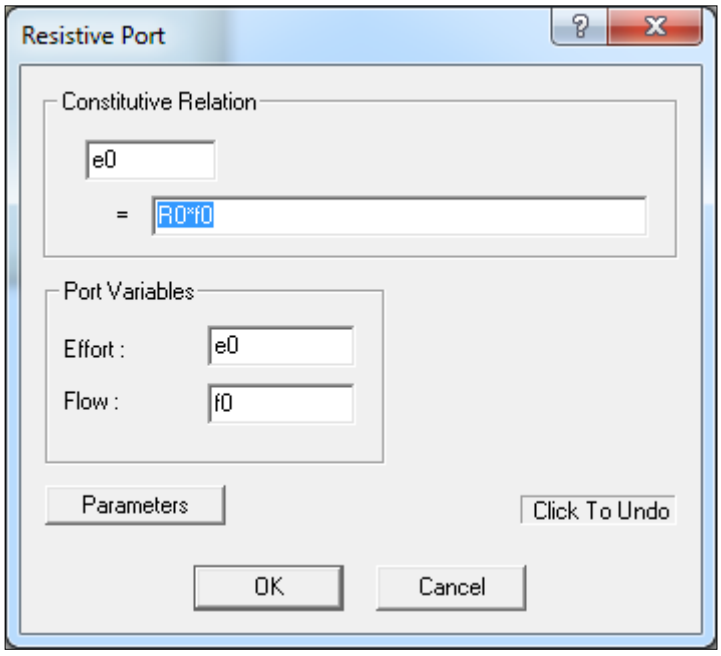
$$P = e \cdot f = R \cdot f^2$$

and is greater or equal to zero (if R>0). Thus, such a component is characterized by dissipation of the energy. It should be noted that constitutive relation of the element can also be nonlinear, but typically such components also dissipate the energy.

Editing the Constitutive Relation

To edit the port constitutive relation we need to open its port , i.e. either by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The inertial dialog appears as shown below.

The constitutive relation is already predefined in the linear form using predefined *effort* and *flow* variables names and parameter R0. The constitutive relation can be defined by the effort as function of the flow, or the flow as function of the effort. Relation may be nonlinear as well. The changes you made can be accepted by clicking **OK**. Otherwise click **Cancel**.



Resistor port dialog

-o-

Sources

Models:

- **Source Effort**

- **Source Flow**

Source Effort and *Source Flow* are single port components.

Function:

Represents power generation (or power sinks) such as voltage and current sources, certain type of force sources (e.g. gravity), volume flow sources (such as pumps) etc.

The constitutive relation for Source Effort component defines efforts generated at its port and is independent of flow at this port. Similarly, constitutive relation for Source Flow defines flow generated at its port and is independent of effort at this port. Thus these sources represent idealized component capable of generating or receiving unlimited power at its port. In many applications constant effort or flow sources suffice:

- **Source Effort (SE)**

$$e = E0$$

where E_0 is the parameter.

- **Source Flow (SF)**

$$f = F0$$

where F_0 is the parameter.

However, efforts or flows generated by the sources often are not constant, but are functions of time. Thus, e.g. constitutive relation of Effort Source which generates sinusoidal voltage may be set by

$$e = a\sin(\omega t + \varphi)$$

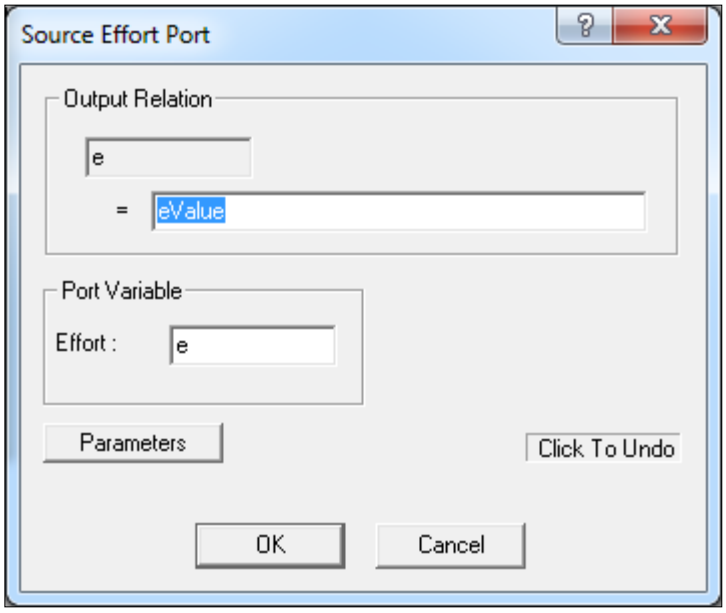
where a is the amplitude of generated voltage, ω is its frequency and φ the phase.

Editing the Constitutive Relation

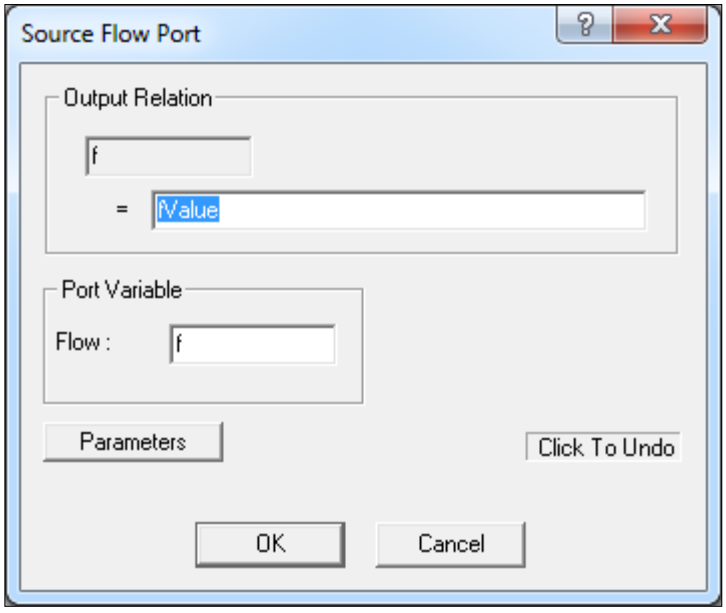
To edit the constitutive relations of Source Effort or Source Flow ports we need to open the port , i.e. either by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The corresponding dialog appears as shown below.

The constitutive relation is already predefined as the constant value of the corresponding effort or source variable. The names of *effort* or *flow* variables are already predefined, as well the

parameters E0 or F0. These values could be changed at will. The changes you made can be accepted by clicking **OK**. Otherwise click **Cancel**.



SE port dialog



SF port dialog

-o-

Transformer and Gyrator

Model:

- **Transformer**

- **Gyrator**

The components have two power ports and it is assumed that power flows through the component.

Function:

They represent two kinds of transformations of the efforts and flows. The constitutive relations for transformers relate effort to effort, and flow to flow. In gyrators, however, the constitutive relations relate efforts and flows across the component.

An important property of these components is conservation of power flow. In other words, the power inflow is equal to the power outflow.

The constitutive relations:

- **Transformer**

$$\left.\begin{aligned} e_1 &= k\, e_0 \\ f_0 &= k\, f_1 \end{aligned}\right\},$$

where *k* is the transformation ratio.

- **Gyrator**

$$\left.\begin{aligned} e_1 &= k\, f_0 \\ e_0 &= k\, f_1 \end{aligned}\right\},$$

where *k* is the gyration ratio.

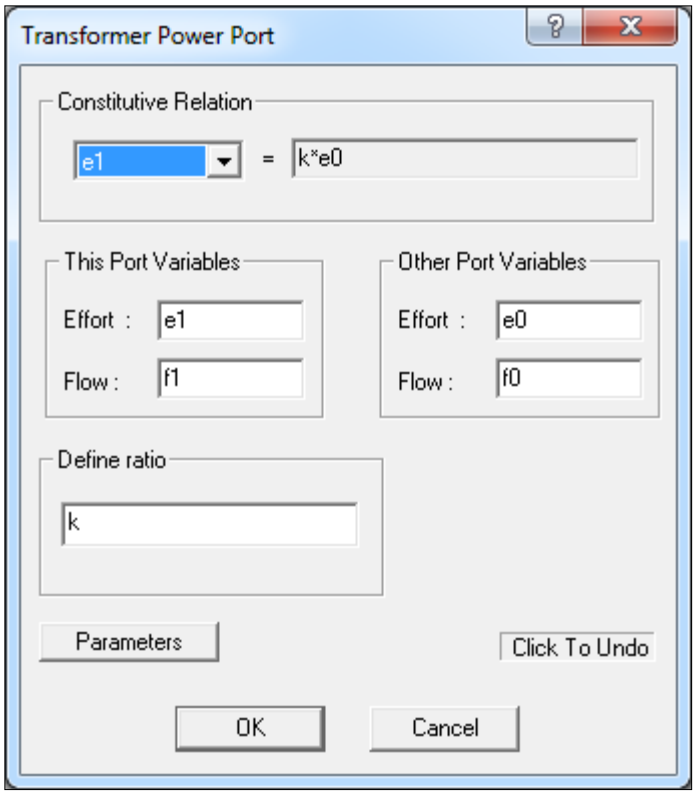
Note that in both cases we have

$$e_1 f_1 = e_0 f_0$$

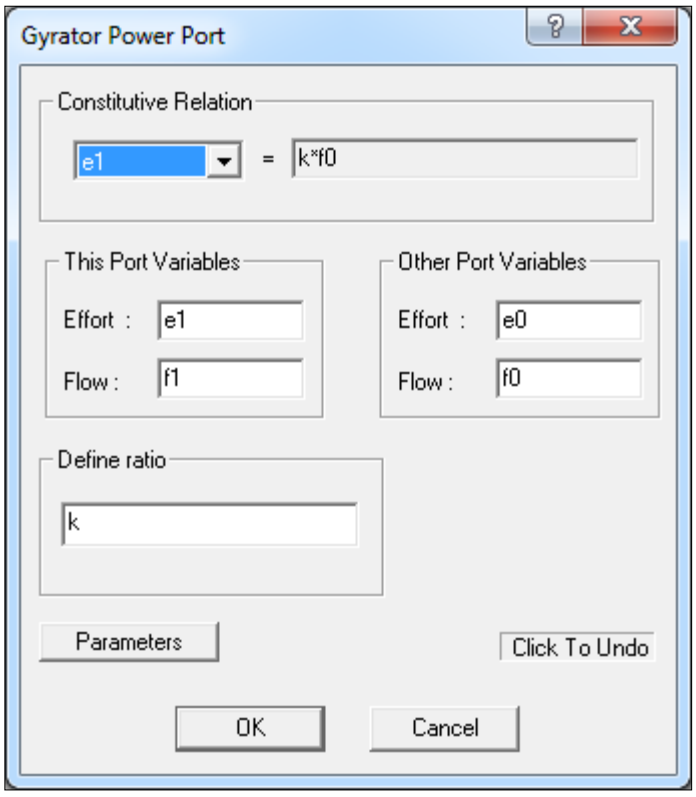
due to conservation of power flow through the component.

Editing the Constitutive Relation

To edit the constitutive relations of Transformer or Gyrator we need to open one of its ports , i.e. either by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The corresponding dialog appears as shown below. We used it to set the constitutive relation seen at this port. The corresponding relation for the other port is automatically satisfied because of power conservation.



TF port dialog



GY port dialog

The constitutive relations for both components are already predefined. Also the names of both ports, the current one and the other, are also predefined as well as the ratio. Note that the constitutive relations for the both components are defined for the effort variable of the current

port with ratio k defined as the parameter and they correspond to the relations given above. However, we may change the type transformation or gyration by using the other port variable, i.e. the flow. This can be achieved by selecting it on the left side of the constitutive relation. The constitutive relations for the component will be also well defined, but the corresponding ratio is simply the reciprocal of the previous one.

The transformation ratio symbol is edited in corresponding *Define ratio* field. It can be defined as the parameter similarly as with other parameters.

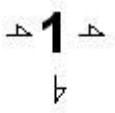
The changes you made can be accepted by clicking **OK**. Otherwise click **Cancel**.

-o-

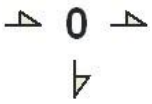
Effort and Flow Junctions

Model:

- **Effort Junction**



- **Flow Junction**



Both junctions are multiport components.

Function:

Effort junction describes branching of efforts with flows being equal at every junction port and thus it is the common junction variable. Similarly, flow junction describes branching of flows where efforts are equal at every junction port, and thus represents the junction variable.

These components are characterized by conservation of power flow through the junction. Counting the power as positive when it flows into the junction and negative when it is directed out of the junction, equation of power conservation yields

$$\pm e_0 f_0 \pm e_1 f_1 ... \pm e_n f_n = 0$$

Now we can formulate the junction's constitutive relations as

- **Effort Junction**

$$f_0 = f_1 = ... = f_n$$

and

$$\pm e_0 \pm e_1 ... \pm e_n = 0$$

- **Flow Junction**

$$e_0 = e_1 = ... = e_n$$

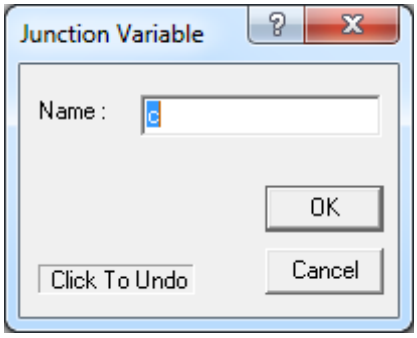
and

$$\pm f_0 \pm f_1 ... \pm f_n = 0$$

Thus, in accordance with power conservation, the constitutive relation for **Effort Junction** represents the balance of the efforts, with the flow being the common junction variable. Similarly, the constitutive relation for **Flow Junction** is the balance of the flows, with the effort as the common junction variable. The sign of the terms in the effort or flow balance equations is taken as plus (+) if at the corresponding junction port power flows into the junction and minus (–) if it flows out of it.

Editing the Junction Variable

The constitutive relation for a junction is defined by its type and number and power sense of its ports. Its junction variable is predefined. However, if we wish we may change it. To edit it we need to open the junction, not any of its ports. To that end we can either double click the junction (on the junction title), or selecting it by clicking and then applying **Open Next** command in **Document** menu. The corresponding Junction Variable dialog appears as shown below.



Junction Variable dialog

We now may type-in the corresponding name and select **OK** button.

-o-

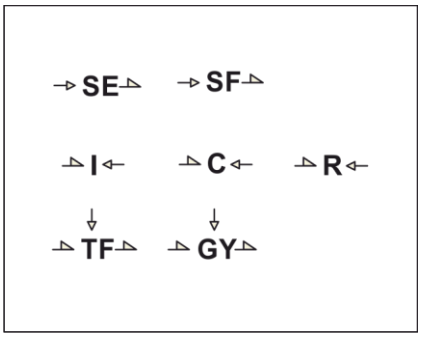
Controlled Components

The components discussed so far contains only power ports. Processes seen at these ports are described by constitutive relations that relate the power variables at component ports and eventually internal state variables. However, there is often the case that processes in a component depend also on some other external variables. Thus hydraulic resistance in a hydraulic valve depends in addition to flow through the valve and pressure difference across the valve (power variables) also on the opening of the valve. Similar situations we have with electric potentiometers, variable capacitors, and many other controlled components in electronics. When dealing with motion of the bodies in space very important role have coordinate transformations of forces and velocities. Such transformations depend on angles of rotations of one coordinate system with respect to other, which are used for description of body motions.

Control-Input Ports

In order to enable modeling of such processes Bond Graphs use so called *modulated* components. These components are typically denoted by adding symbol M (for modulated) to usual symbol of the components, e.g. MSE and MSF denotes modulated sources, and MTF and MGY denotes modulated transforms and gyrators. In BondSim these special symbols are not necessary because every component is treated as a modulated or *controlled* if it in addition to the power ports has also the control-input and control-out ports, i.e. signal ports directed into and out of the component.

Every component with exception of the junctions can have the input signal ports (see Figure below).



Controlled components

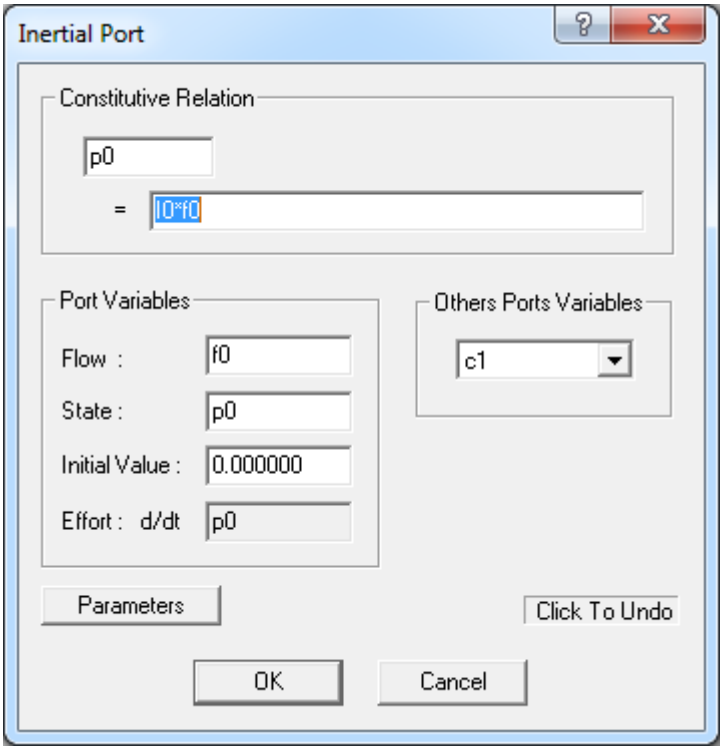
Constitutive relations defined at the power ports can now depend also on the signals at these input ports. Thus e.g. *effort* at the power port of a controlled source effort component may read

$$e = a \cdot \sin \varphi,$$

where a is a parameter and φ is signal at the control input port. In transformers and gyrators signal input ports are often added in order to create variable ratio transformations and gyrations.

Similarly holds for the other components.

The presence of the control input ports change appearance of the port dialogs of different elementary component discussed so far. Thus, e.g. Inertial port dialog now have a modified form as shown below.

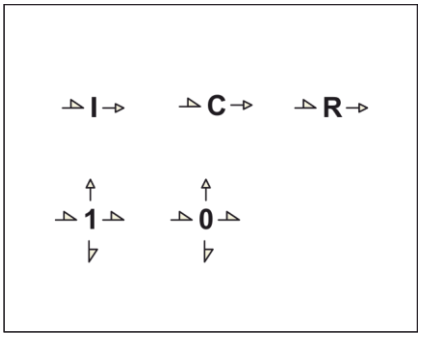


Modified Inertial port dialog

We can see that in addition to data for the current port there is at the right a list of other port variables. There can be found the name of the control input port variable. There could be also the corresponding port variables for the other power ports if any. The similar case is with the other modulated components shown above. These additional variables can be freely used to define the constitutive relation of the ports.

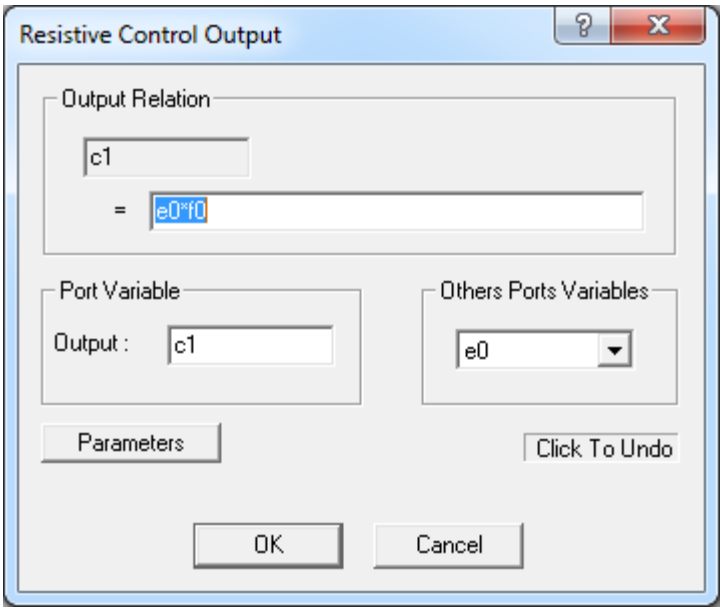
Control-Out Ports

In addition to signal input ports, it is often necessary to extract information from the components. To do that, the control-out ports can be also added to the most components (the exceptions are the sources, transformers and gyrators).



Components with control-out ports

The constitutive relations of control-out ports can be used to define signal generated at the port. Thus e.g. signal at output ports of an inertial or capacitive component can be used to output information on energy stored in the component, or power dissipated in the resistive component. The next figure show the dialog which opens when double clicking the resistor control-out port.



Resistive control-out port dialog

The dialog enables editing of constitutive relation of the control-out port variable in terms of other ports variables given in the port variables list. Similar is the case with the other components. The signals extracted at junctions are different, however, because they are predefined. These are simply the common junction variables, i.e. flows at 1-junctions and efforts at 0-junctions.

-o-

Switch

Model:

Switch component has one power port and one control input port.

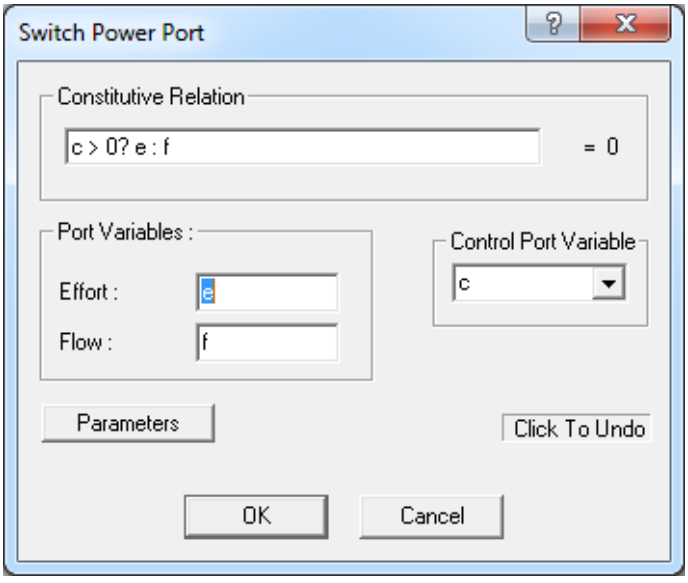
Function:

This is a special discontinuous component which is introduced to simplify modeling of discontinuous processes such are impacts, dry friction and similar processes. The component can be viewed as controlled source that impose condition of zero effort or zero flow depending on sign of input signal. The constitutive relation for the component is

$$\begin{aligned} &if\,(c > 0) \\ &\quad e = 0 \\ &else \\ &\quad f = 0 \end{aligned}$$

Editing the Constitutive Relation

To edit the constitutive relations of the Switch we need to open its power port, e.g. by double clicking the port, or selecting it and then issuing command **Open Next** in **Document** menu (similarly as when opening the component, but applied to the port). The corresponding dialog appears as shown below.



Switch power port dialog

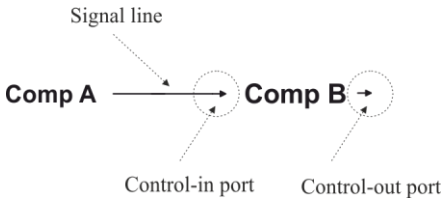
The constitutive relation for the component power port can be modified to allow more complex switching logic then given above.

-o-

Continuous Signal Components

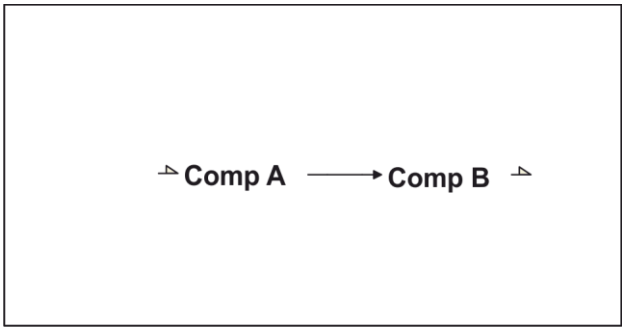
Control Ports and Control Variables

Power transfer between the components often is negligible, but information content is important. In those cases the interaction between the components is represented by signals. Places on the component boundary were such signal are transferred are called control ports and are represented by common arrows. Port with arrow directed into the component is termed *Control-in* port and denotes input of a signal into the component (see figure below). Similarly, a port whose arrow is directed outwards denotes the signal output and is known as *control-out* port. The line connecting a control-out and control-in ports is the signal line or *signal* for short.



Connecting components by signals

To create a component with control-in and control-out ports as shown in the figure above we also make use of **Editing Box**. Suppose that we already have an opened project and wish to create a component with only control-in and –out ports. We will expand **Ports** and **Bond Graph Component** tool branches. Next we drag **Component Model** tool and drop it in the current active document and type in the component title such as **Comp A**. Similarly, we drag again the component model tool and drop it in the same document to the right of the first one and type in the **Comp B** title (see figure below).



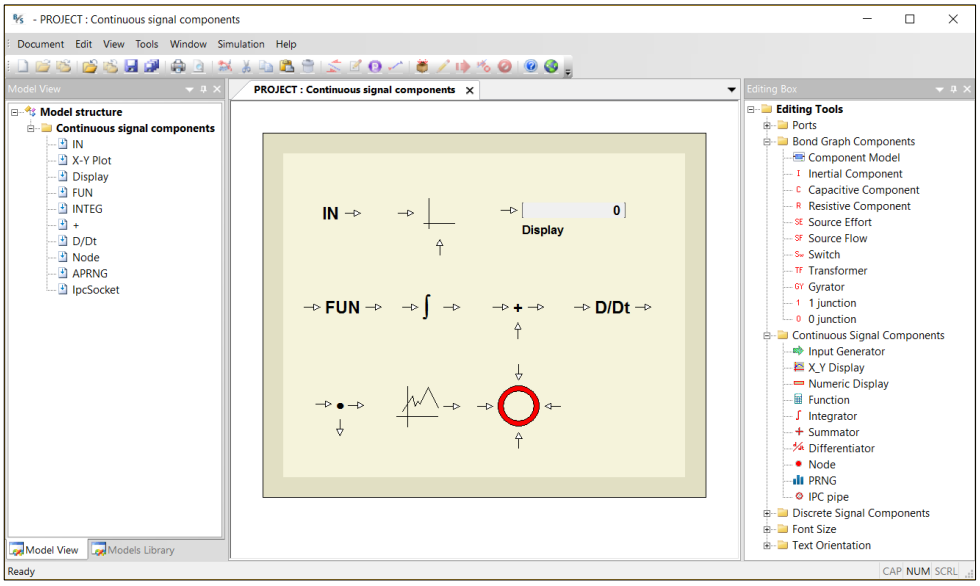
Connecting the components

When components are created they already contain two power ports. We will replace these ports by control ports. To remove the right port of **Comp A** we click it to select and then click **delete** key, or select **Delete** command form **Edit** menu. We drag now the **Control-out** port from **Ports** branch of **Editing Box** and drag it towards **Comp A**. As the cursor is over the component a red component boundary rectangle appears and we move the cursor on the right side near its boundary and drop it there. The control-out port has created and is in the selected state (colored in red). Similarly we do with **Comp B**. We select the left power port and press **delete** key to

remove it. Drag next the **Control-in** tool and drop it on **Comp B** boundary, on the left side facing the control-out port of **Comp A**.

We can now connect these two control ports by a signal. We do it in similar way as we draw the connecting bond. We press on the control-in port of **Comp A** and drag the mouse until is over the control-out port of **Comp B** and drop it. Now the signal is created connecting the ports and is in selected state (see below). We may break the drawing of the signal before dropping it into the port simply by double clicking.

We may continue in the similar way as we did with Bond Graph components. Only difference is in types of the ports. Similarly to Bond Graph elementary components, there are the elementary components that use the control ports only. These are called *block diagram components* because they follow the block diagram technique from Control Engineering. These components can be created using again **Editing Box**. The necessary tools are listed under **Continuous Signal Components** branch. These are the *continuous-time* or *analog* components, and belong to the same time domain as Bond Graph components. Thus they can freely mix. The figure below shows expanded **Continuous Signal Components** branch. To see how these components look like we created a new **Continuous Signal Components** project and in the document that opens in the central region we have created the components by dragging and dropping their tools onto the document window. There are ten components in the total.



Continuous signal components

We create these components basically in the same way as the elementary Bond Graph component. After dragging and dropping the most tools a predefined component text appears and the caret and we may edit it or accept it by clicking outside of it. Some of them such as Integrator, PRNG, and IPC pipe are created right away. We can manipulate these components in the same way as we did with Bond Graph components. These are also the components, but specialized in such a way to support analog block diagram operations.

With every control port of a component we associate a control signal. At control-in port it is an input signal, and the port stores the name of this signal. Similarly at the control-out port there is the output signal. But, there is also the constitutive relation, which defines the output generated by the component in terms of the input signal(s), time t and parameters. To access the ports it is necessary to open them, e.g. by double clicking, or alternatively by selecting port and applying **Open Next** command from **Document** menu, similarly as with the other components.

-o-

Input Generator

Model:

Input Generator is a single output control port component.

Function:

Serves for generating a control signal. The signal generated can be just a constant

$$c = C_0$$

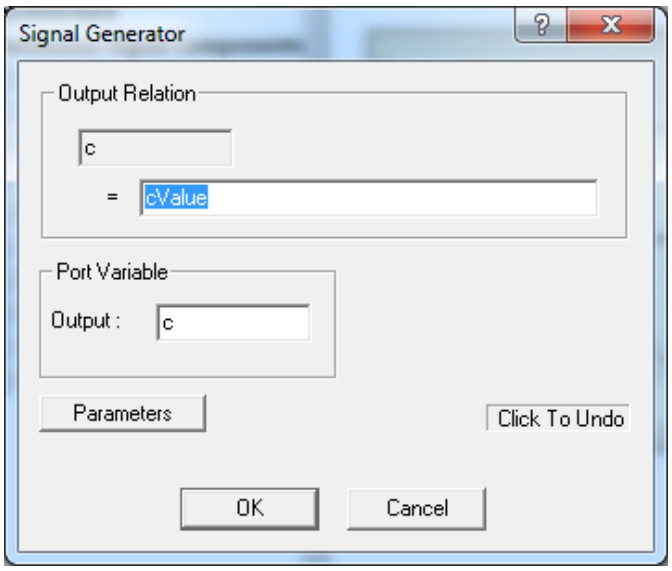
where C_0 is a constant parameter, or a function of time t, e.g. a ramp input

$$c = a + bt$$

with *a* and *b* are suitable constants.

Editing the Constitutive Relation

To edit the constitutive relations we need to open the port, e.g. by double clicking. The corresponding dialog appears as shown below.



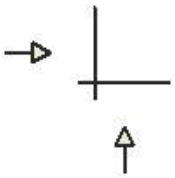
Signal generator port dialog

We may edit the constitutive relation as with the other components.

-o-

X-Y Display

Symbol:



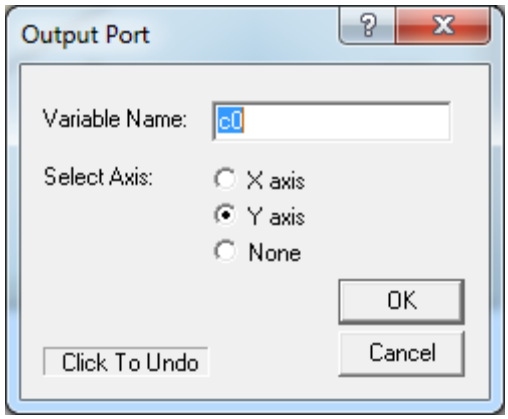
X-Y Display component can have only the input ports and can be used only at the system level.

Function:

Serves to display signals connected to its input ports. Every project must have at least one either *X-Y* or *Numerical display* component. Simulation without generating and recording the outputs does not have much sense.

Set Display Axes

The ports of the display component should be connected to outputs of components which we wish to display. To set the ports and thus the axes along which we wish to display the signal connected to this port we need to open the port, e.g. by double clicking. The corresponding dialog appears as shown below.



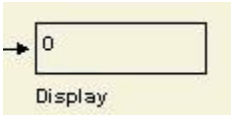
X-Y display port

The dialog gives the name of variable which is plotted along the selected axes. This name would be written near mid of the axis we choose. If we accept default selection the Y-axis for the all ports the generated plot is X-T plot with the signals plotted on Y-axis and X-axis serves for time in s. Of all ports only one can have selected X-axis. All other must select the Y-axis, or none. In that case the generated plots are X-Y plots. Of course if we select the None the variable will not be plotted at all. We can change this setting any time we wish. Note that during the simulation and plotting the outputs BondSim automatically scales the axes to achieve the good looking plots.

-o-

Numerical Display

Symbol:



The numerical display can have only one input port and can be used only at the system level.

Function:

Serves to display numerical value of signal connected to its input port. Every project must have at least one either *X-Y* or *Numerical display* component.

The component is very simple. We usually only change its name (title) to indicate the variable which is displayed. We can edit the title as we did with other components.

-o-

Function

Symbol:

Function component has one or more input ports, but only single output port.

Function:

Serves to generate a signal at its output port as a function of signals connected to its input ports. It can be used to implement different linear or nonlinear functions of its inputs and the parameters. One of simple and often used form of such function is as a single input *gain* block defined by relationship

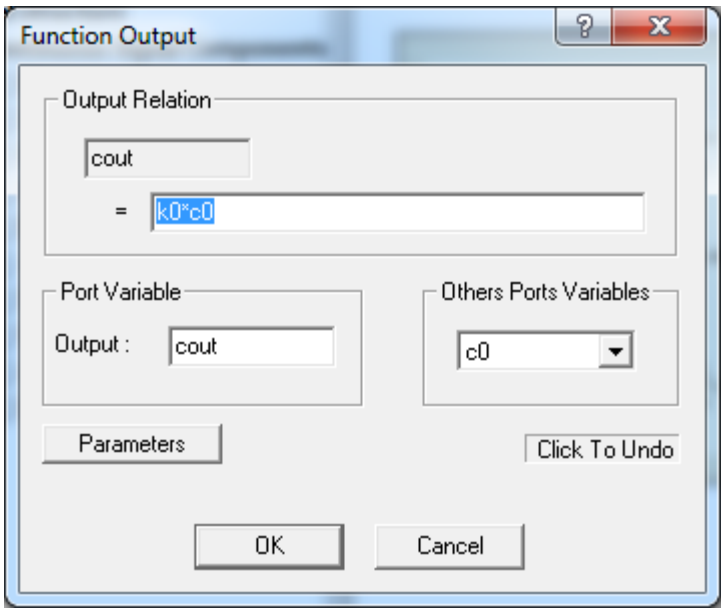
$$c_{out} = k \cdot c_{in}$$

where *k* is a constant.

Note: The *function*, when created already contains one input and one output ports. To insert more input ports it is necessary first to remove the output port. This port is added again after all input ports are inserted.

Editing the Constitutive Relation

To edit the constitutive relations we need to open its control-out port, e.g. by double clicking. The corresponding dialog appears as shown below.

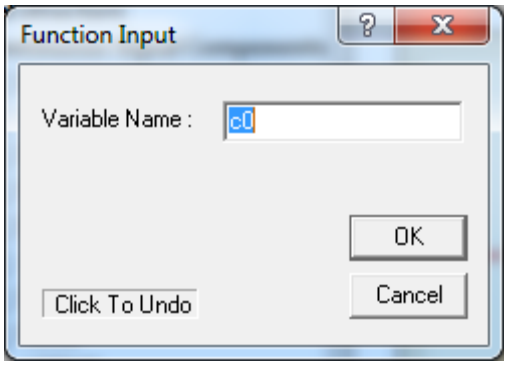


Function output port dialog

Similarly to BondGraph elementary components the dialog gives the name of the output and the

list of the input variables. Thus, to change the output name we edit it in the output edit box. The name at the left side of the output relation at the top of the dialog automatically changes. We edit the constitutive relation in the edit box on the right.

If we wish to change the input names we need to open the corresponding input ports. The corresponding dialog appears as shown in the figure below and we may edit the name. If we have changed the variable name when clicking **OK** button the program asks to change this name in the output relation as well and open this dialog for us. Only after we did it and click *OK* all the changes are accepted.

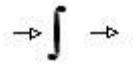


Function input port dialog

-o-

Integrator

Symbol:



It is a single input - single output component.

Function:

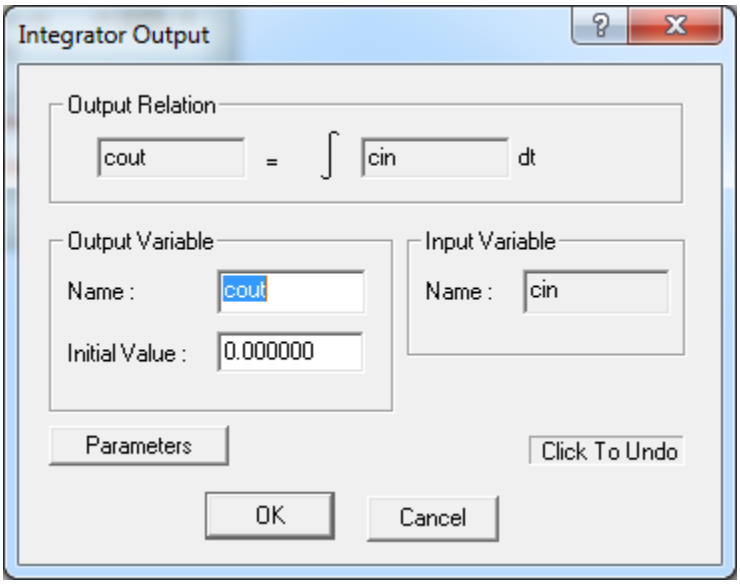
Generates at the output port integral over time of the signal fed to its input port,

$$c_{out}(t) = c_{out}(0) + \int_0^t c_{in}(t)dt$$

The generated output depends also on initial value of the output. This way the output expresses previous history of the input and plays a role of a state variable, i.e. a variable defining the state of the element.

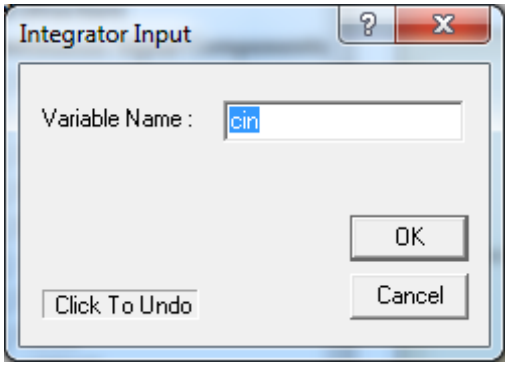
Editing the Constitutive Relation

The constitutive relation for Integrator is predefined. We may change, however, the output name and the initial value of the output. To that end we open the control-out port. The corresponding dialog appears as shown below and may make the changes we wish.



Integrator output port dialog

If we wish to change the integrator input we need to open the corresponding input port. The corresponding dialog appears as shown in the figure below; we may edit the name in the box. The name is automatically set in the constitutive relation.



Integrator input port dialog

-o-

Differentiator

Symbol:

This is a single input - single output component.

Function:

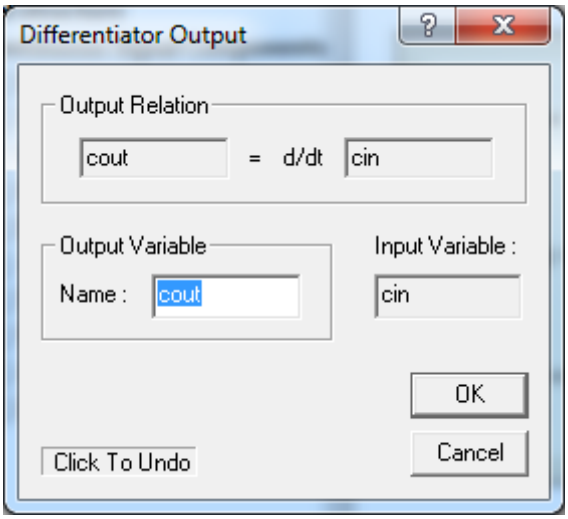
Generates at its output port time derivative of its input,

$$c_{out} = \frac{dc_{in}}{dt}$$

It should be noted that differentiator is not often used because it usually assume the numerical differentiation and this is not nice operation because it amplifies the noise in the system. In BondSim it is not the case. Because the solution of model equations is based on BDF (Backward Differentiation Formula) type solver, differentiation of the signals is implemented using the essential properties of the solver.

Editing the Constitutive Relation

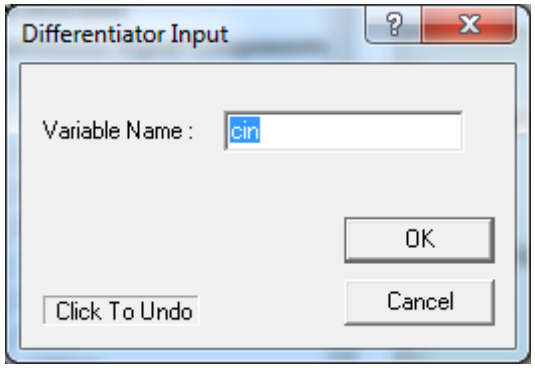
The constitutive relation for Integrator is predefined. We may change, however, the output name. To that end we open the control-out port. The corresponding dialog appears as shown below and may make the changes.



Differentiator output port dialog

The name is automatically accepted in the constitutive relation.

If we wish to change the input we need to open the corresponding input port. The dialog appears as shown in the figure below and we may edit the name.

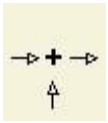


Differentiator input port dialog

-o-

Summator

Symbol:



This is a single output port component which can have one or more input ports.

Function:

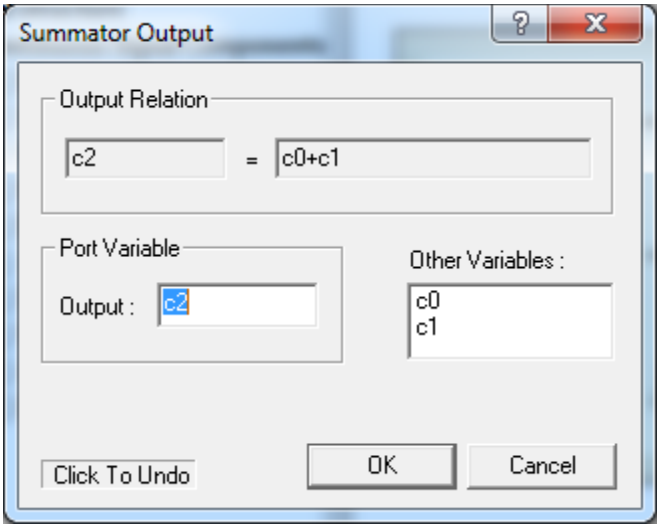
Generates at its output port a sum of its inputs preceded by the optional plus or minus sign,

$$c_{out} = \pm c_1 \pm c_2 ... \pm c_n$$

The sign of the signal is defined at every input port. By default the sign is +. To change it to -, the port should be opened and the input sign set.

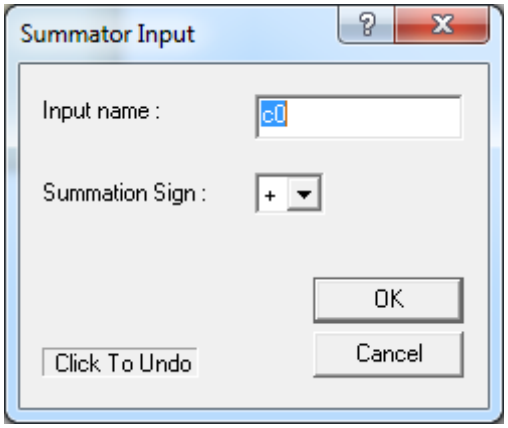
Editing the Constitutive Relation

The constitutive relation for Summator is predefined. We may change, however, the output name. To that end we open the control-out port. The corresponding dialog appears as shown below and may make the changes.



Summator output port dialog

If we wish to change input by name and sign we need to open the corresponding input port. The dialog appears as shown in the figure below and we may edit the name. To change the sign of the input we need to select the appropriated sign form the summator sign list. These two parameters are accepted and applied in the constitutive relation as soon we click **OK** button.

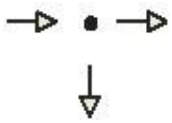


Summator input port dialog

-o-

Node

Symbol:



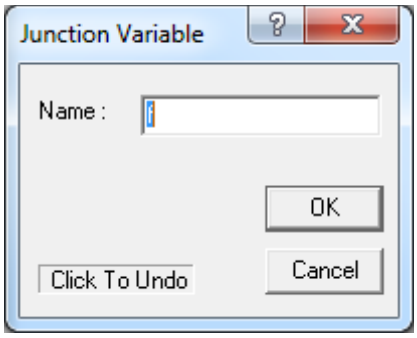
It is a single input port component, which can have one or more outputs.

Function:

Serves for branching of the signals. The signal at the input port is the same at the out-going ports. Thus the signals variable are the same and represent the node (junction) variable.

Editing the Node Variable

To edit the junction variable open the node, e.g. by double clicking to it. A dialog opens as shown below and you may edit the node variable

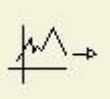


Node variable dialog

-o-

Pseudo Random Number Generator (PRNG)

Symbol:



PRNG is single output control port component that generate pseudo random numbers.

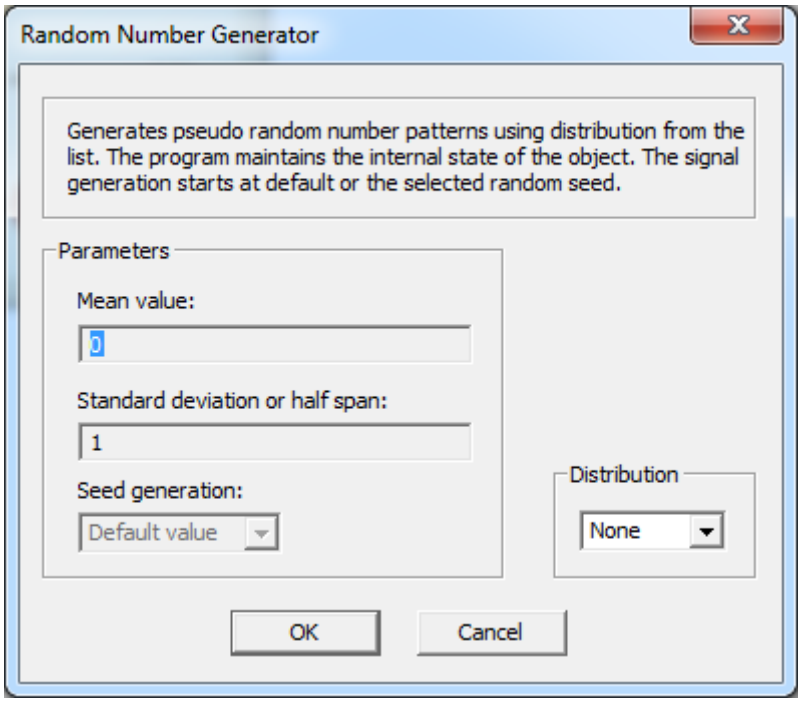
Function:

The components generates a pseudo a random number sequence (PRNG) at every function call. Program maintains internal state of the object. The set of initial values determines the seed of the PRNG. The seed completely determines the generated sequence. Repeating the seed repeats the generated sequence. The component supports two kinds of seeds: predefined (default) and random. The generated sequence is based on the *Mersenne Twister* algorithm, which is currently known as one of the best. The PRNG-generated sequence is not truly random, because it is completely determined by a relatively small set of initial values (seed*)*. PRNG is useful because of its speed in number generation and reproducibility. The component also supports two type of statistical distribution of the generated numbers: uniform and normal (Gaussian).

Using PRNGs in analog simulations can cause problems during the model solving because it introduces a high level of non-smoothness in the model.

Setting PRNG distribution

To set distribution of PRNG signal generated at the port we open the port. A dialog appears as shown below.



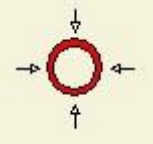
PRNG port dialog

Currently two types of the distributions are supported: the uniform and normal. When distribution is selected we need to define the mean value of the signal, and its half span for the uniform, or standard deviation for the normal. The initial seed can be default or random.

-o-

IPC pipe

Symbol:



IPC pipe is based on the named pipe technology. The currently only single component is supported and it must be created at the project document level.

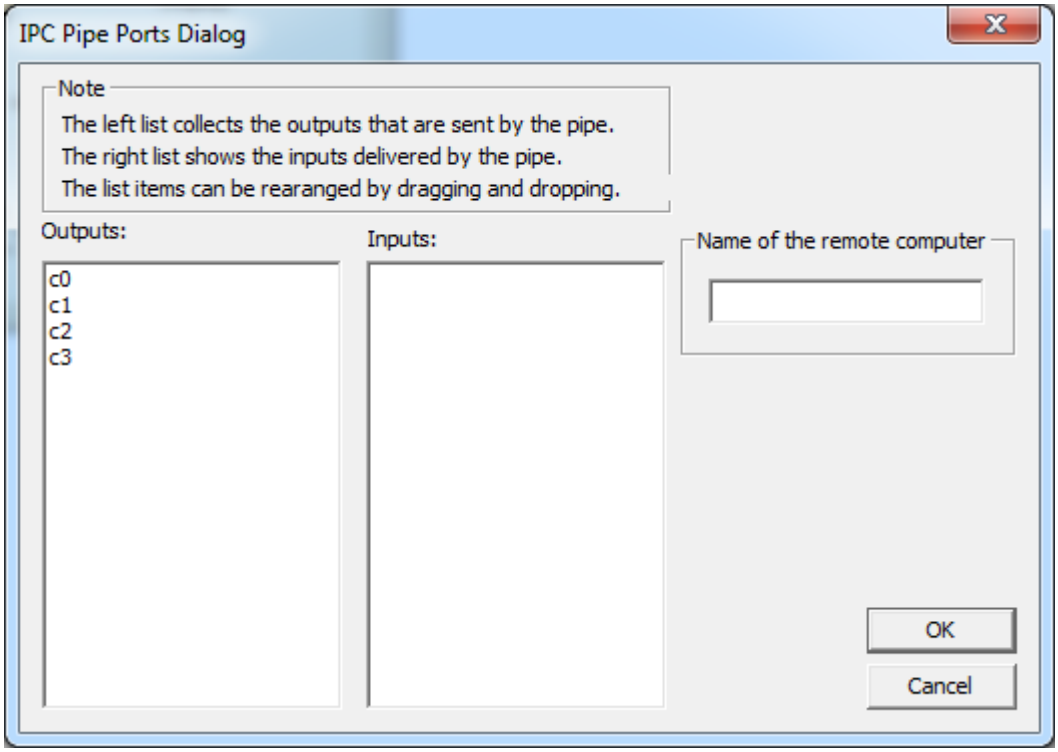
Function:

Enables the inter process communication (IPC) between BondSim dynamic model and any other external application that supports the named pipe IPC, e.g. BondSimVisual. Using IPC two applications running on the same or different computers connected by the local net, can synchronously exchange the data.

Typical application of IPC is for the visualization of the motion of mechanical system such as robots in virtual 3D space. Thus, using BondSim and BondSimVisual the dynamics of a mechanism can be simulated showing both its dynamical behavior in the form of the plots of different variables, and simultaneously generating its motion in a virtual 3D space.

Setting IPC pipe

By opening the IPC pipe component, the **IPC Pipe Ports Dialog** appears as shown below. It enables to rearrange the order of signal connected to the pipe from top to bottom so that signal values sent are packed as the server expects them. In addition we can input the name of the server computer, on which the named pipe object is created. This is the computer to which the client BondSim connects. If it is on the same computer this name is not necessary.



IPC Pipe Dialog

Initially, when IPC pipe component is created it has four pre-created ports for exporting the signals from the model. More control-in ports can be added to the component if more output signals are generated. Similarly, if it is expected to receive the signals from the sever by IPC, the corresponding output signal ports can be added at the component by dragging and dropping them from the **Editing Box**.

-o-

Discrete Bond Graph Components

About Discrete Components

These components are similar to the analog components but work exclusively in discrete time, i.e. time which changes discretely by a constant sampling interval

$$t_n = nT_s$$

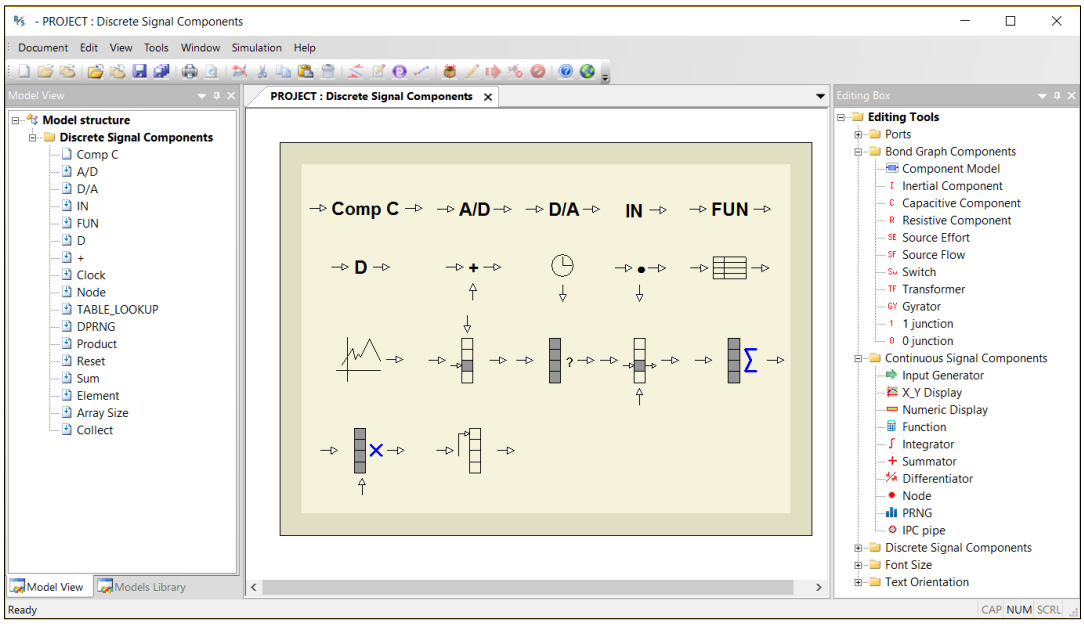
They play the similar roles in the discrete models. Many are similar to corresponding analog components. We have discrete **Component Model**, which have similar role as in analog components but having entirely digital ports and serves to host the discrete components only. A part of discrete components corresponds to **Continuous Signal Components**. This is the case with the **Input**, **Function**, **Summator**, and **Node**. The others are, however, specific to the discrete-time processes, in particular digital ones. Examples of such components are **Analog to Digital Converters** (ADC), **Digital to Analog Converters** (DAC), **clocks** and **Delay** (memory) components, and others.

We can create the discrete components similarly as we did with **Bond Graph** and **Continuous Signal Components** using **Editing Box**. The necessary tools are listed under **Discrete Signal Components** branch. The figure below shows expanded **Discrete Signal Components** branch. In addition **Ports** branch is also expanded. We can see in addition to **Control-in** and **Control-out** tools, also the tools for creating **Activation** and **Triggering** ports.

To see how these all components look like when created we created a new **Discrete Signal Components** project. In the document that opens in the central region we have created the discrete components by dragging and dropping their tools onto the document window. There are seventeen components in the total.

The discrete components are the separate entities and could not be intermixed with analog ones. The only exceptions to this rule are **Analog to Digital Converters (ADC)** and **Digital to Analog Converters (DC)**, which serve to bridge these two worlds.

Note that there are no separate continuous and digital control port tools. When a port is added to the analog component it is constructed as the continuous one. Similarly, when a control port tool is dropped onto a discrete component the port becomes the discrete one. The discrete port cannot be converted into a continuous one because the discrete component can have the discrete ports only. The converse is, however, possible. To convert we select the continuous control port in the continuous component and apply command **Change to Discrete** from **Edit** menu.



Discrete components

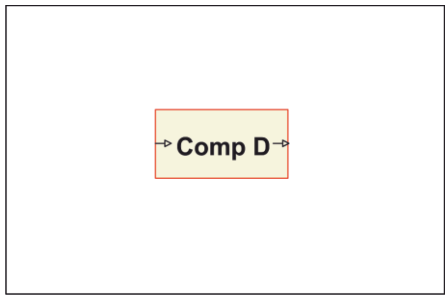
-o-

Creating Discrete Components

To create a discrete component

- Activate **Editing Box** on the right side of the program frame
- Expand **Ports** branch to have access to the **Control-in**, **Control-out**, **Activation** and **Triggering** ports
- Expand **Discrete Signal Components**.

The procedure is practically same as with Bond Graphs and continuous signal components, but we use the tools from **Discrete Signal Component** branch. Thus, to create a **Component Model** we drag the corresponding tool from the discrete branch and drop it on the current active document. Edit then the component title and click outside of it (see figure below). It already has two ports, one control-in and other the control-out. However, these ports are discrete, and thus its model document can contain the discrete components only.



Discrete Component Model

The component model represents the starting point for development of discrete-time models. To develop its model it should be opened and the corresponding document edited, similarly as did with Bond Graph model components. This time, however, use is made of the tools from the discrete branch of **Editing Box**.

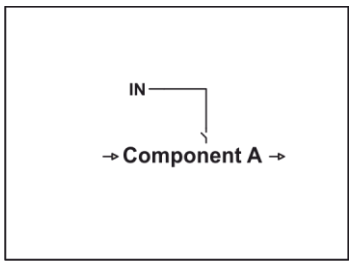
We can add other ports if necessary. When the control ports tools are dropped onto a discrete component the discrete ports are created.

Besides the **Component Model** tool there are other tools for creating different specific discrete components. Many are just discrete version of the continuous components, e.g. **Input Generator**, **Function**, **Summator**, **Node, PRNG**. The others are inherently discrete ones. Thus **A/D** and **D/A** components are converters between the continuous (analog) and discrete (digital) space. The **Clock** defines the sampling frequency of the discrete part, and **Unit Delay** is a discrete analog of the continuous **Integrator**. There are other more specific discrete functions. They are created by dragging and dropping the corresponding tool in the currently active document window. A suitable component name is shown, which we can retain. Editing is finished by clicking outside as with other components.

-o-

Triggered Component

The triggered component is a component to which a triggering port is added (see figure below). This port is visible by all components contained in the discrete component model document, and in all contained components up to all levels. The component is controlled by a input signal fed to the triggering port. When the triggering condition is satisfied, the processes in the all contained components are executed and for the component is said that it has been triggered. When the compones are not triggered their outputs retain the previous state, i.e nothing happend.

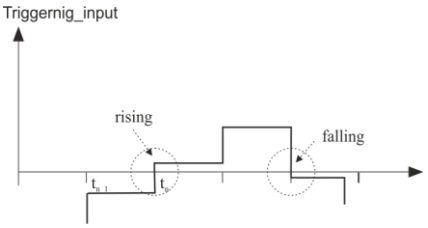


Triggered component

The triggering condition are determined by comparing values of the current value of the triggering input signal and that at the previous time instant, which is denoted by appending "_1". There are different possibilities as shown in the table below:

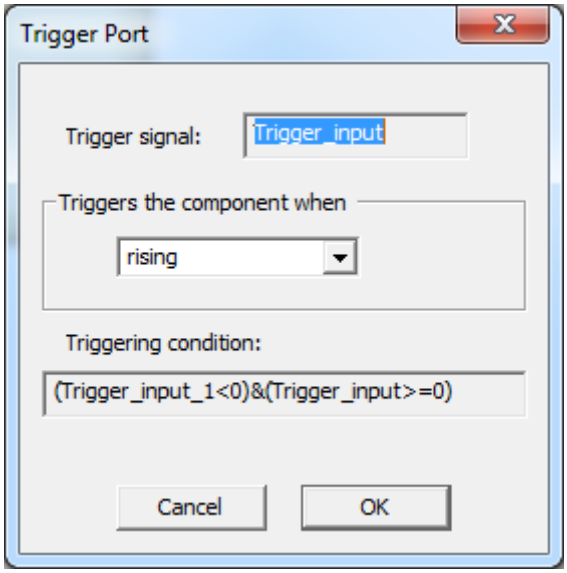
Kind of triggering	Triggering conditions
rising	(Trigger_input_1<0)&(Trigger_input>=0)
falling	(Trigger_input_1>0)&(Trigger_input<=0)
either	((Trigger_input_1<0)&(Trigger_input>=0))|((Trigger_input_1>0)&(Trigger_input<=0))
specific	To be defined

The *rising* triggering corresponds to the case when the triggering input was negative in the previous time instant, but becomes greater or equal to zero in the current time (see the figure below). The *falling* is just the opposite condition, i.e. the input signal is positive in the previous time instant, but becomes less or equal to zero in the current time. Thus, in both cases the triggering means crossing zero by the trigger input signal, which either rises up or falls down. The third kind denoted as the *either* includes the both possibilities. Finally, it is possible to define a specific triggering condition by a logical expression.



Triggering conditions

To define the triggering condition we open the triggering port (e.g. by double clicking) and when the *Trigger Port* dialog opens as shown below select or specify the condition.

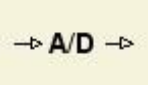


Triggering port dialog

-o-

A/D converter

Symbol:



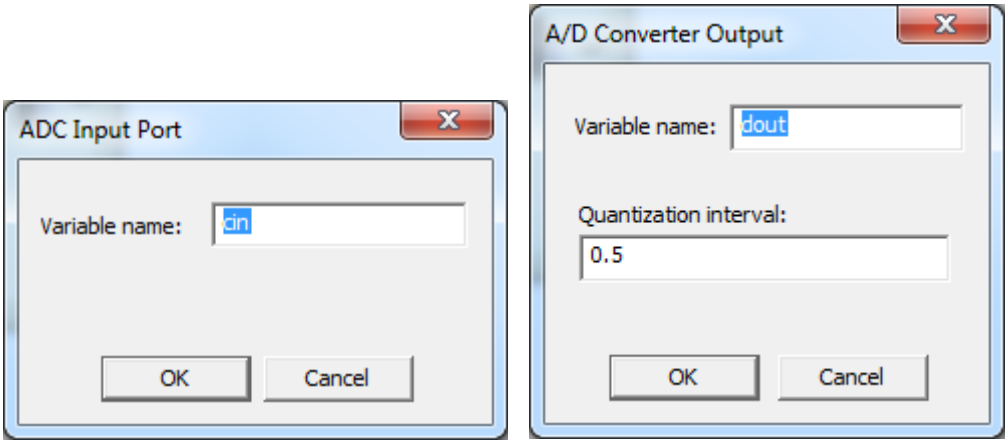
A/D converter has only one control-in and one control-out port.

Function:

Converts the analogue input signal to digital output signal. This is a hybrid component whose input port (A port) has to be connected to the output port of the analog component, and its output port (D port) to the input to a discrete signal component. It could be created either within a continuous-time component or in the root project document.

Setting the Converter

To set a converter we may edit the name of its input signal by opening the input (analog) port. The dialog opens as shown in the figure below on the left. It is similar to other dialogs for defining the input names.



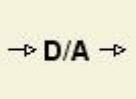
ADC ports dialogs

However, more important is the output (discrete) port. Figure above on the right shows the converter output port. In addition to the name of the output variable we can edit the quantization level. If there is no noise in the input signal the default value can suffice. But, in real equipment there are usually noises present so that such low resolution cannot be achieved and much higher quantization value is necessary. For more details see here.

-o-

D/A converter

Symbol:



D/A converter have only one control-in and one control-out port.

Function:

Converts the discrete input signal to the analogue output signal. This is a hybrid component whose input port (D port) has to be connected to the output port of a discrete-time component and the output port (A port) to the input port of a continuous time component. There are different ways how the conversion can be achieved. The simplest and often used method is based on holding the discrete value at the constant value over the sampling interval. Because the interpolation over the interval is by zero-order polynomial such a method is known as zero-order hold. It is also used in BondSim.

Setting the Converter

The converter does not need a particular setting, but change the names of input (digital) and output (analog) signal. The corresponding dialogs are similar to other such dialogs, see e.g. A/D input dialog.

-o-

Input Generator

Symbol:

Input Generator component is a single output control port component.

Function:

Serves to generate a *discrete* control signal. It is a special case of continuous signal Input component and hence the discussion given there applies here as well.

-o-

Discrete Function

Symbol:

Discrete Function component has one or more discrete inputs and one discrete output ports.

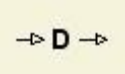
Function:

Serves to generate a discrete signal at its output port as function of discrete signals connected to its input ports. It is a special case of continuous signal Fun component and hence the discussion given there applies here as well.

-o-

Unit Delay

Symbol:

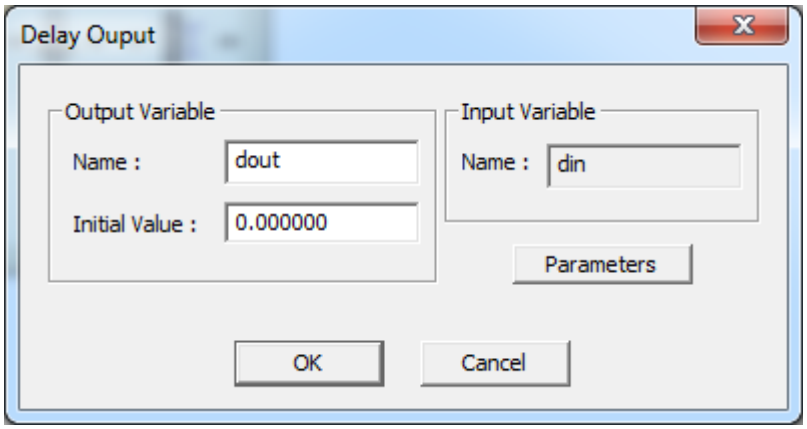


Function:

This is a discrete counterpart of analog Integrator. It serves to store values in the memory and thus is often known as memory element. Because the value stored at the current time step is available in the next step, its output is equal to value of its input at the previous discrete time. Hence, it also introduces delay for one sampling interval.

Editing of Unite Delay

To define the function we open its output port, e.g. by double clicking it. In the dialog windows that opens and which is shown below we may define the name of the output and its initial value. By default it is preset to zero, but can be any value. To define the input name we need to open the input port. A dialog such as in integrator input ports appears.



Delay output port dialog

-o-

Summator of Discrete Signals

Symbol:

This is a single output discrete port component which can have one or more input discrete ports.

Function:

Generates at its output port a sum of its inputs preceded by the optional plus or minus sign. It is a special case of continuous signal Summator component and hence the discussion given there applies here as well.

-o-

Clock

Symbol:

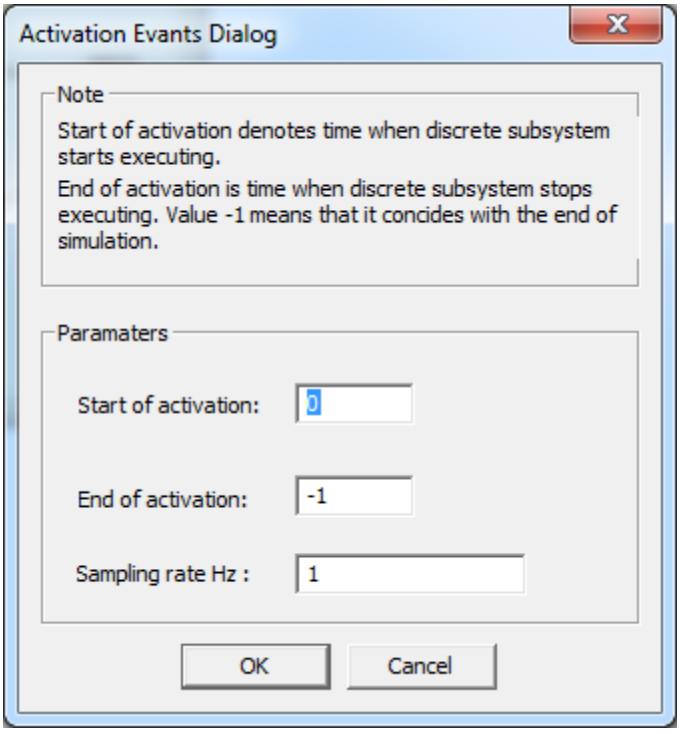
Clock has already attached a control-out port which can't be removed. The clock activates the discrete processes at time intervals equal to one or more intervals at which the simulation generates the outputs. There is only one clock per project.

Function:

The model of discrete system has to have an activation clock. It is connected via activation port, which can be only attached to the discrete signal branching node. It defines starting and ending of discrete process and the sampling frequency. Usually it defines the sample rate of to the ADC component. However, there is only one clock allowed per the project.

Defining Clock

By opening the clock's output port the Activation Events Dialog box appears which enables defining the start and end of the activation period and the sampling frequency in Hz (see below).

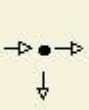


Clock activation dialog

-o-

Discrete Signals Node

Symbol:



It is a single input discrete port component, which can have one or more discrete outputs.

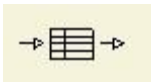
Function:

Serves for branching of the discrete signals. In addition to single digital input port, an *activation* port can be added too. This last one looks as an ordinary input port, but can be connected only to the **Clock**. It is a special case of continuous signal **Node** component and hence the discussion given there applies here as well.

-o-

Table Lookup Function

Symbol:



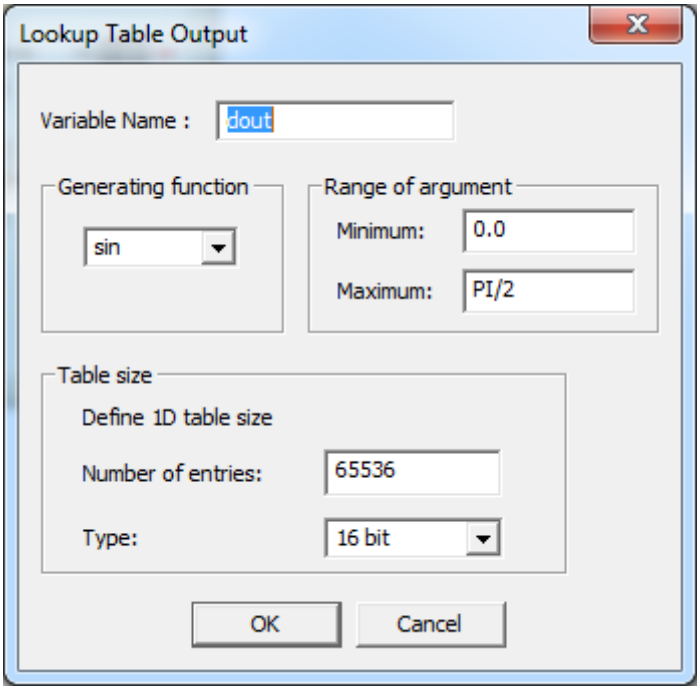
Component has one input and output port.

Function:

This function is based on a table look up for efficient generation of the values.

Defining Table Lookup function

To define the output variable name, type of function, range of argument and the table size, open the output port, e.g. by double clicking. The Lookup Table output dialog box appears as shown below. The function currently supports sine and cosine functions. The rage of input arguments is 0 to $\pi/2$ and 16 bit table size.

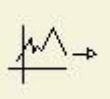


Lookup table output dialog

-o-

Pseudo Random Number Generator (PRNG)

Symbol:



PRNG is single output control port component that generate discrete pseudo random numbers.

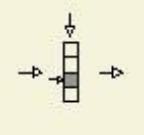
Function:

Generates at any discrete time a pseudo random number (PRNG) using the distribution from the list. This is a special case of continuous signal PRNG component and thus description given there applies here as well.

-o-

Generate Array

Symbol:



Component has three ports - one input data port, the reset port and output port.

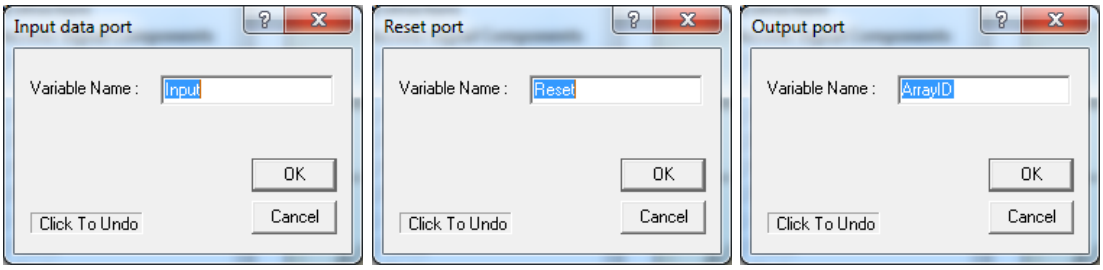
Function:

The component generates an array by collecting data received at the input. The size of the array grows up as the discrete values are received. At the output it gives ID of the array which can be used to access the array. At the construction it is empty with predefined ID value. When the reset signal is applied to the reset port the index of the array is reset to zero.

Editing ports

We can edit the names of the port variables by opening the ports. The dialogs that correspond to Input data, Reset and Output ports are shown below. By opening these ports we may find out the purpose of the each of them.

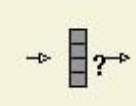
The port dialogs



-o-

Array Size

Symbol:



Component has one input and output port.

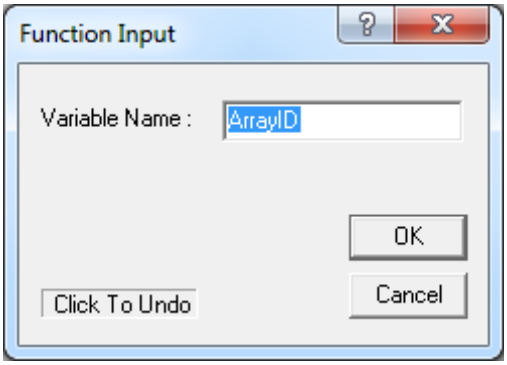
Function:

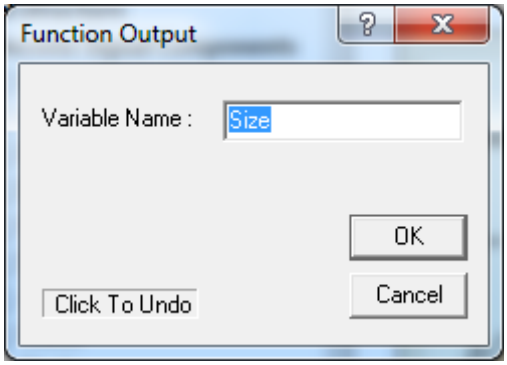
The output signal of the component gives the size of the array whose ID is set at the input port.

Editing ports

We can edit the names of the port variables by opening the ports. The dialogs that correspond to Input and Output ports are shown below. By opening these dialogs we see clearly the purpose of the each of them.

The port dialogs

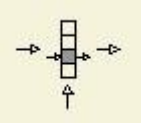




-o-

Array Element

Symbol:



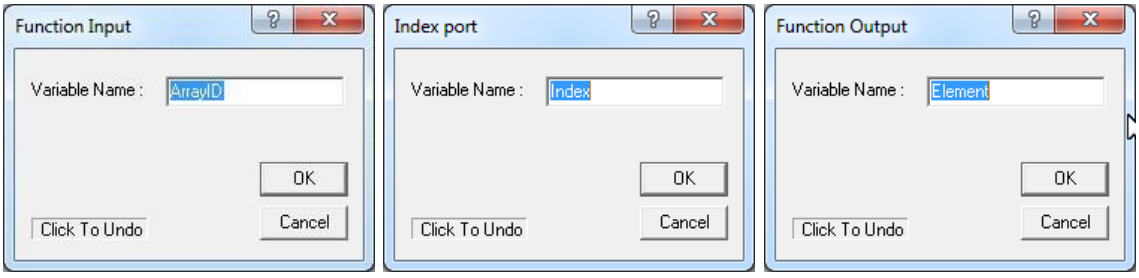
Component has two input ports and the output port.

Function:

The component gives at its output port the value of the element of the array whose ID is set to the input port and the value of the corresponding index set at the index port.

Editing ports

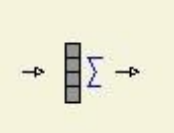
We can edit the names of the port variables by opening the ports. The dialogs that correspond to Input, Index and Output ports are shown below. These dialogs show clearly the purpose of the each of these ports.

The port dialogs 

-o-

Sum of Array Elements

Symbol:



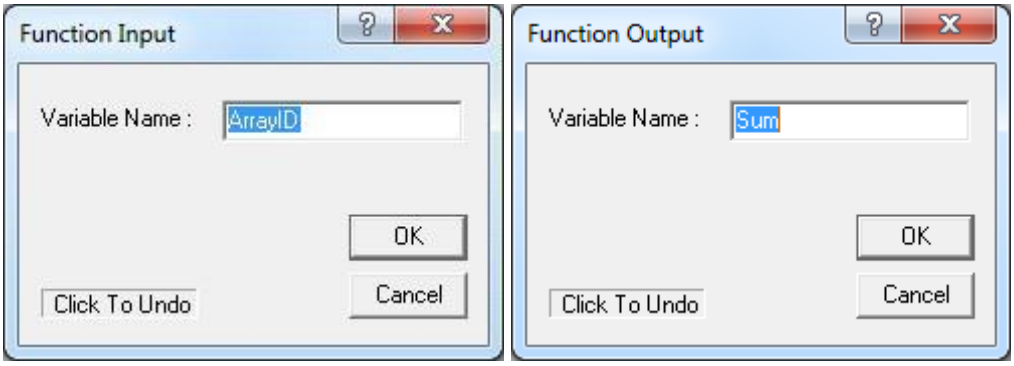
Component has one input and output ports.

Function:

The component at output port generates the sum of the elements of the array whose ID is set at the input port.

Editing ports

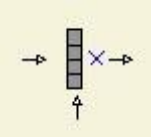
We can edit the names of the port variables by opening the ports. The dialogs that correspond to Input and Output ports are shown below. These dialogs show clearly the purpose of the each of these ports.

The port dialogs 

-o-

Product Array

Symbol:



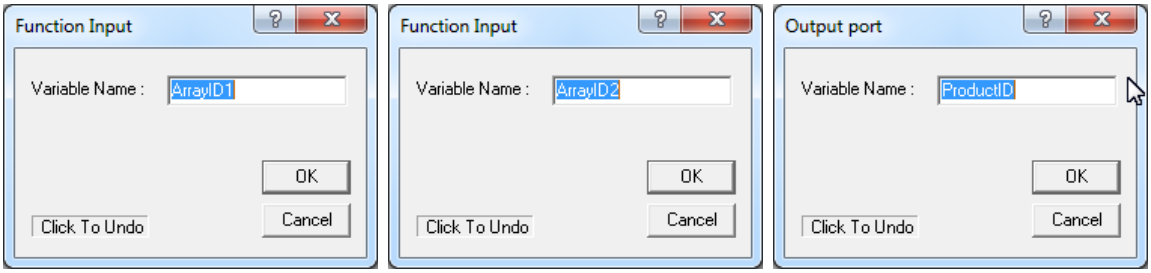
Component has two input ports for two input arrays and one output port.

Function:

The component generates the ID of an array whose elements are the product of the corresponding elements of the arrays whose IDs are set at the input ports.

Editing ports

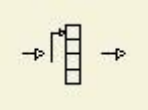
We can edit names of the port variables by opening the ports. The dialogs that correspond to Input 1, Input 2 and Output ports are shown below. These dialogs also show clearly the purpose of the each of these ports.

The port dialogs 

-o-

Reset Generator

Symbol:



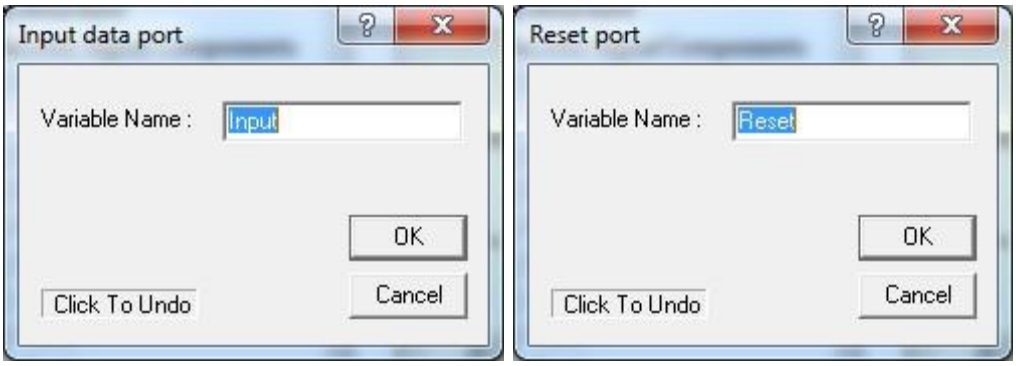
Component has one input and output port (for the signal reset).

Function:

The component is used for reset other components, e.g. generated array, but also the other not related to the array operations. If the component is used within a triggered component it generates "on" (1) signal when the component is triggered, and "off" (0) otherwise. On other hand, if it is not used within a triggered component it unconditionally generate "on" reset signal. The input signal serves to set proper order of the component evaluation.

Editing ports

We can edit names of the port variables by opening the ports. The dialogs that correspond to Input and Reset ports are shown below. These dialogs also show clearly the purpose of these ports.

The port dialogs 

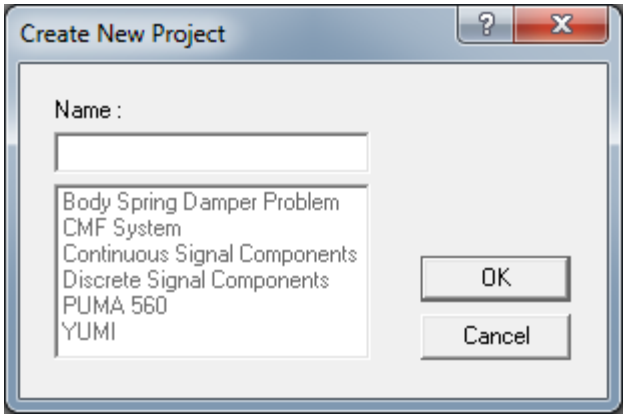
-o-

Working with Projects

Creating New Project

- **To create a new project**

1. From the **Project** menu, click **New** command.
   The **Create New Project** dialog box appears. It lists the names of the existing projects.



Create new project dialog

2. Type a project name you wish to create and then click **OK**. The name of the new project has to be unique.
3. The new project document will be opened in a window in the central part of the program frame

- **Shortcuts:**

Toolbar: 

Keys: **Ctrl + N**

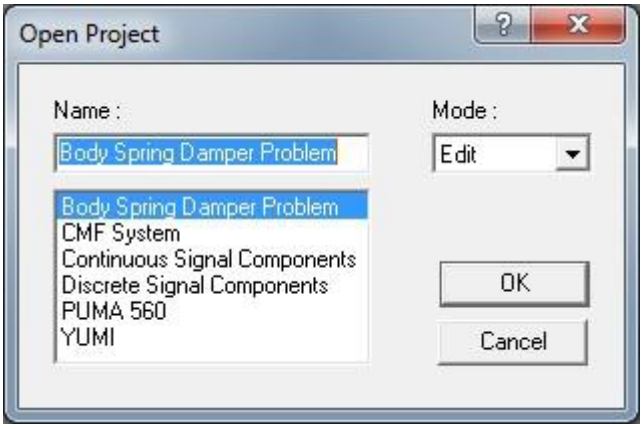
-o-

Opening Existing Project

To open an existing project

1. From the **Project** menu, choose **Open** command.

   The **Open Project** dialog box appears with list of the existing projects.



Open project dialog

2. Double-click a project name in the list. You can go across the name list with the scrollbar. Also, you can select project in the list or type the project name in the text editing box and click **OK** button.
3. You can set the **Edit** mode if you wish to allow changes to the project or **Read only** mode if you wish to review the project only. **Edit** mode is selected by default.
4. The document project will be read from the workspace and displayed in the central part of the program frame.

Shortcuts:

Toolbar: 

Keys: **Ctrl + O**

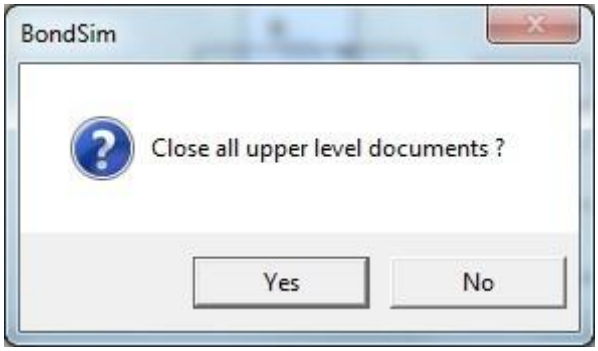
Note: Command **Open** is available if there are the stored projects in the workspace.

-o-

Closing Project

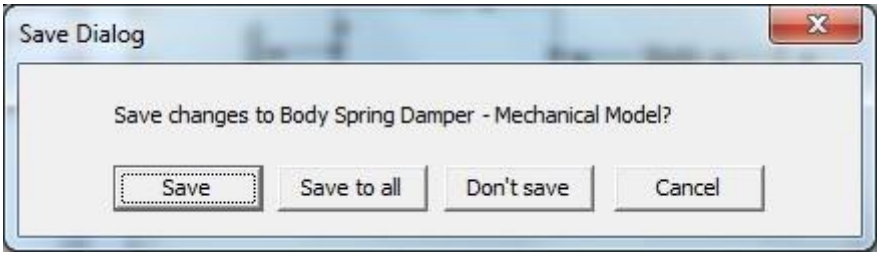
- **To close an opened project**

1. From the **Document** menu choose **Close Project** command. If at the root system level document it is also possible to us **Close** command. (It is normally used for closing a component, but can also be used to close the root project component.). You can also click the ☒ button in the project document window, or collapse the project branch in the **Model View** tree at the left.
2. If there are opened components the system ask to close them by the following message:



Close upper window message

3. If some of the opened components are modified the program asks what to do next by issuing the following message:



Save document message box

There are several options

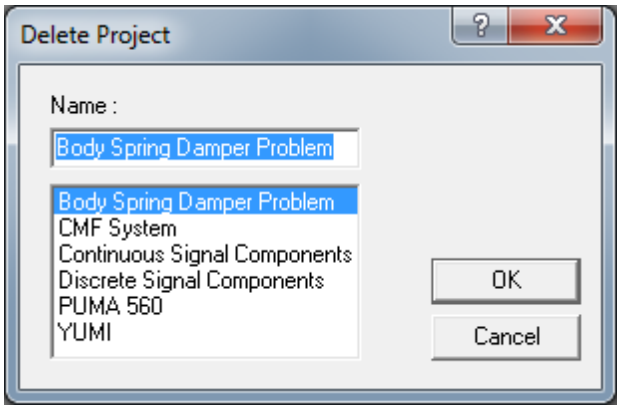
- Save — To store the document into the corresponding component
- Save to all — To store all documents in their components down to the level of the project
- Don’t save — To reject the current document changes
- Cancel — To cancel the operation

-o-

Deleting Project

- **To delete an existing project from workspace**

1. From the **Project** menu, choose **Delete** command
   The **Delete Project** dialog box appears.



Delete project dialog

2. Click the project name or type the project name in the *name text editing box* you wish to delete and then click **OK** button, or double-click the project name in the project list. You can go across the project list by the scrollbars if the all projects are not visible. The project is not deleted right away but moved into **Waste Bin** from which it can be finally removed from the system or recovered back.

- **Shortcuts:**
  Keys: **Ctrl + D**

Note: Command **Delete** is available if there are stored projects in the workspace.

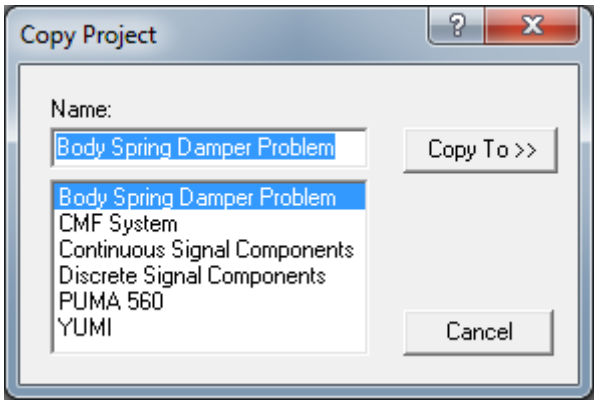
-o-

Copying Project

To create a copy of existing project in workspace under a new name

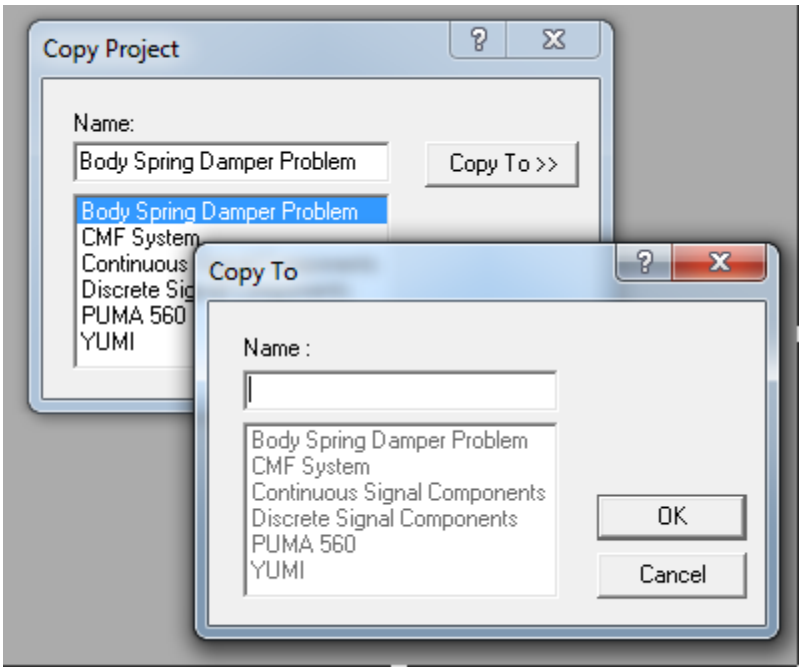
1. From the **Project** menu, choose **Copy** command.

   The **Copy Project** dialog box appears with list of existing projects.



Copy project dialog

2. Choose the project name from the offered project list or type the project name in the *name text box* you wish to copy and then click **OK** button or double-click the project name from the offered project list. You can go across the name list with scrollbar. The **Copy To** dialog box appears.



Copy to dialog

3. Type the name of a project copy and click **OK**.

- **Shortcuts:**

  Keys: **Ctrl + Shift + C**

Note: Command **Copy** is available if there are stored projects in the workspace.

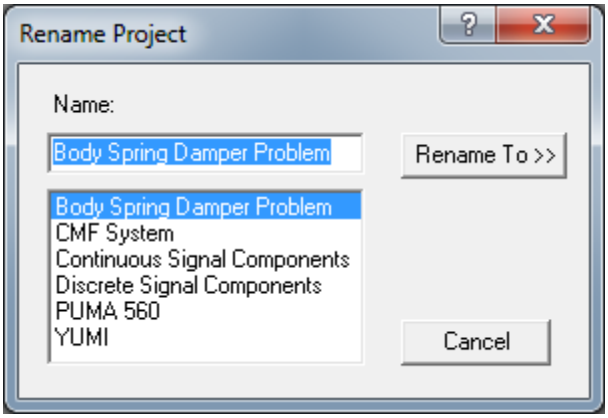
-o-

Renaming Project

To change of existing project's name

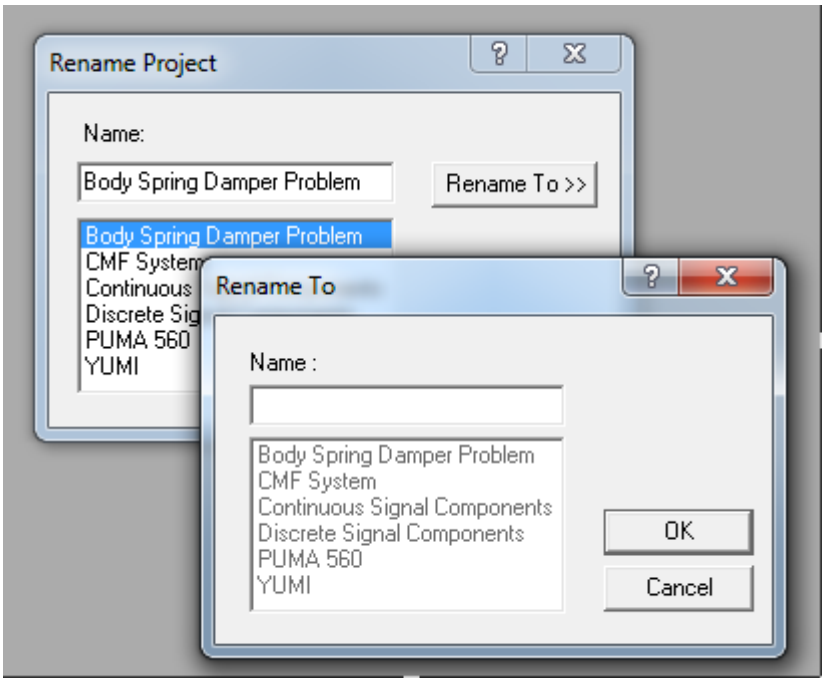
1. From the **Project** menu, choose **Rename** command.

   The **Rename Project** dialog box appears with list of existing projects.



Rename project dialog

2. Click the project name from the offered project list or type the project name in the *name text box* you wish to rename and then click **OK** button or double-click the project name from the offered project list. You can go across the name list with scrollbar. Click **Rename To** button. The **Rename To** dialog box appears.



Rename to dialog

3. Type the new project name and click **OK**.

- **Shortcuts:**

  Keys: **Ctrl + Shift + R**

Note: Command **Rename** available if there are stored projects in the workspace.

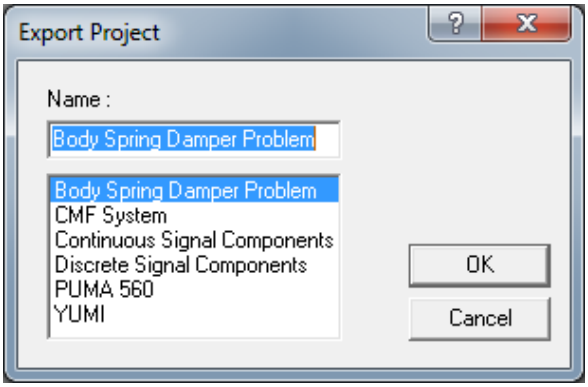
-o-

Exporting Project

To export a project

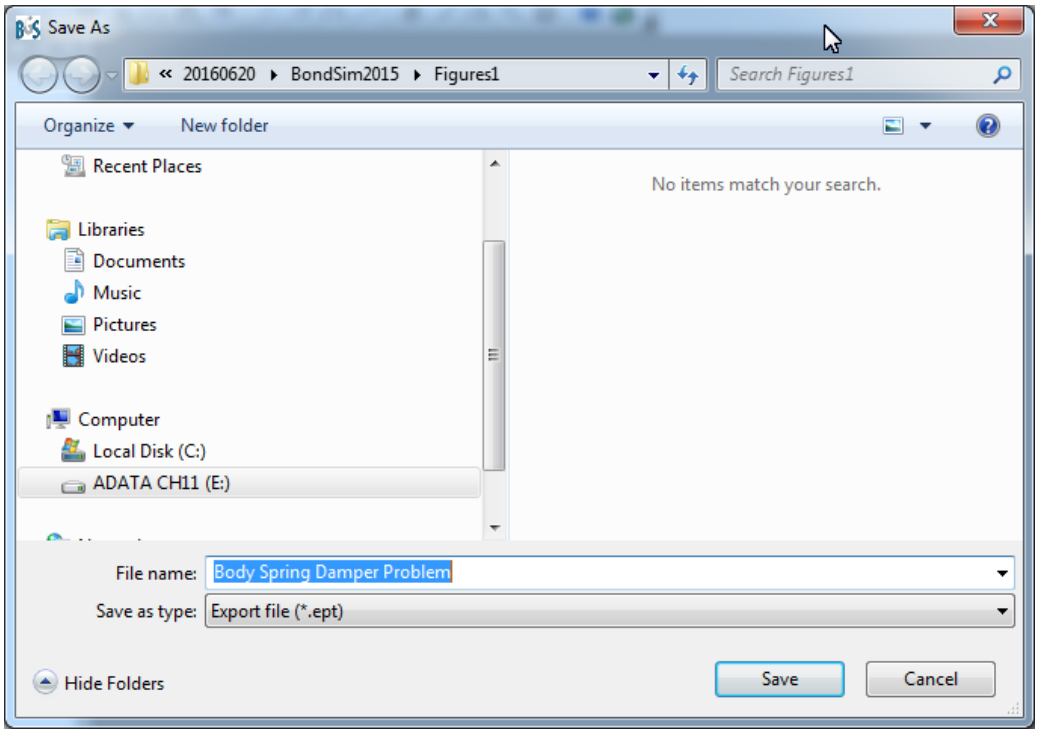
1. From the **Project** menu, choose **Export** command.

   The **Export Project** dialog box appears.



Export project dialog

2. Click the project name in the project list or type the **project name** in the *name text box* you wish to export and then click **OK** button or double-click the project name. The **Save As** dialog box appears and you may select the destination and the name of the exported file.



Export Save as dialog

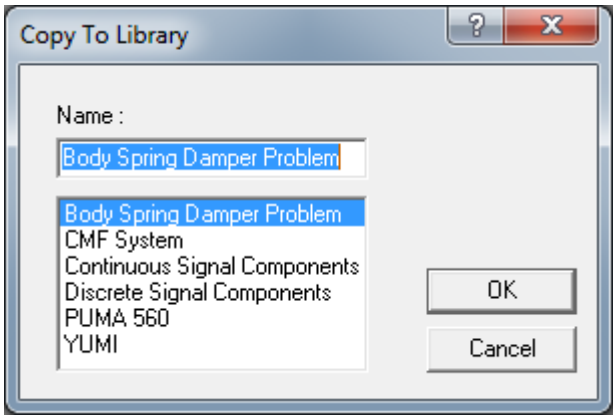
3. By clicking on **Save** the export version of the project (*.ept) will be created.

-o-

Copying Project into Library

- **To move a copy of an existing project from the workspace into Library**

1. From the **Project** menu, choose **Copy to Library.** The **Copy To Library** dialog box appears with name list of existing projects in the workspace.



Copy project to library dialog

2. Select project you wish to put into the Library by clicking *project name* in the offered list box or type *project name* in the *name editing text box and* click **OK** button (or double click project *name* in the list box). If a project under the same name already exists in the library the program will ask you to approve adding the project under the same name.

- **Shortcuts:**

  Keys: **Ctrl + L**

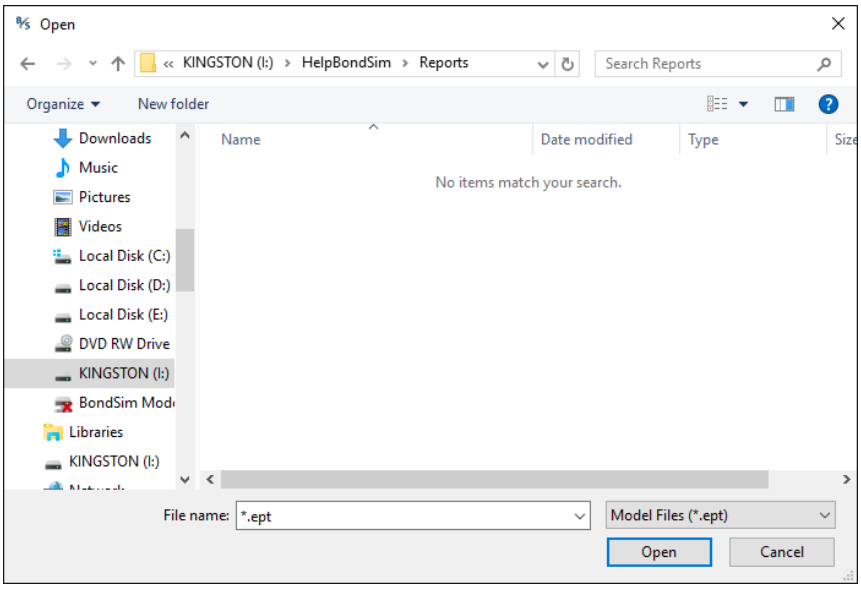
Note: Command **Copy To Library** is available if there are projects stored in the workspace

-o-

Importing Project

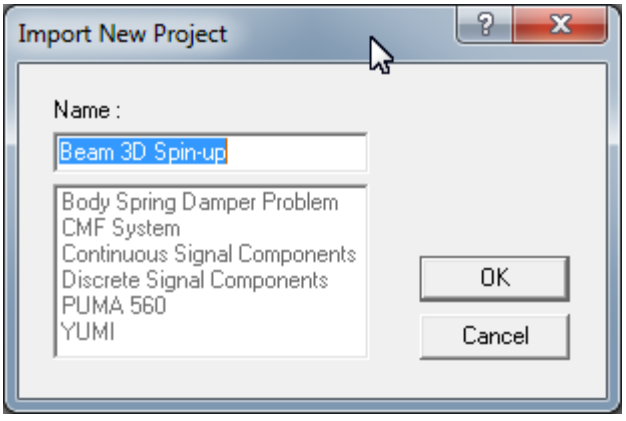
To import a project into the workspace from an external medium

1. From the **Project** menu select **Import...** command. The **Open** dialog box appears.



Import project open dialog

2. Select model file to import. It could be project (.ept), symbolic (.fun) or tabular (.tab) functions.
3. Browse the source folder for the import file and select the file name. Click then **Open** (or double-click on the import file).
4. An **Import New Project** dialog opens to define the new name under which the project will be imported.



Import new project dialog

Note: An import operation may need some time to finish.

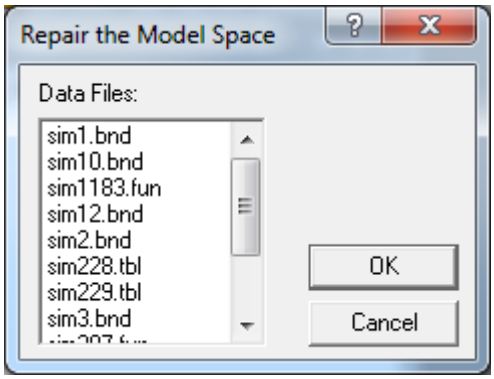
-o-

Repairing Projects

To repair the model database in the case of corruption

1. On the **Project** menu, click **Repair Projects** command

   The **Repair the Project Space** dialog box appears.

Repair model space dialog 

   Dialog contain list of the files held in the project data base. Some file could be missing. One of the purpose of the repair is to rebuild the data base.

2. Click **OK** button to start repairing of the model workspace.
3. The messages in the **Repair** tab of the **Output** window at the bottom of the program frame show the progress of the repairing process. The proper projects and components found are moved to Waste Bin from which after the process completes could be returned into the model workspace or removed from the system. If there were the corrupted files found the user is asked to remove them.

-o-

Waste Bin Operations

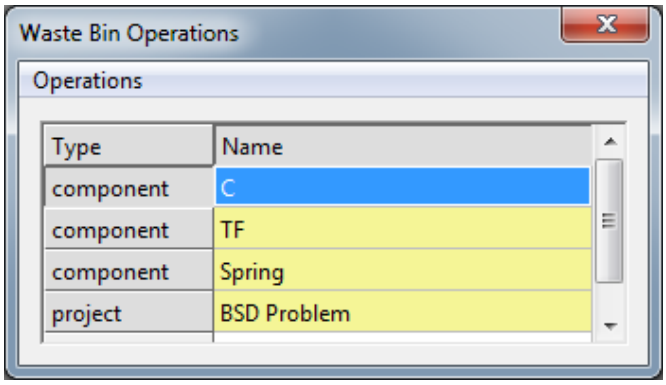
- The **Waste Bin** is a structure maintained by BondSim to hold different objects (projects and component models) collected during remove and repair operations. To access it we select **Waste Bin** command in **Project** menu, or in **Document** menu.

- **Shortcuts:**

  Toolbar: 

  Keys: **Ctrl + W**

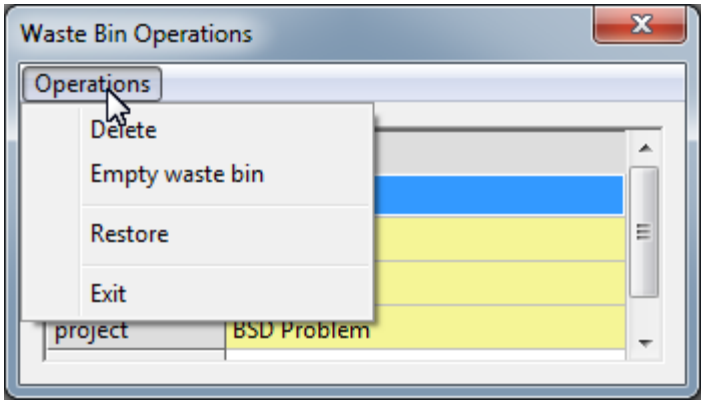
  The following dialog opens.



Waste Bin dialog

The dialog lists the components and projects held in **Waste Bin**. **Waste Bin** also has a local menu **Operations**. As figure shows the first entry is already selected and we may apply an operation on it. Or, we can select another item. We can also select a range of items by clicking to one item, pressing and holding **Shift** key, and clicking on another item below or above the selected one. All items between these two including them will be selected.

Clicking on **Operations** menu the menu opens showing available commands (see figure below).



Waste Bin menu commands

There are four commands. If we select **Delete** the selected items will be deleted from the

system. On other hand if we select the **Restore** the selected items will be removed from **Waste Bin** and restored back. That means that the project will be inserted back in the projects database and will be available from **Project** menu. The library projects will be inserted back into the **Library**. Finally the components are returned into **Component Buffer** and could be accessed from there. The Empty waste bin command removes everything from the **Waste Bin** and destroys. To exit waste Bin we can select the Exit, or simply use  command.

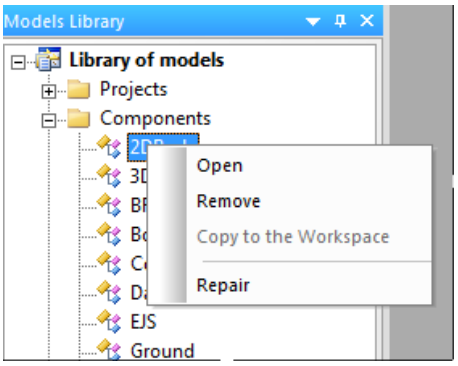
-o-

Library Operations

Opening Project or Component

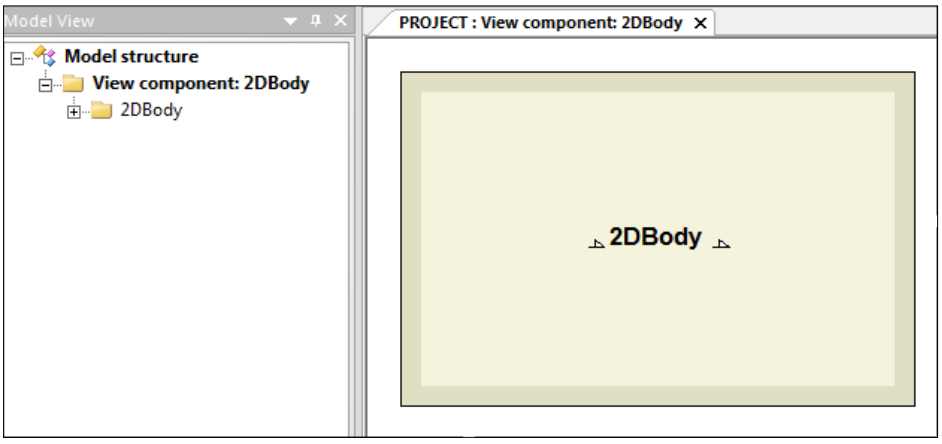
- To overview a project or component in the Model Library

Expand the **Library of models** branch. If we wish to open a project expand **Projects** branch, and in the list of project right click the project we wish to open (as we already did in Example Multilevel Model Structure). Similarly if we are interested in a component we expand the corresponding component branch. The branch **Component** contains list of the different complete component models. On other hand **Electrical** and **Mechanical Component** branches contains specific electrical and mechanical components. We find the corresponding component and right click on it. The figure below shows drop-down menu that appears after right clicking library component **2DBody**.



Library drop-down menu

Click **Open** command. The selected component root document opens in the central area (see below).



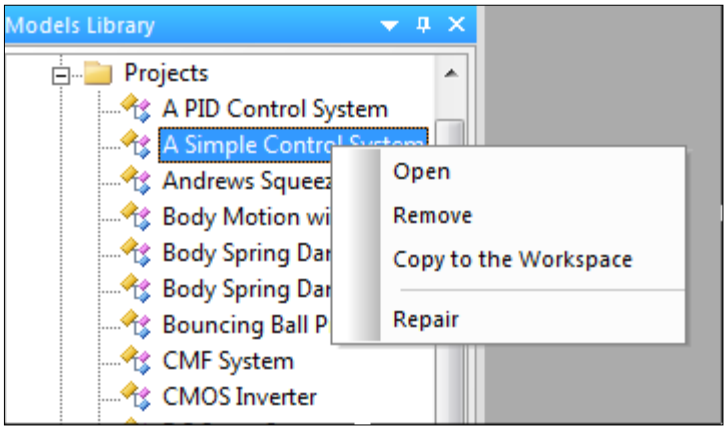
Viewing a component

Note that **View component: 2DBody** project also opens in **Model View** window on the left in the model tree and containing one branch: **2DBody** component. The model opens in the read-only mode. We can freely walk through it by opening the components as with any other project, but we cannot change anything.

Copying Project from Library into Workspace

To copy a project from the library and insert it into the workspace:

In the **Models Library** in Library of models expand the Project branch and then right click a project. Drop-down menu appears as shown below.



Library project drop-down menu

Select **Copy to the Workspace**.

The program checks first if such a project already exists in the model workspace. If there is no such a project, the project is copied and installed into the model workspace under the same name. However, if there already such a project, the user is asked to supply a new name. The copied project is then stored under this new name. When the operation was successfully completed the user is informed on it.

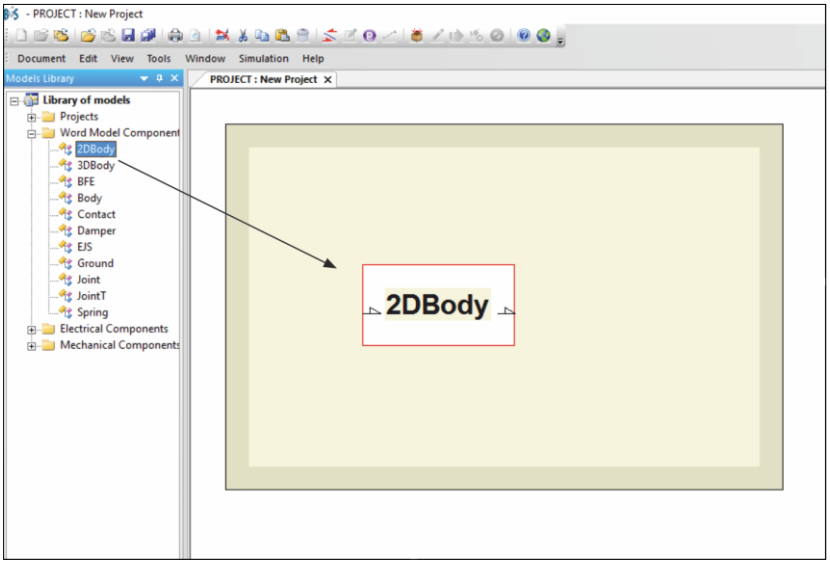
When the project is copied into the model workspace we may open it and edit it.

-o-

Inserting Component from Library

- The components from the library can be directly used in a project we are working on. Procedure is very simple:

1. Expand the corresponding component branch in the **Models Library** (**Component**, **Electrical Component** or **Mechanical Component**) and find the component.
2. Drag the component from Library and drop it at the desired place of the currently active document window (see below).



Dragging and dropping a library component

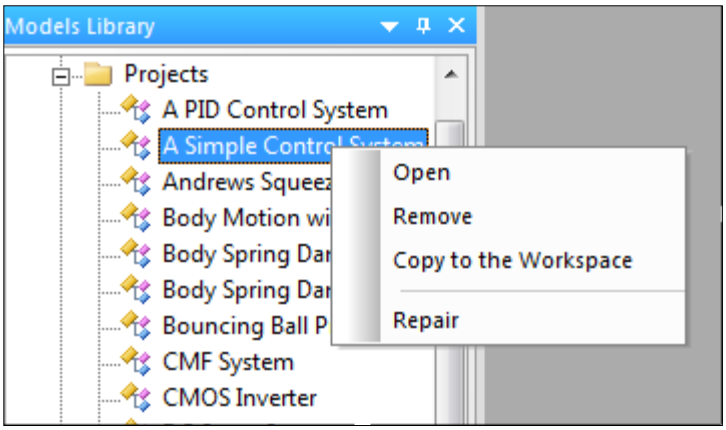
Program BondSim make a copy of the library component model and insert it into the document window together with all underlying documents and components. Thus, the complete document tree is silently inserted into the document and copied into. After it is inserted we can freely work with it.

-o-

Removing Object from Library

- We can remove a project or component from the library and sent it into the Waste Bin. Later we may delete or recover it from there (see Waste Bin operations). The operation is similar to the previous library operations:

1. Expand the corresponding projects or component branch in the **Models Library** and right click the item we wish to remove. The following drop-down menu appears:



Library drop down menu

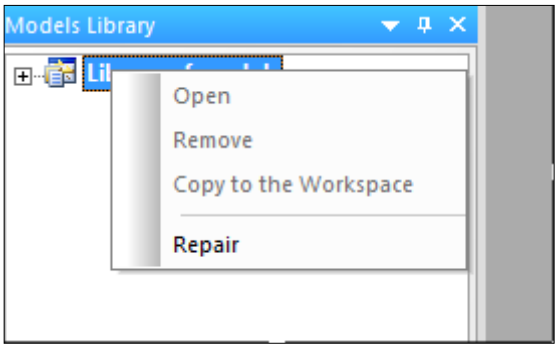
2. Select **Remove** command. Similarly to other remove operations the object of the command (project or component) is removed into the Waste Bin.

-o-

Repairing Library

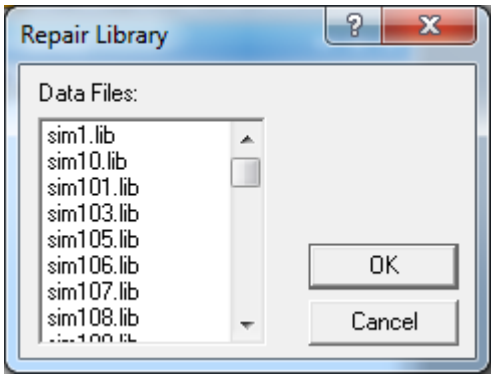
- Similarly to model space repair we may repair the Library space. Commands to repair the library workspace, in addition to the workspace restore operation this command offers also to recover the originally installed library projects or components that was removed. If we wish to do it a dialog window opens to assist us in specifying what if any project or component we wish to recover.

1. Right click anywhere on **Models Library** window. The following drop-down menu appears:



Library repair Drop-down menu

2. Select **Repair** command.
3. Dialog box **Repair Library** appears as shown below. It list all library files contained in the data base.

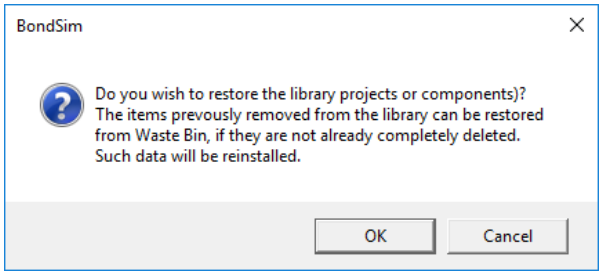


Library repair dialog

4. Click **OK** button to start the library workspace repair.

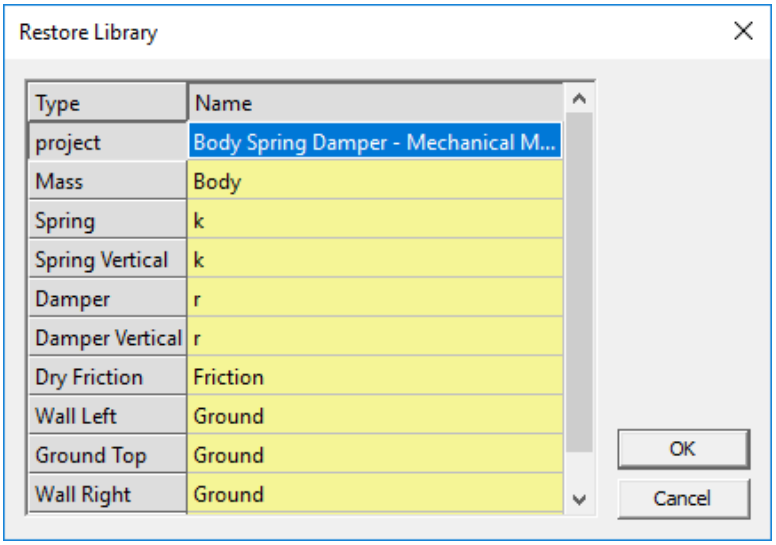
   The messages in the **Repair** tab of the **Output** window at the bottom of the program frame shows the progress of the repairing process.

   At end of this operation a dialog appears offering to recover the missing projects or components as shown below.



Recovering the library message box

If we accept this by OK and if there are missing items a dialog opens which enables us to select what we would like to restore (see below).



Dialog to restore the library items

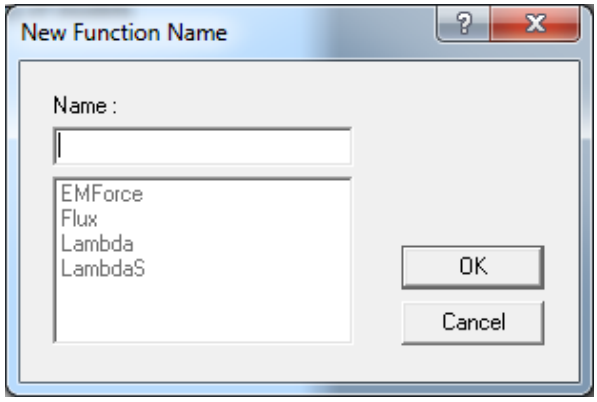
-o-

Working with Functions

Creating New Function

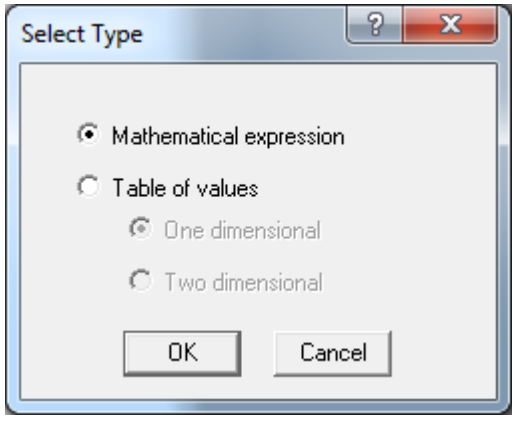
To create user-defined function

1. On the **Tools** menu, point to **Function** command and then select **New** command. The **Create New Function** dialog box appears as shown.



New function dialog

2. In the *name editing text box* type a function name you want to create and then click **OK**. Function name has to be unique.
3. The **Select Type** of function dialog box appears.



Function type dialog

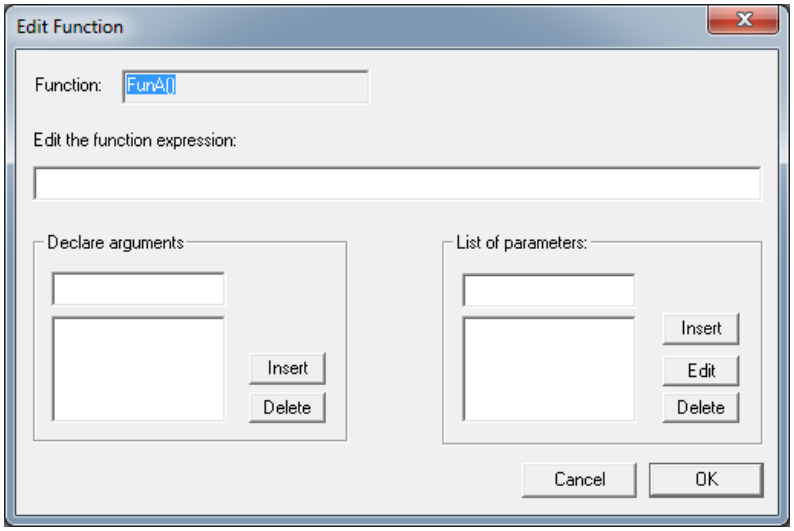
There are two types of functions that can be created

- Function defined by mathematical expressions. This is the symbolically defined function.
- Function defined by a table of values. This is the tabular function. They can be
  a. One-dimensional, or
  b. Two-dimensional.

-o-

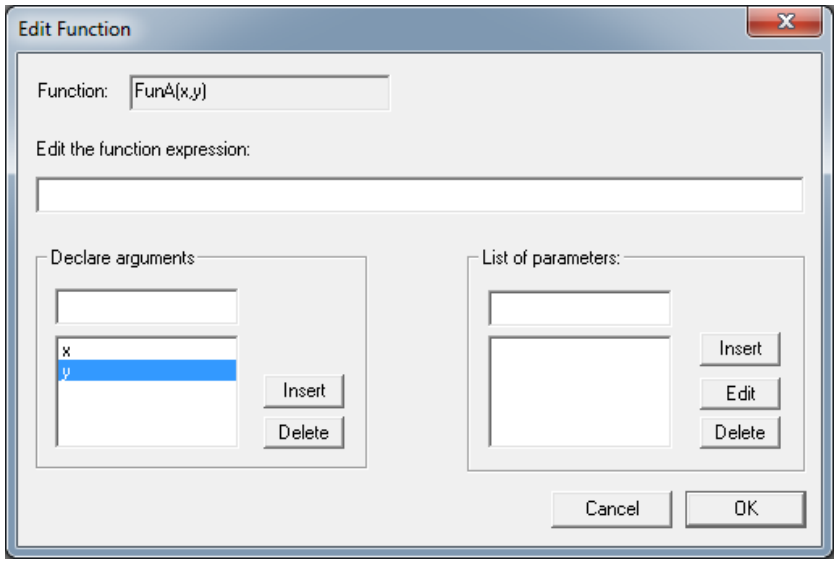
To define symbolic function

To define function by mathematical expression select option **Mathematical expression** and click OK. The **Edit Function** dialog box appears. The function name that we defined in the previous step, e.g. **FunA**, appears in the function text box and empty opening and closing parentheses were added, i.e. **FunA()**, as shown below.



Edit function dialog

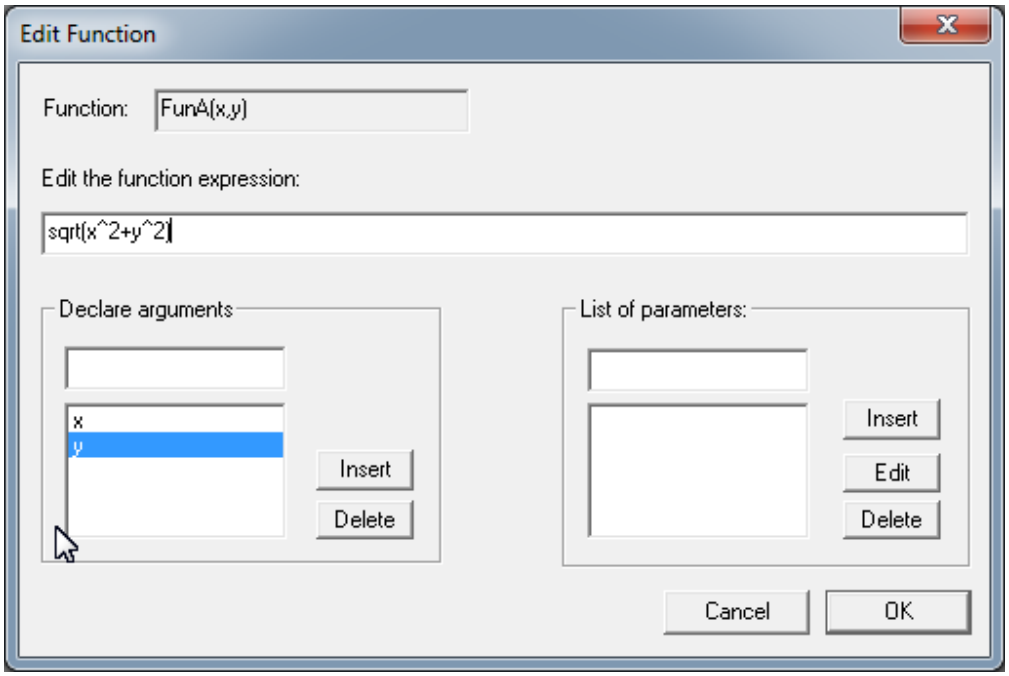
To define the arguments, if any, we may use the commands within **Declare argument** box. Thus, we may type in edit box ***x*** as the first argument and click **Insert** button. The argument is added into the list box beneath the edit box and is also inserted into FunA argument list. If we add another argument e.g. ***y*** it is also added in the list, and in the function argument list, separated by comma form the first, etc. (see figure below).



The function with added arguments

Now we can define function by editing the right side of the function. As example we may define expression as sqrt(x^2+y^2). Thus we have a function which evaluates the square root of the

sum of the arguments squared, e.g. distance of (x, y) point from the origin in plane (see figure below).



Editing function expression

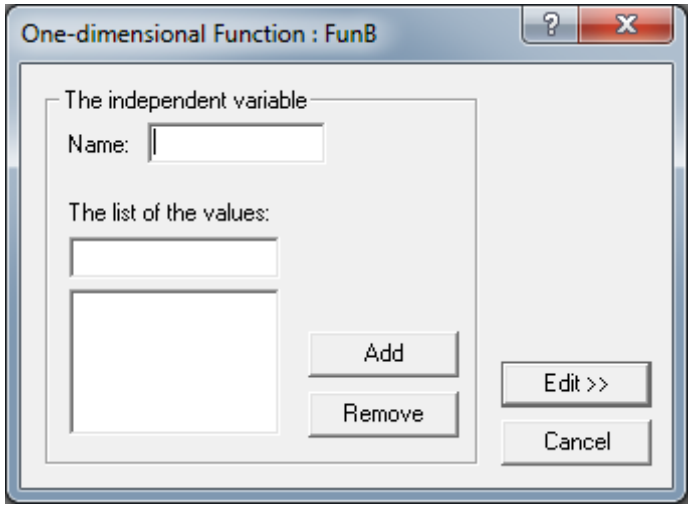
We can also use parameters defined within the function. We managed parameters using the same technique as earlier. Thus using **List of parameter** block we may insert a parameter and define its value, edit an already defined one, etc. In the same way we may remove the parameters or arguments from the function by selecting them in the corresponding lists and then applying the **Delete** button.

When we click **OK** the expression is parsed to determine if it is well defined. That means that all symbols are either declared arguments, locally defined parameters or constants. Also the rules described in Description of Constitutive Relations apply. If the test went OK the function is stored in the database. We can use the function in the model simply by calling it through its name and the argument list. It is assumed that all arguments and calculations are in double precision.

-o-

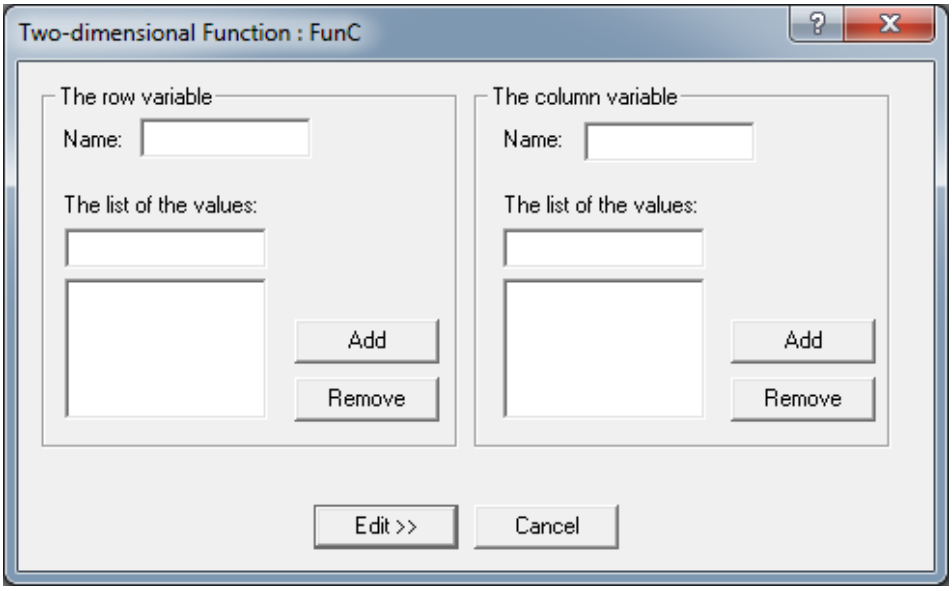
To define tabular function

1. If you wish to create a function by a table of the values select in the function type dialog **Table of values**, and either **One dimensional** or **Two dimensional** functions (see Creating New Function).
2. In the first case a One-dimensional Function dialog appears, which is used to define the values of the function (see below).



1D tabular function dialog

Similarly for two-dimensional function the following dialog appears.



2D tabular function dialog

3. In the either case define the name(s) of the variable(s) in the **Name** editing box, and write in and **Add** the values in the list box(es). The inserted value may be removed by selecting the value in the list box and click **Remove** button. As an example in 2D tabular function dialog the row is defined as x, and the column as y. The corresponding row values are added as 1,2,3,4 and 5. And the column values as 10, 20, 30, 40 and 50. The resulting dialog reads now as shown below.

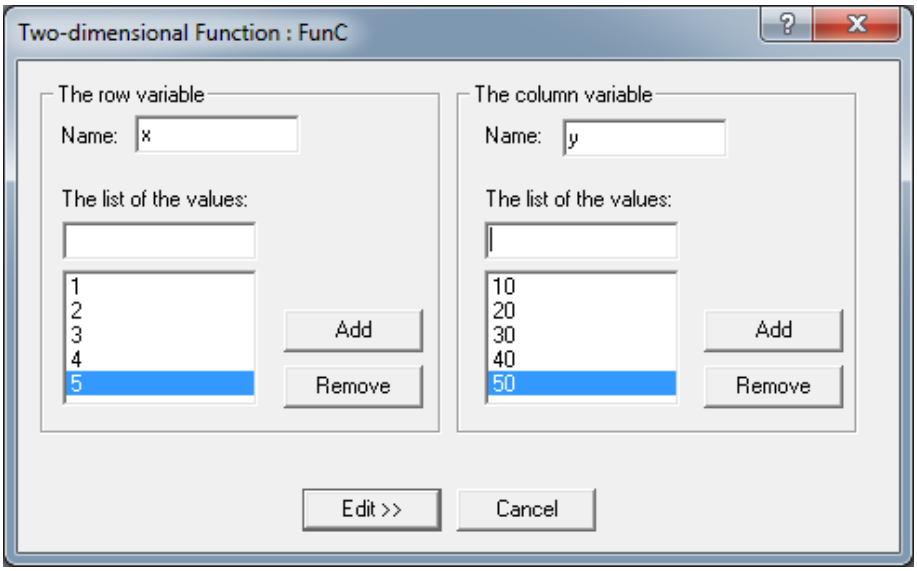
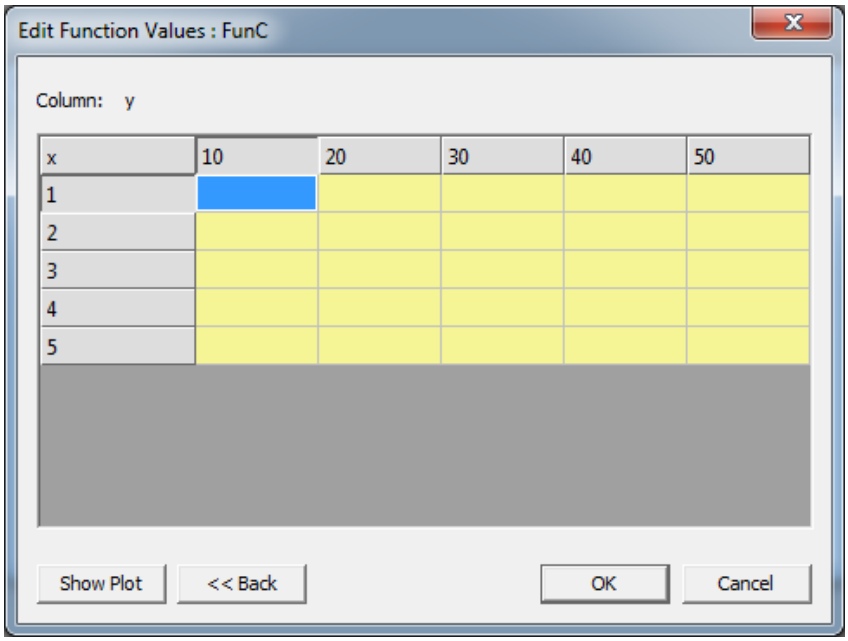


Table dialog with input values

After all values are set select **Edit** button to edit the function's values.

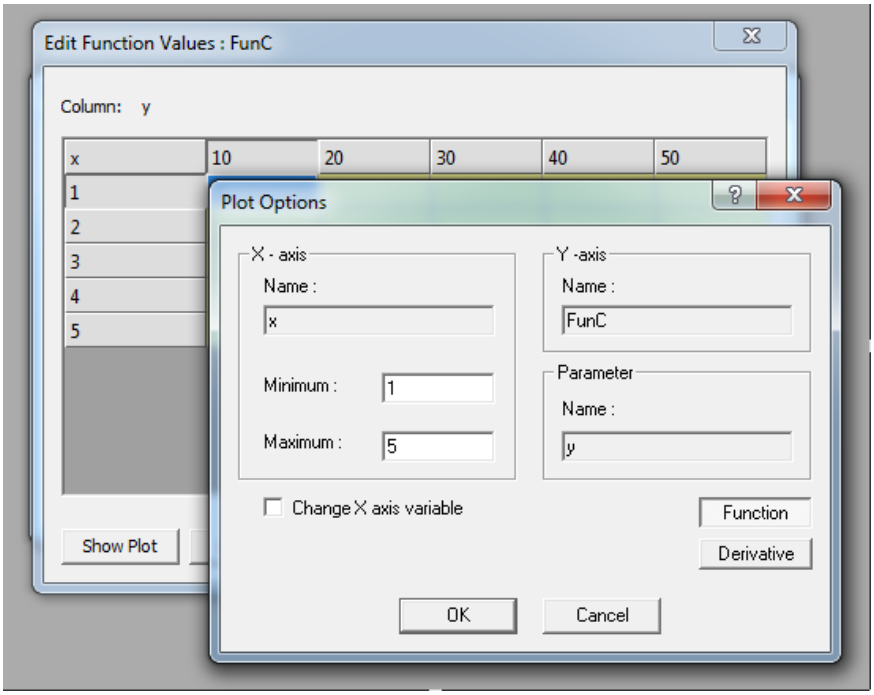
4. A tabular dialog appears which enables to define a function value for every single argument or pair of the arguments values. In figure below a two-dimensional grid corresponding to the previous dialog appears. The rows and the column values are inserted from the previous dialog. We need to type in the corresponding function values.



Grid dialog

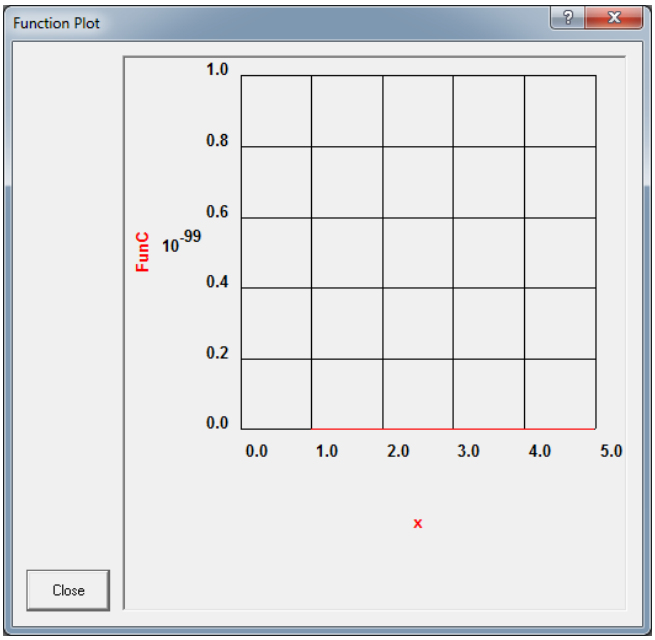
5. You can view a plot of the function defined by the inserted values by clicking **Show Plot** button. We can return to the previous dialog by using **Back** button. The plot is generated by approximating the function by single variety or bivariate cubic splines.

6. After clicking **Show Plot** button a **Plot Option** dialog opens as shown below.



Plot options dialog

7. We may select to plot the functions or its derivative. The derivative is based on cubic spline approximations of the function. We also can plot X-axis variable along the x- axis of the plot (default), or change the axis by clicking to the check box. To generate the plot we click OK. The form of the plot (without the function values) is shown below.



The form of the plot

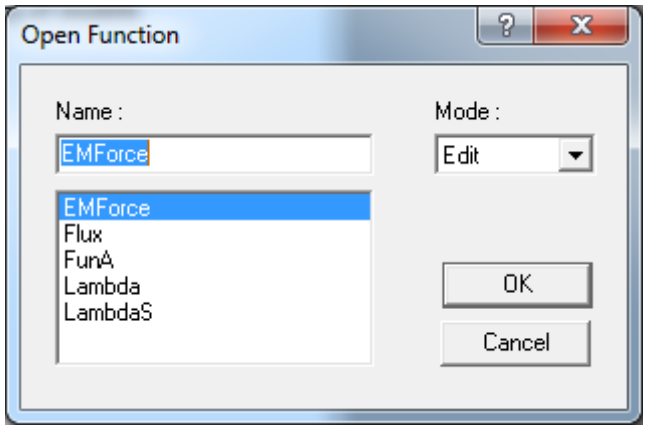
8. When the function editing is finished we click **OK** to accept and store the values generated or **Cancel** to reject them (see figure Grid dialog above). The function can be used in the constitutive relations of the component models by calling its name and the arguments. Thus in the case of the above function we call it as **FunC(x, y)**. The case is important. The tabular functions are approximated by single variety or bivariate cubic splines, i.e. by the cubic polynomials.

-o-

Opening Already Defined Function

To open already user-defined function

1. On the **Tools** menu, point to **Function** command and then click **Open** command. The **Open Function** dialog box appears.



Open functions dialog

2. Double-click the function name in the list you wish to open, or either click or type the function name in the *name text box* and then click **OK**.
3. The corresponding dialog windows open, which support editing of the function. For details see
   - To define Symbolic Function
   - To define Tabular Function

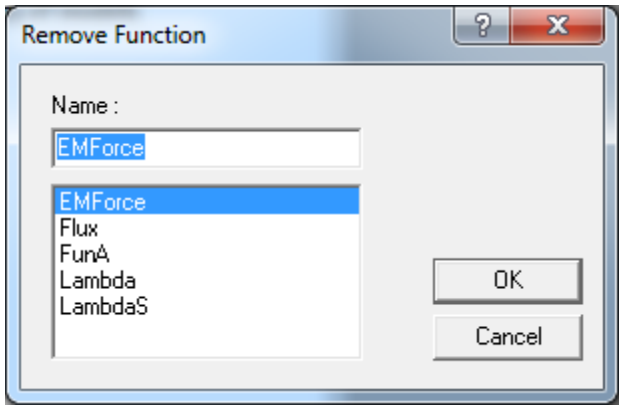
Note: Functions can be access from any project or component. Thus, every care must be taken not to change a function used elsewhere.

-o-

Removing Function

To delete already user-defined function

1. On the **Tools** menu at the default project level, point to **Function** command and then select **Remove** command. The **Remove Function** dialog box appears as shown below.



Remove function dialog

2. Double-click function name in the list you wish to remove, or either select or type the function name in the *name text box* and then click **OK** button. After dialog closes the function is removed from the functions database and moved into **Waste Bin** from where it can be removed completely or recovered back.

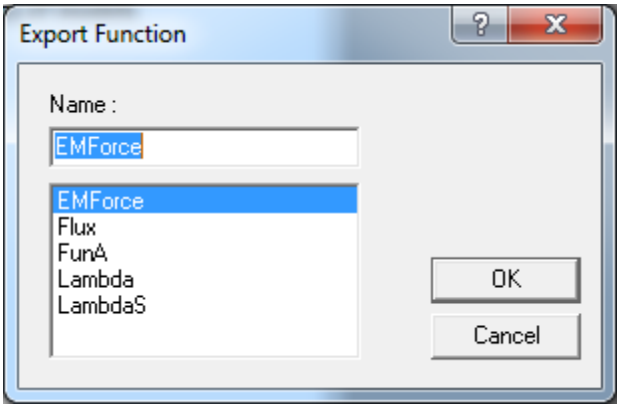
Note: The functions can be access from any project or component. Every care, thus, must be taken not to remove a function used elsewhere.

-o-

Exporting Function

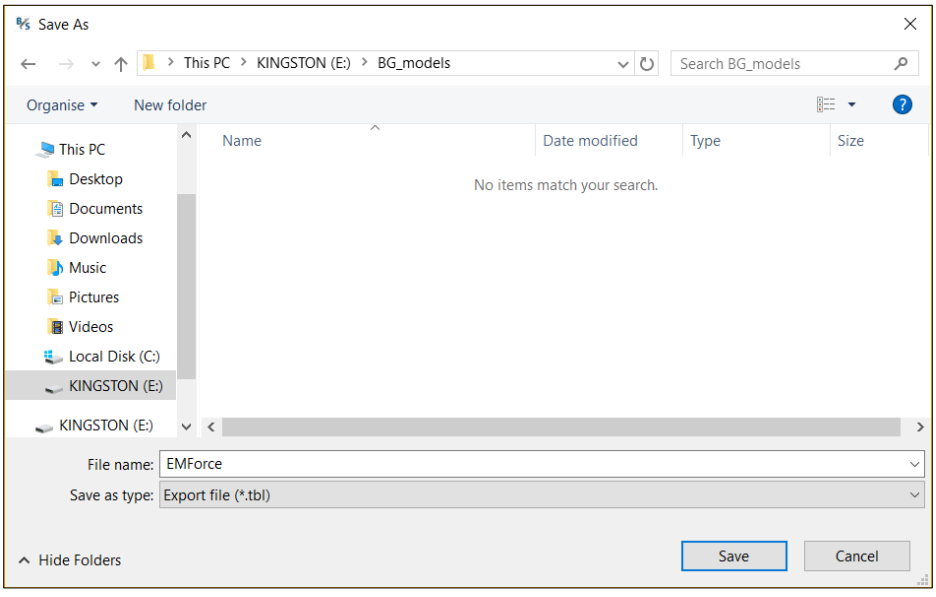
To export function

1. On the **Tools** menu at the default project level, point to **Function** command and then select **Export** command. The **Export Function** dialog box appears.



Export function dialog

2. Double-click **function name** in the list you wish to export, or either select or type the function name in the *name text box* and then click **OK**.
3. The **Save As** dialog box appears in which you can choose destination folder and type the name of the exported function.



Save As function dialog

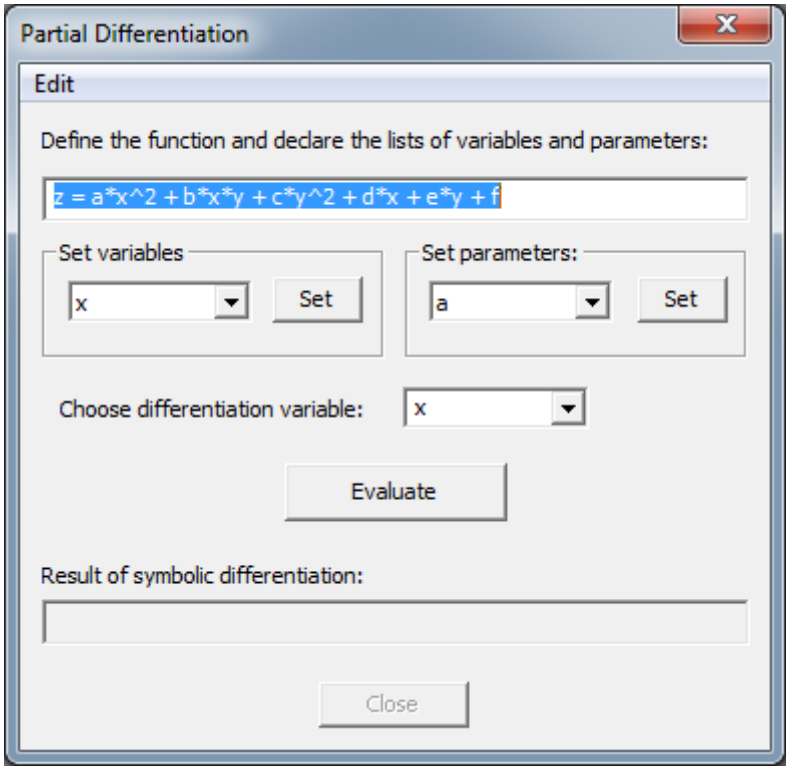
4. Selecting **Save** an export version of function (*.tbl or *.fun) will be created in the destination folder.

-o-

Symbolical Function Differentiation

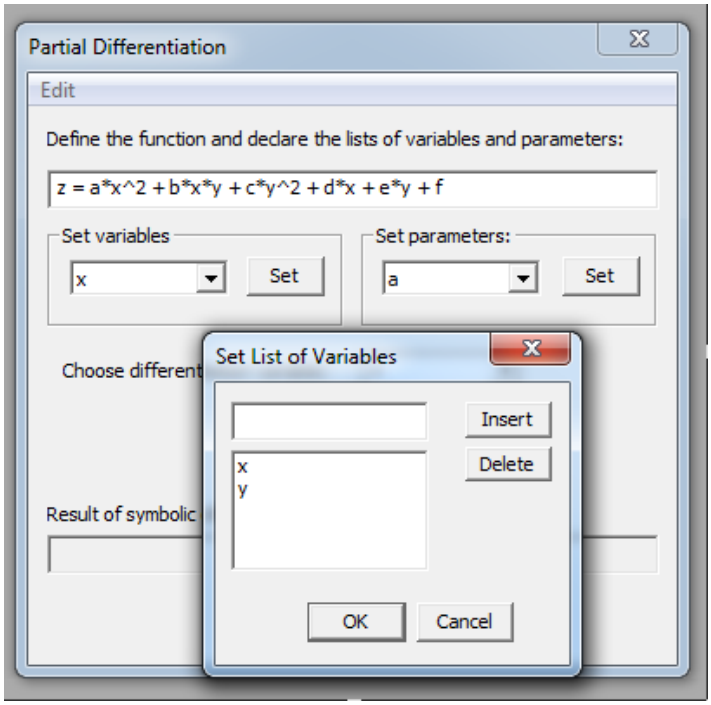
To create derivative of symbolically defined function

1. On the **Tools** menu at default project level, point to **Symbolic Operations** command and then select **Differentiation** command. The **Partial Differentiation** dialog box appears as shown below.



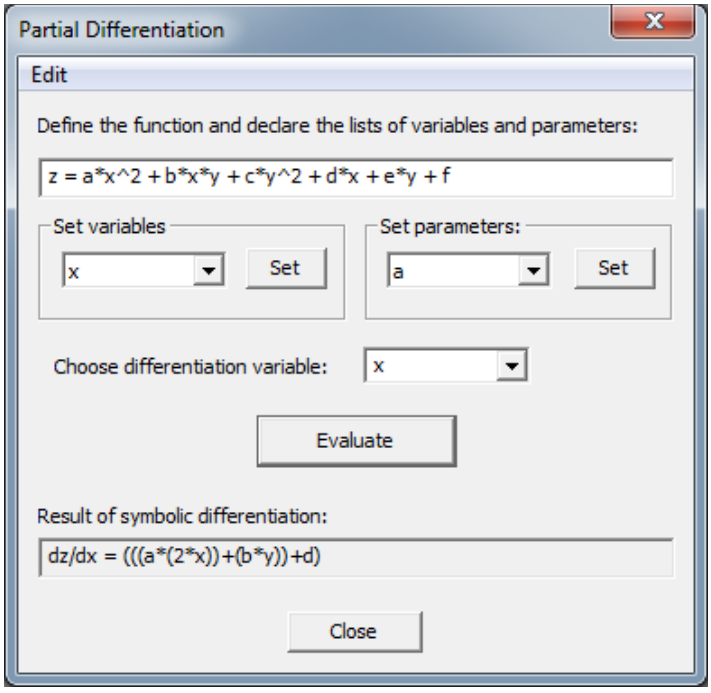
Function partial differentiation dialog

2. In appropriate *text box* define the function which derivative you wish to create. An example of the function is already shown.
3. Now you have to define variables and parameters. By clicking **Set** buttons the **Set List of Variables** (see below) or the **Set List of Parameters** dialog box appears weather you want to set variables or parameters.

Set list of variables dialog 

Typing variables or parameters in the corresponding boxes and clicking **Insert** we add them into the corresponding lists. When you define all variables or parameters clicks **OK**. These dialog boxes offer also deleting of some variables or parameters.

4. Finally, choose the differentiation variable from the corresponding list box and press **Evaluate** button. The result of evaluation is shown in figure below.

Partial Differentiation evaluation 

-o-

Project Simulation

Building Model

Before starting the simulation of developed project's model is necessary to build its mathematical model. During that phase all connected model documents starting from the project root documents are visited several times. First all bond and signals are followed to find out if all components are properly connected. After that the unique set system variables are generated, and finally the mathematical model for the system is generated. The model is not generated in human readable form, but in the form which can be more efficiently evaluated at run-time.

To build the mathematical model

1. Open the project which behavior you wish to analyze.
2. From the **Simulation** menu, choose **Build** command, or click toolbar button  , or use key shortcut *Ctrl* + B.

The program asks if we would like to save the project first to the disk. We can click **Yes** or **No**. The program then starts building the model. During the lengthy build operations a progress control bar appears on the right end of the status bar and shows percentage of completion of different build phases.

During the build operations the messages are written in Output window under Build tab. If an error is found the build operation is stopped and program tries to open document window where error was found. We can save the message written by right-clicking on Build window and from drop-down menu select Copy command. The program then open **Save As** dialog which can be used to define the name and folder where the text file containing the messages are saved. The next text box shows typical messages generated during a successful build.

The build messages

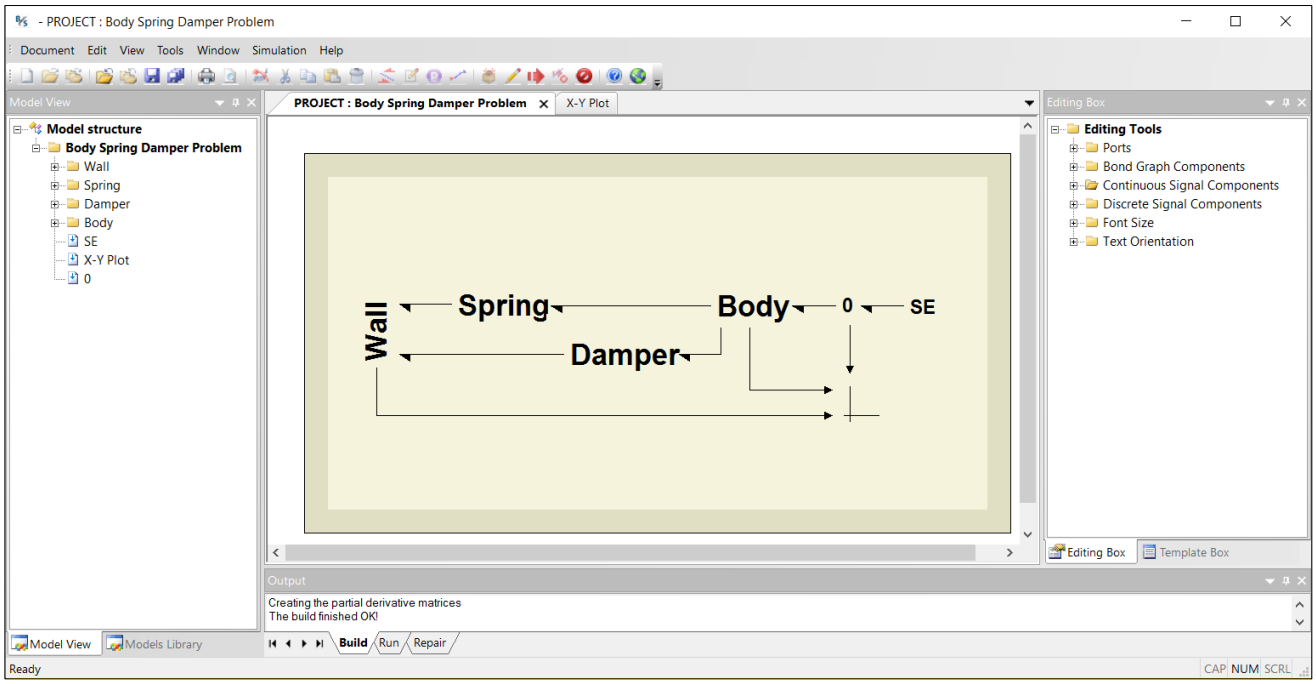
```
Building the simulation model
Creating model variables
Compiling the model equations
Creating the partial derivative matrices
The build finished OK!
```

Program generates the model equations in implicit form. Special partial derivative matrices are generated as well. These matrices are necessary for solving problem using BDF solver. The matrices are generated in analytic form, which improves stability of the solution process.

After finishing the build operations models cannot be changed. Thus, we can open the components to inspect the constitutive relations, but cannot change the model. Only the parameter values

could be changed. During the build operations all X-Y displays in the model are represented by separate plot documents, which are currently are not active. The figure below gives the form of the central windows of BSD Project after the build.

The problem screen after build



-o-

Showing Model Size, Equations and Matrix of Partial

The generated model can be inspected if we wish. To that means we must first decompile the model equations into human readable form. For this purpose serve the commands in **Simulation** menu under submenu header **Show Model** with subcommands:

- Model Size
- Write Equations
- Matrices of Partials

To show the Model Size

On the **Simulation** menu, point to **Show Model** and select **Model Size.** Information of Model Size will appear in the **Output** window under Build tab. We can right-click the window and form drop-down menu select Copy operation. Using save As dialog we can save the text file with generated data in suitable file and folder. The generated results are shown in the text box below.

The problem model size

```
The overall size of the model:
The total number of the equations: 14    The length of generated byte
codes: 236 chars
The total number of the partial derivative matrix entries: 31
The length of generated byte codes: 261 chars
The total number of the time rate variables matrix entries: 3
The length of generated byte codes: 27 chars
```

Note that model is not in the common mathematical expression form, but the special byte code form. Thus, its length is expressed in the number of characters. Here it is rather simple problem and small model having only fourteen equations. We will see that equations are very simple.

To show the mathematical model

From the **Simulation** menu, point to **Show Model** and choose **Write Equation**, or click  button in the **Toolbar**. The mathematical model is decompiled in common equation form and written in the Output Build window. From that we can similarly as we did above copy its contents in the suitable text file. The results are shown in the text box shown below.

The mathematical equations data consist of several groups of data. First there are mathematical equations written in the implicit form, i.e. all terms are transferred on the left side, which is equal to zero. There are fourteen such equations. Every equation typically contains two or three terms. Thus, we have a number of near trivial equations. This is the characteristics of BondSim models! The meaning of the variables will be clear shortly.

The next group gives list of differential variables, i.e. variables which appear in the equation differentiated with respect to time. These are the common state variables, if they are not related by the equations. If some of these are not independent they are termed simply the *differentiated* variables.

Model equation

Equations of the model:
EQ(1) = Y'(9)-Y(8) = 0
EQ(2) = Y'(12)-Y(11) = 0
EQ(3) = Y'(13)-Y(5) = 0
EQ(4) = ((-Y(2))+Y(3))+Y(4) = 0
EQ(5) = (-Y(7))+Y(1) = 0
EQ(6) = ((-Y(1))+Y(5))-Y(8) = 0
EQ(7) = ((-Y(1))+Y(5))-Y(10) = 0
EQ(8) = (((-Y(3))+Y(6))-Y(4))-Y(11) = 0
EQ(9) = (-Y(5))+Y(14) = 0
EQ(10) = Y(7)-P(1) = 0
EQ(11) = Y(3)-(P(2)*Y(9)) = 0
EQ(12) = Y(4)-(P(3)*Y(10)) = 0
EQ(13) = Y(12)-(P(4)*Y(5)) = 0
EQ(14) = Y(6)-P(5) = 0

List of differential variables:
Y(9), Y(12), Y(13)

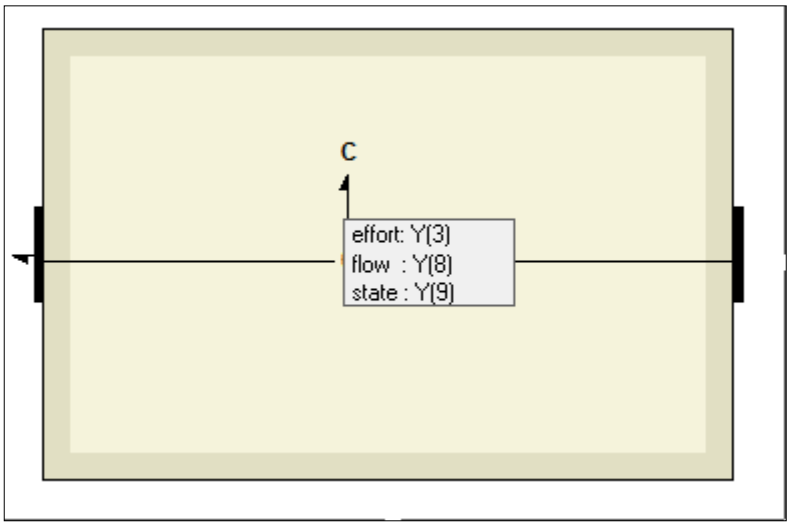
Indexes of the algebraic equations:
EQ(4), EQ(5), EQ(6), EQ(7), EQ(8), EQ(9), EQ(10), EQ(11), EQ(12) EQ(13), EQ(14)

The initial value expressions:
Y(9)= 0 Y(12)= 0 Y(13)= 0 Y(7)= P(1) Y(6)= P(5)

Parameters:
P(1) = 0, P(2) = 112500, P(3) = 150, P(4) = 5, P(5) = 500

The next group is the pure algebraic variables; then the initial condition expressions, and finally the parameters.

To see which variables in the equations correspond to which in the bond graph model just move the mouse pointer over the corresponding elementary component port. To show this, we open **Spring** component model and as we move cursor over the **C** component port a tooltip appears showing the effort, flow and state variables (see the figure below).



C component port variables

In similar way we can found the meaning of other variables.

- **To show the matrix of partial derivatives (Jacobian)**

On the **Simulation** menu, point to **Show Model** and choose **Matrices of Partials.** The matrix of partial derivatives, represented in the coordinate form, appears in the **Output** window from which we can copy the content into a text file by right-clicking. A part of the generated matrix (10 out of 31 elements) is shown in the text box below.

```
Matrix of partial derivatives:
J(1) = -1
IRow(1) = 1    JCol(1) = 8
J(2) = cj
IRow(2) = 1    JCol(2) = 9
J(3) = -1
IRow(3) = 2    JCol(3) = 11
J(4) = cj
IRow(4) = 2    JCol(4) = 12
J(5) = -1
IRow(5) = 3    JCol(5) = 5
J(6) = cj
IRow(6) = 3    JCol(6) = 13
J(7) = -1
IRow(7) = 4    JCol(7) = 2
J(8) = 1
IRow(8) = 4    JCol(8) = 3
J(9) = 1
IRow(9) = 4    JCol(9) = 4
J(10) = 1
IRow(10) = 5    JCol(10) = 1
.....
```

A part of the matrix of partials

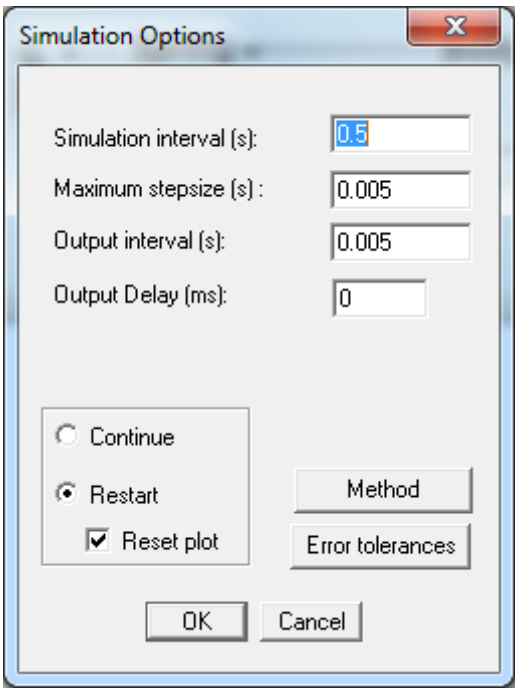
-o-

Running Simulation

To run simulation

1. Open the modeling project and build it.
2. From the **Simulation** menu, choose **Run**, or click Toolbar button  , or use shortcut *Ctrl+R.*

The **Simulation Options** dialog box appears.



Simulation option dialog

The **Simulation interval** defines the length of time in *s* during which the processes is simulated. This depends on the problem that is solved.

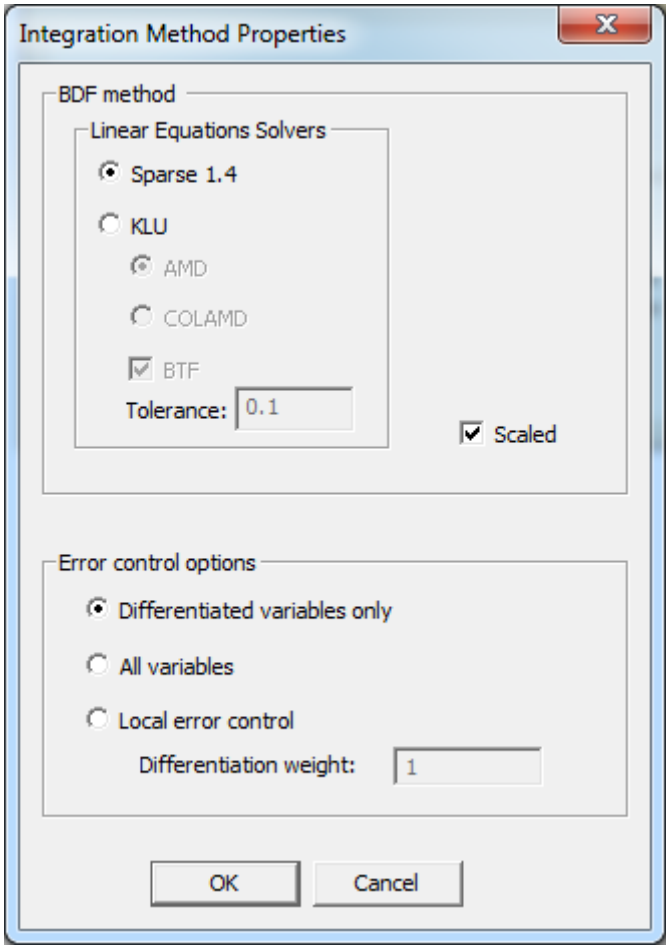
The **Output interval** defines the time interval after which the outputs are repeatedly generated and displayed. It defines the resolution of the generated plots. Typically it is taken at about hundred points per simulation interval and thus the output interval is about 1/100 of it. It can be much less if we wish plots with good resolution. Thus, if we are interested in the system frequency characteristics we may ask for 1000 or more points.

The **Maximum stepsize**, on the other hand, is related to the numerical solver. BondSim use **BDF** (Backward Differentiation Formula), which is variable size and variable order solver. The **Maximum stepsize** gives the upper limit to variable sizes. By default it is set equal to **Output interval**, but can be lower.

The last time interval in the dialog shown above is the **Output delay**. It is related to the delay in generating outputs which are sent by IPC to a server, e.g. **BondSimVisual** program for

generating 3D visualization. By, default it is set to 0, i.e. it is not used. It is used when both programs work together.

We may set the integration method by clicking on **Method** button in the dialog above. The corresponding dialog opens as shown below.



Integration method dialog

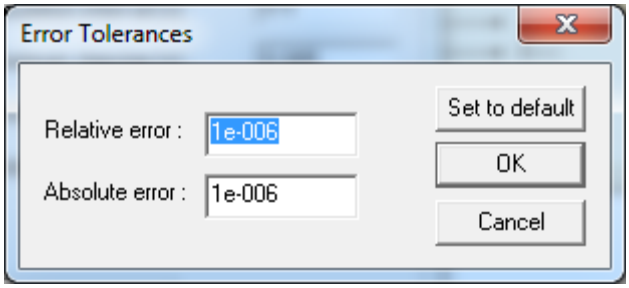
The solver BDF in BondSim currently uses two types of linear equation solvers. The default method is Sparse 1.4. It is well known the sparse matrices solver and it works well. However, there is also more modern solver based on KLU, and it is up to user which solver to use. The model equations are also scaled by default. It is advised to use it.

The next part deals with the integration error control options. The default method is to control only the differentiated variables (see: To show the mathematical model). Thus, the algebraic variables are not included in the integration error control. However, this does not mean that they are not controlled; all variables are generated in such a way that all equations of mathematical model are very close to zero. However, the integration error, as a separate control mechanism, is based on the first one.

The other possibility is to include all variables in the integration control, or to apply the advanced Local error control. It should be pointed out that mathematical model for some

problems can be very difficult to solve. In that case the relieving of the error control is necessary. This appears typical when solving multi body problems such as in robotics.

Returning to *Simulation option dialog* above, we can see that there is also button **Error Tolerances**. If we click on it a dialog shown below appears



Error tolerances

There are two error tolerances: relative and absolute. The default values are set to 1.0 e-6. Often it gives the good results, but sometimes we may apply a stricter control e.g. 1.0 e-8 or less.

When we start simulation for the first time the Simulation options dialog as shown above sets the **Restart** option and checks the **Reset plot**. We normally set **Simulation interval, Output interval** and **Maximum stepsize** as discussed above. We may accept Method and Error tolerance defaults and click OK to start simulation. When simulation starts a messages are written in the Output windows, under Run tab. During the simulation a progress bar appears on the right side of the Status bar showing the simulation progress in percent. The typical messages after the successful simulation are shown in the text box below.

```
The simulation run starts.
Initializing the simulation.Running the simulation.
The simulation run finished OK!
The BDF variable coefficient solver was used.
The model was evaluated using CLR.
```

Simulation run messages

After completed simulation run, we may continue simulation by increasing Simulation interval and selecting the **Continue** option (see Simulation option dialog above).

-o-

Stopping Simulation

To stop simulation running

From the **Simulation** menu, choose **Stop Run**, or click **Toolbar** button  , or use shortcut *Ctrl+Q*.

Simulation will be stopped and message *'The simulation run forced to stop!*' appears in the Output Window.

-o-

Ending Simulation Session

- **To end the simulation session**

  From the **Simulation** menu, choose **End Simulation**, or click **Toolbar** button  , or use shortcut *Ctrl+Shift+Q.*

-o-

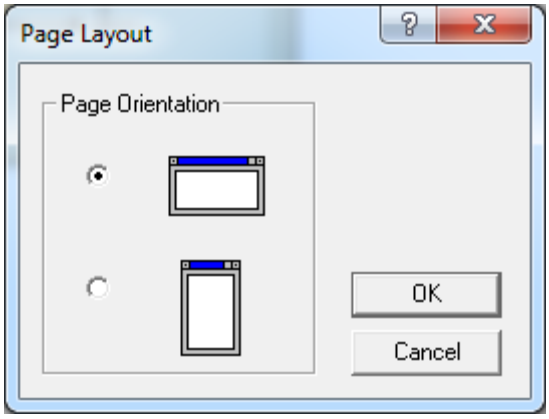
Miscellaneous

Setting Page Layout

- **To set document orientation**

From the **Project** menu, choose **Documentation** menu and then point to **Page Layout.**

The Page Layout dialog box appears:



Page layout dialog

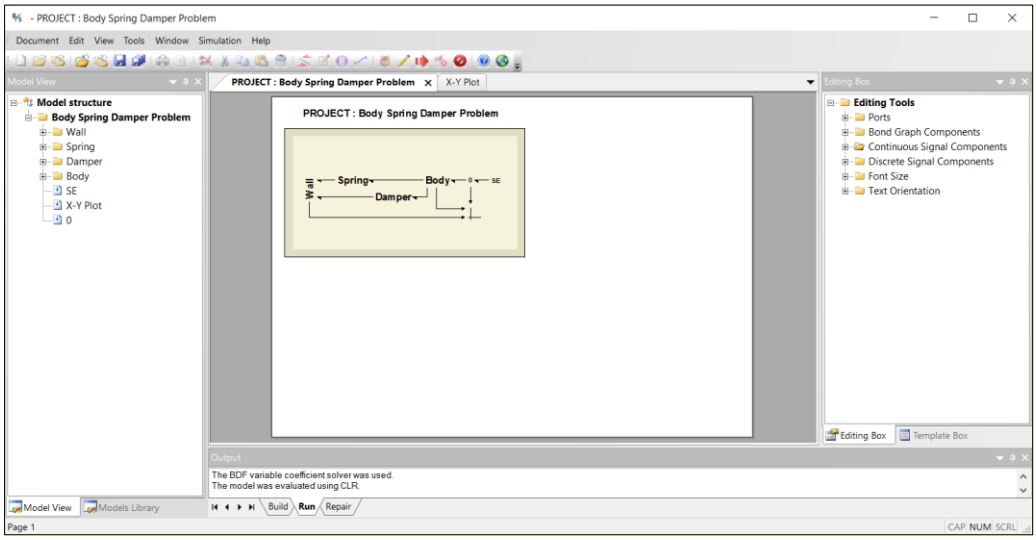
The default model document orientation is horizontal, but we can change it to the vertical. This is the orientation of the generated document page and is used when applying Print or Print Preview of the active document.

-o-

Printing the Document

To print preview document

From the **Documentation** menu select **Print Preview**, or click **Toolbar** button  . A print preview screen appears as shown below. Also the cursor takes the form of the magnifying glass. Thus, clicking to the document magnifies it. Clicking it several times it returns back to the original size.

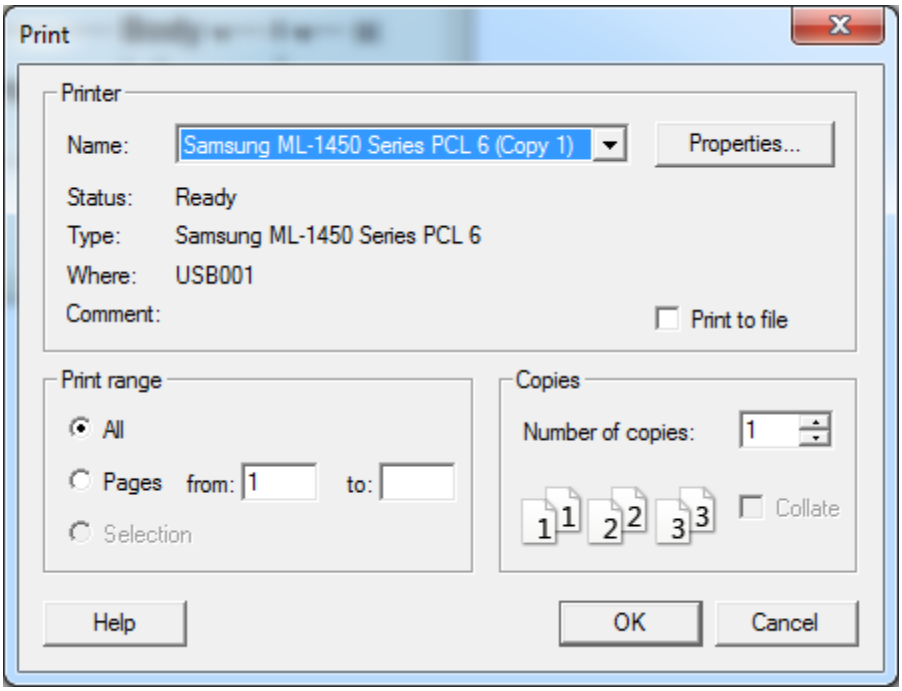


Print preview screen

Note: To close Print Preview click on the close button on the right upper corner of the document.

To print document

From the **Documentation** menu select **Print** command, or select **Toolbar** button  , or use shortcut *Ctrl*+P. The standard print dialog appears as shown:



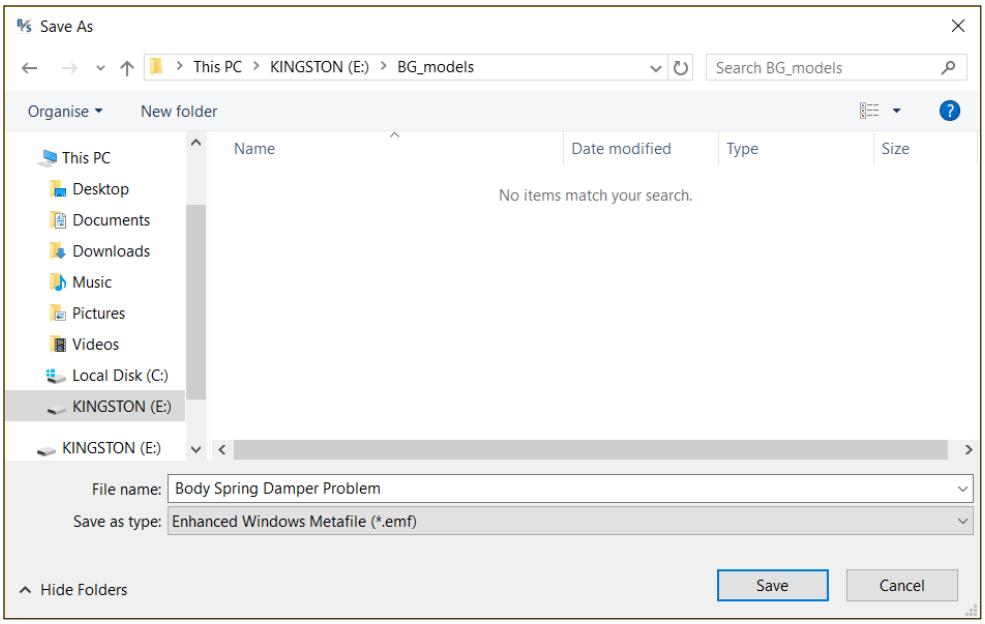
Standard print dialog

The currently active document is sent to printer. The document is printed in textual form giving the relevant information of all model objects and their characteristics. If we need the pictorial representation of the document we can use another print command: **Print To File**.

To print document to file

From the **Documentation** menu select command **Print to File**.

The **Save as** dialog box appears (see below) in which we can defined file name and choose the folder we wish to save to the current document in the graphic form using *Enhanced Window Metafile* (emf) format.



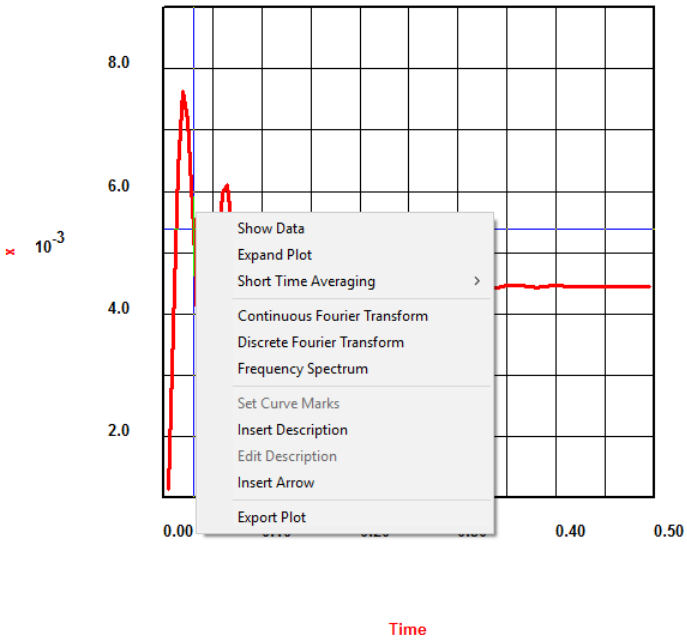
Print to file dialog

These component model files can be directly inserted into the Word documents. Many Bond Graph models in this Help are generated using this technique.

-o-

Operations on Plots

- By right-clicking the displayed x-y plot a drop-down menu appears containing many useful commands as shown below.

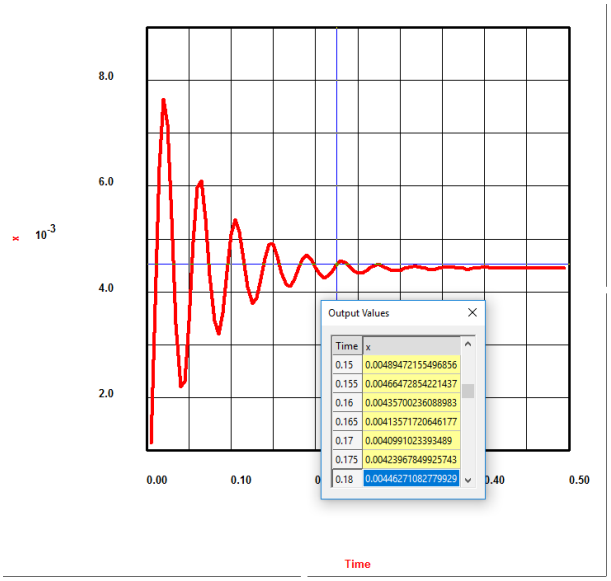


Drop-down plot menu

We describe these commands:

- **Show Data**

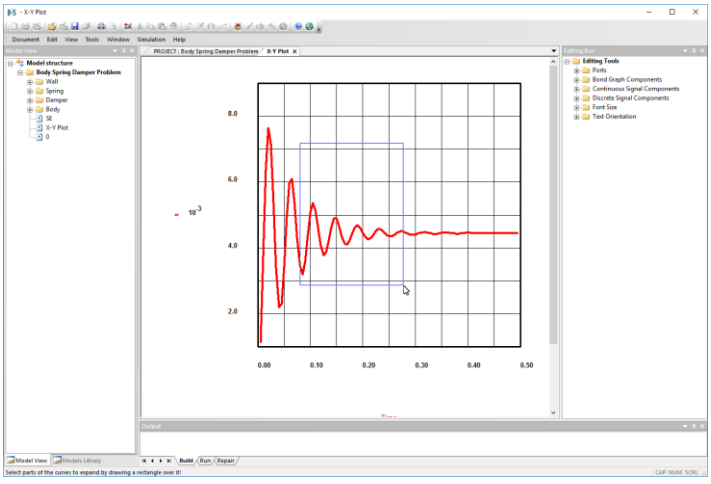
  By selecting this command a window opens which list the values of the right-clicked curve (see the Figure below). The values listed are symmetrical about the clicked point.



Window shown the Output Values

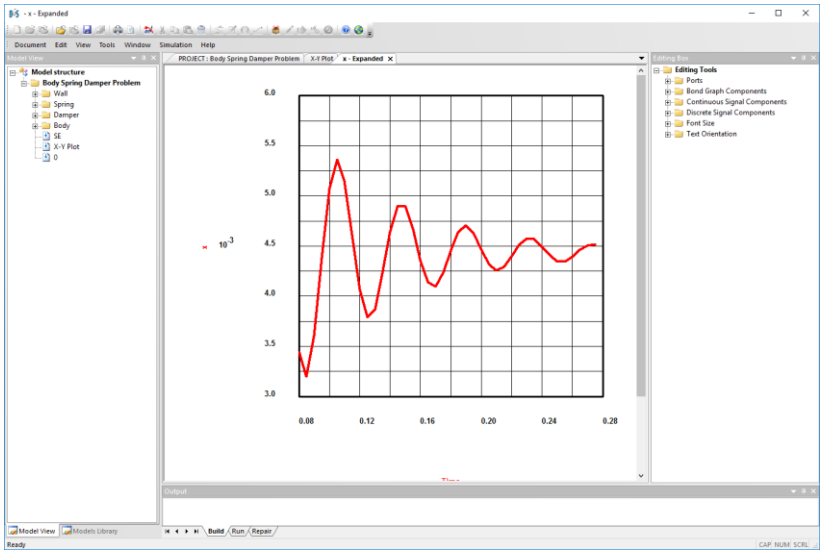
- **Expand Plot**

  After this command was selected we put the mouse curser above a part of the plot we which to expand, press the mouse and hold it. By holding down the mouse we drag it in the south-east direction and a selection rectangle in blue color appears as shown below. We can also drag the mouse in the opposite direction if we wish



Selection rectangle

We release the mouse when we have covered the part of the plot we wish to expand. As exampled, plot of selected part appears in a new window as is shown below.

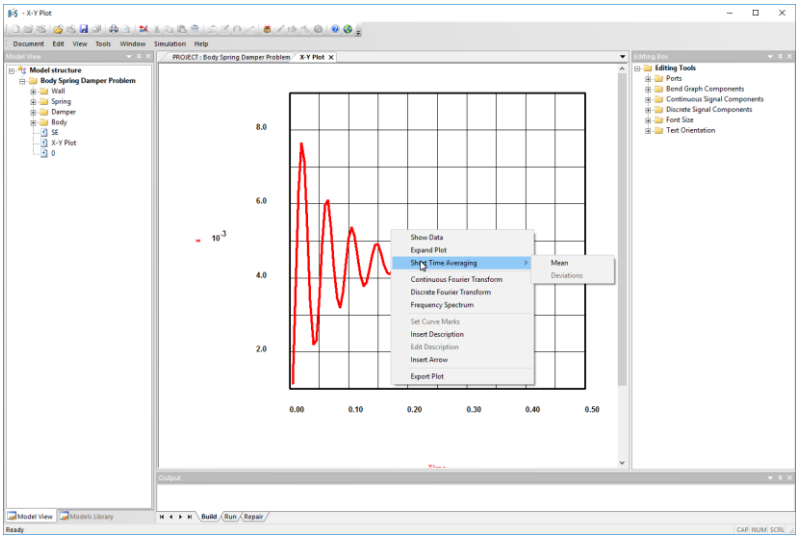


Expanded plot

Short Time Averaging

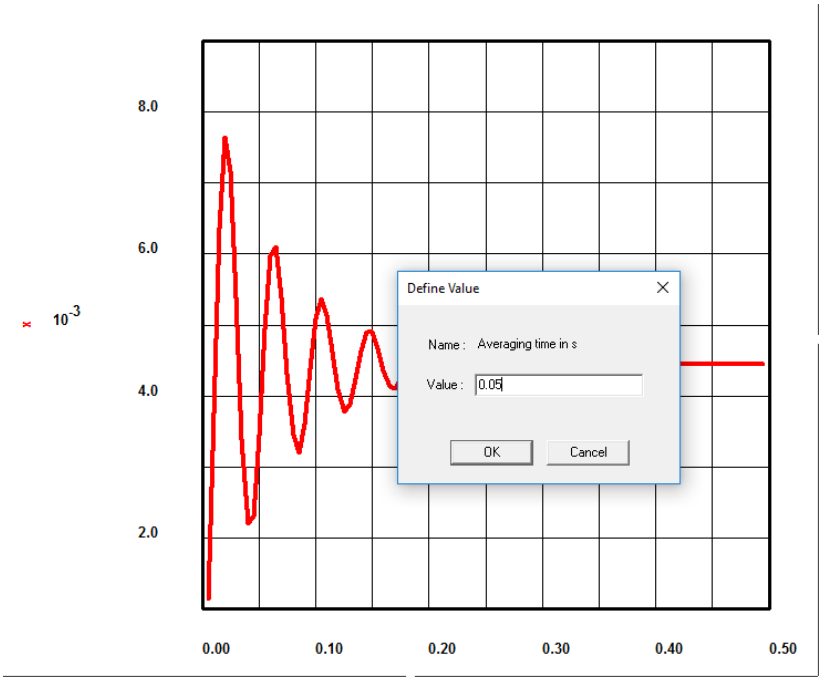
The command is available only if there is only one curve displayed in the plot. This command applies a short time moving average on the curve. It is commonly used for smoothing out short-term fluctuations and highlight longer-term trends or cycles. The threshold between short-term and long-term depends on the application. The mean is normally taken from an equal number of data on either side of a central value. This ensures that variations in the mean are aligned with the variations in the data rather than being shifted in time.

When we put the mouse cursor on the command a submenu appears showing two commands: *Mean* and *Deviations*, where only the first is active (see figure below).



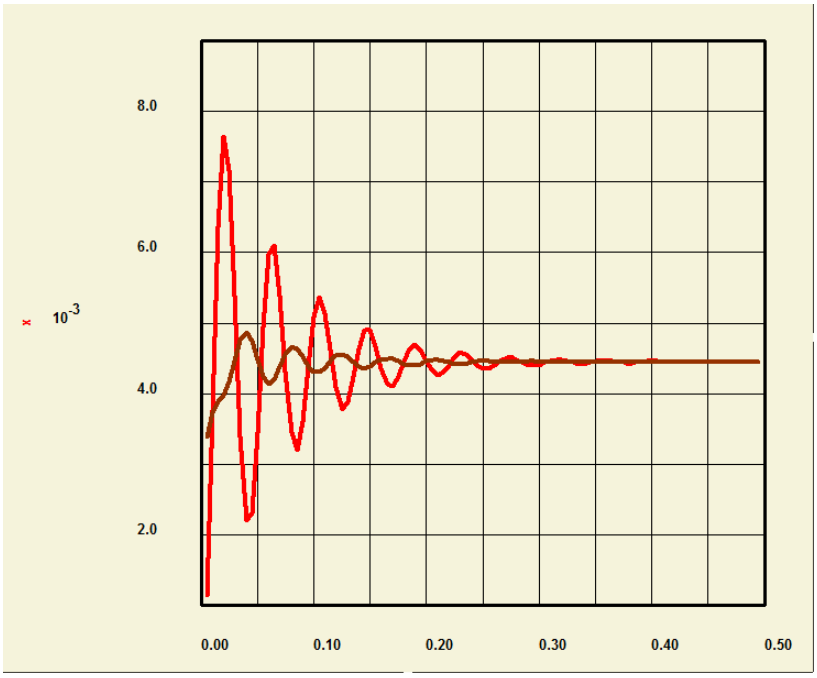
The averaging subcommands

Selecting the *Mean* the window shown below opens in which we can type the length of the moving window. We type 0.05 s for the time window length, which approximately corresponds to the period of the damped vibrations of the process.



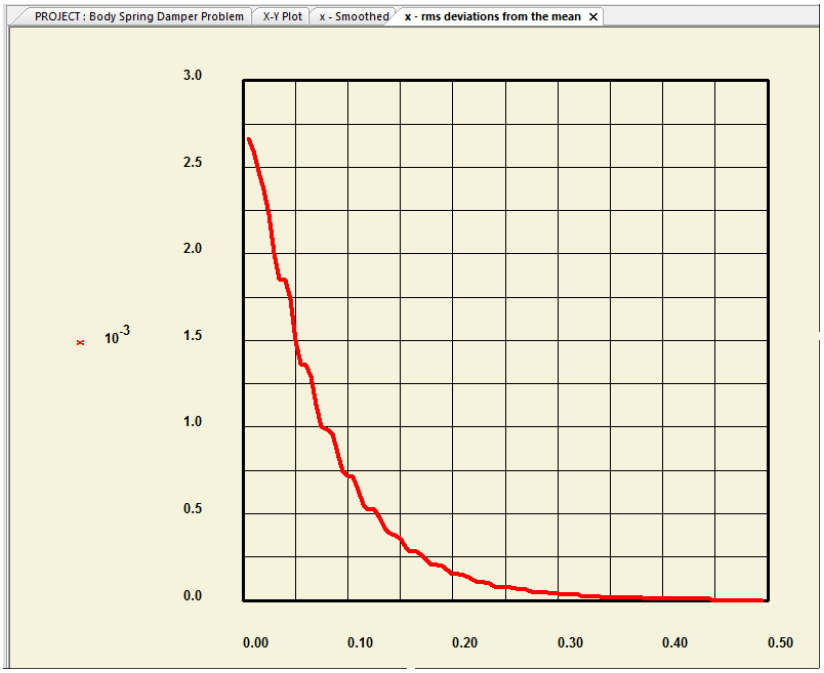
Setting the time length

Next, we click OK and the smoothed curve appears plotted over the original one.



The smoothed curve

If we right-click on the new plot and put the cursor on the *Short Time Averaging* command the same subcommands appear as before, but this time the *Mean* is inactive and *Deviations* is active. Selecting the last one a new plot appears displaying the root-mean square deviations of the original curve from the averaged (see figure below).

Rms deviations 

- **Continuous Fourier Transform**

The command may be applied on a single curve. By selecting the command the continuous Fourier Transform of the plotted function is generated. The continuous Fourier transform is defined by

$$G_c(f) = \int_0^{(N-1)T} g(t) e^{-2\pi f t} dt$$

We may approximate this integral by applying the rectangular integration rule. Hence, we divide the integration range by the strips of width equal to the output interval T, and of height equal to $g(kT)$, (k=0,1,…,N-1). Thus, from above equation we obtain

$$G_c\left(\frac{n}{NT}\right) \approx T \sum_{k=0}^{N-1} g(kT) e^{-j2\pi nk/N}, \quad n = 0,1,\ldots,N-1$$

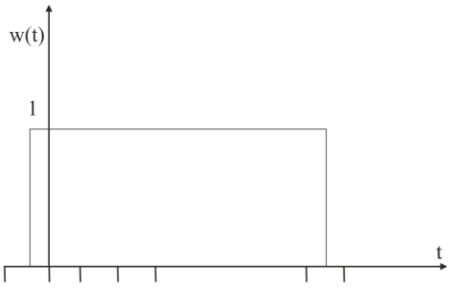
Thus, we approximate the continuous Fourier Transform by the discrete Fourier Transform multiplying the last by the output interval,

$$G_c\left(\frac{n}{NT}\right) \approx T G_d\left(\frac{n}{NT}\right)$$

- **Discrete Fourier Transform**

The command may be applied on a single curve. By selecting this command the discrete Fourier transform of the plotted function is generated. To evaluate this transform consider a real function $g(t)$, which is zero for t<0. The function is sampled with frequency f_s=1/T, where T is the sample interval. We consider only part of this

function defined by window of length $T_0=NT$, where N is number of the sample points within the window (see figure below). In our case the sampling interval is the output simulation interval.



The time window

Discrete Fourier Transform (DFT) is defined by

$$G_d\left(\frac{n}{NT}\right)=\sum_{k=0}^{N-1} g\left(kT\right)e^{-j2\pi nk/N}\ ,\quad n=0,1,\cdots,N-1$$

The expression relates N samples of time and N samples of frequency by means of continuous Fourier transform. Development of this formula is given in [1].

A property of discrete FT is that its real part is symmetric about $n = N/2$. The values for $n > N/2$ correspond simply to the negative frequencies and cosine is an even function. Similarly, the sine is an odd function, and thus the imaginary part for $n > N/2$ is antisymmetric function of the frequency. Using [2] discrete FT is evaluated using Fast Fourier Transform (FFT). It is evaluated only for positive frequencies, i.e. for n in the range 0 to $N/2$-1. The values corresponding to the negative frequencies could be easily reconstructing from this one.

References

[1] E.Orhan Brigham, The Fast Fourier Transform and its Applications, Prentice Hall, Signal Processing Series, 1988.

[2] FFTPack documentation, http://www.netlib.org/fftpack.

Frequency Spectrum

The command also applies to a single curve. It generates amplitudes of the harmonics contained in the signal as function of their frequencies. To find them we start from the corresponding inverse DFT. It is given by [1] as

$$g\left(kT\right)=\frac{1}{N}\sum_{n=0}^{N-1} G_d\left(\frac{n}{NT}\right)e^{j2\pi nk/N}\ ,\quad k=0,1,\cdots,N-1$$

The amplitudes of the harmonics can be found as

$$|g(kT)| = \frac{1}{N} G_d(0) + \frac{1}{N} \sum_{n=1}^{N-1} \left| G_d\left(\frac{n}{NT}\right) \right|, \quad k = 0,1,\cdots,N-1$$

Because the modulus of DFT is an even function of the frequency, we can consider only the positive frequencies. From the last equation we obtain

$$|g(kT)| = \frac{1}{N} G_d(0) + \frac{2}{N} \sum_{n=1}^{N/2-1} \left| G_d\left(\frac{n}{NT}\right) \right|, \quad k = 0,1,\cdots,N-1$$

The last equation defines the *single sided amplitude-frequency spectrum* of the input signal evaluated using FFT of the plotted signal.

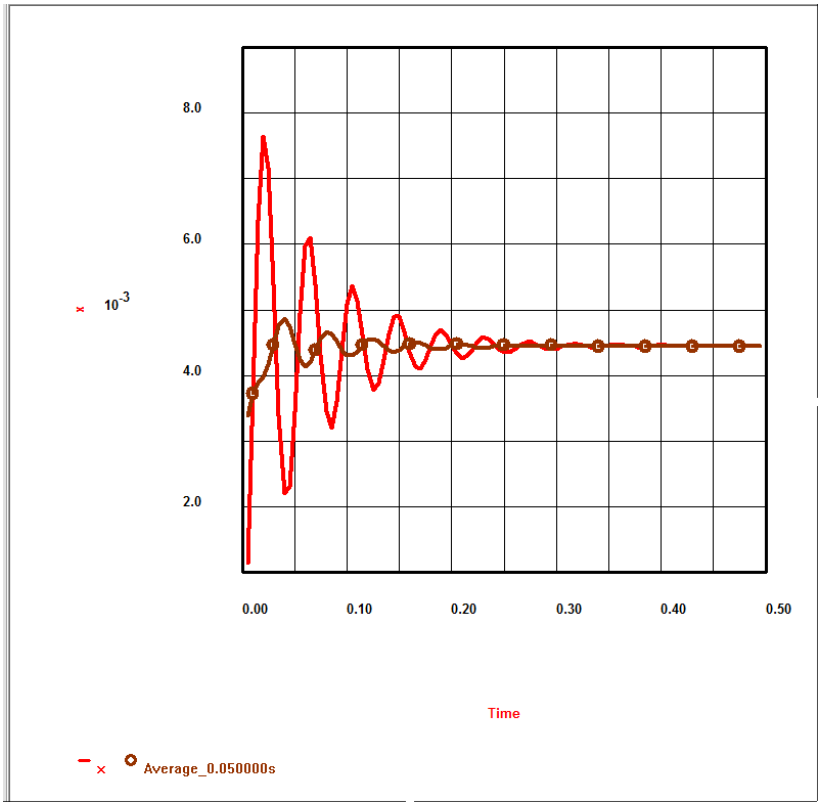
References

[1] E.Orhan Brigham, The Fast Fourier Transform and its Applications, Prentice Hall, Signal Processing Series, 1988.

[2] FFTPack documentation, http://www.netlib.org/fftpack.

Set Curve Marks

This command serves to annotate the curves if there are more such curves in the plot. Thus if right-click the deviation plot shown above the plot changes as shown below.

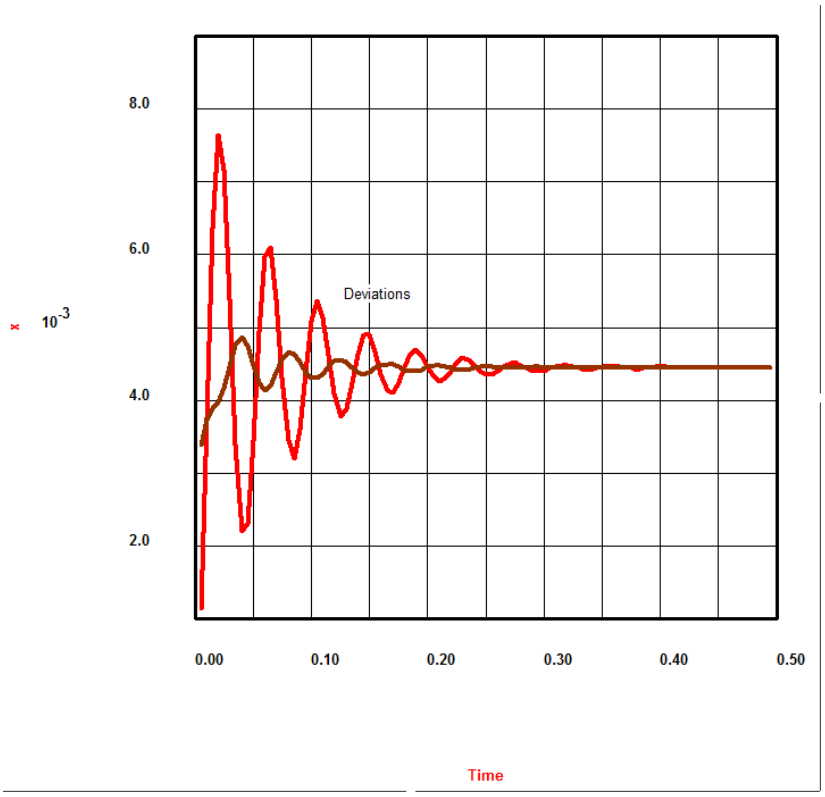


Annotated curves

It is possible to annotate up to 10 curves by assigning the different symbols. To remove the annotation symbols you may apply the same command again (right-click and select the command).

Insert Description

We may add the annotation also by adding a short description to the plot. Selecting this command we put the plot into text editing mode. The operation is very similar to that used when dragging and dropping the components during the model development. The cursor changes into a cross and we move it to a suitable place on the plot and click. Now the text caret appears and we my type the text. When we are finished we simple click somewhere out of the text.



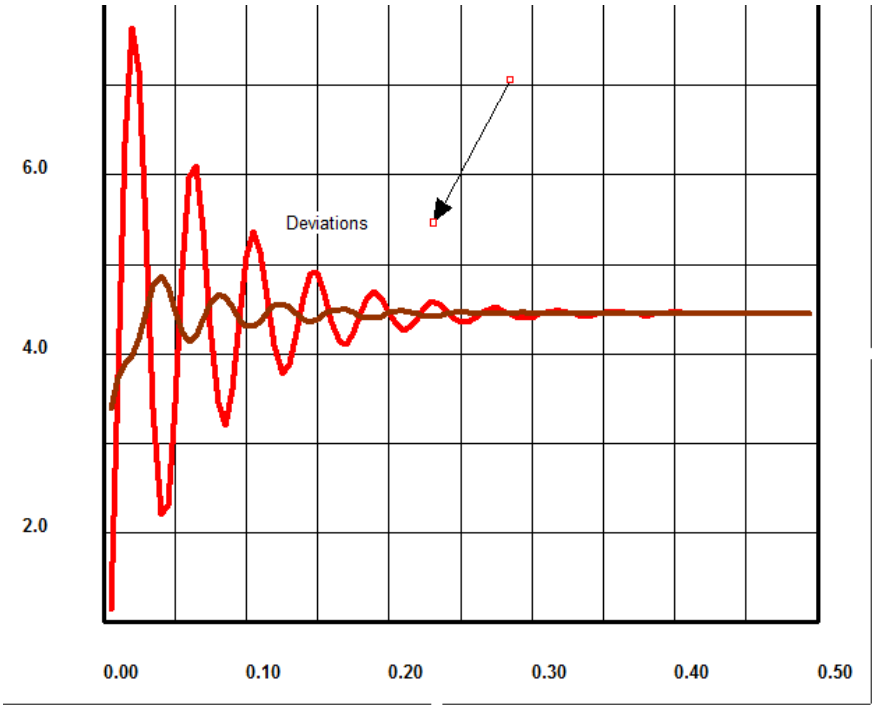
Edit Description

We can move the text by selecting it first, i.e. by clicking to the text and a red rectangle appears around the text. Now we may drag it and drop somewhere. We can also delete the selected text by pressing the **delete** key. To deselect the text we may click please outside of it. To change the text we right-click the selected text and when the text caret appears we start editing. When we finished with editing we click outside of the text.

Insert Arrows

We may add an arrow to the plot, e.g. to use it jointly with added text description. Hence we select this command and when the cursor changes into a cross we move it at some position in the plot, press it, drag it and release the mouse. A arrow appears. We may move the arrow in a different position and or change its shape by selecting it first.

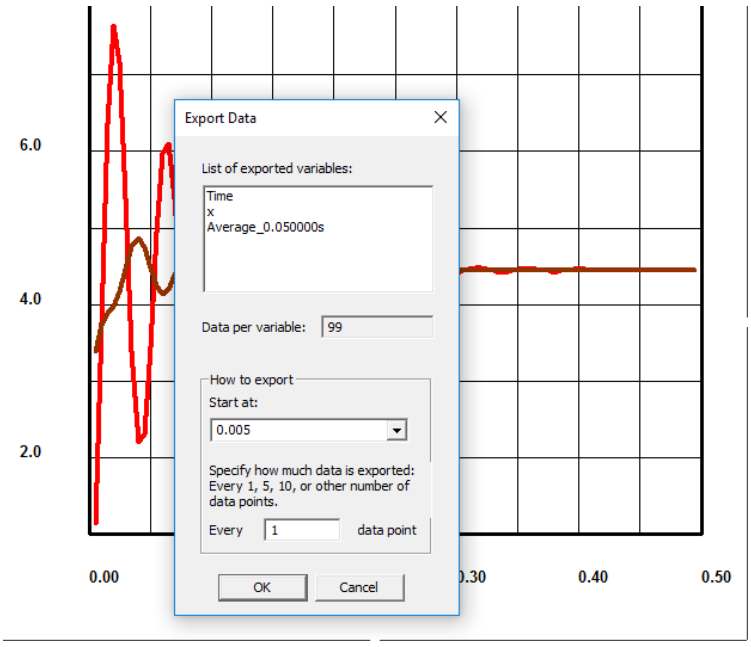
Now the small red squares appear denoting the tail and head of the arrow (see the Figure below). We can put the cursor on the arrow line and drag and drop it within the plot area. Similarly by dragging and dropping the arrow head or tail we change its form.



Selected arrow

- **Export Plot**

Using this command we may exporting the plotted data as a text file. Later it could be imported into MS Excel or similar tools for further processing. As we select this command a common Save As dialog opens which enables us to browse to a folder we wish to store the file to and define the file name. The file has default .txt extension. After we click OK button a new dialog opens as shown below.



Export data dialog

This dialog helps to define how data will be exported. At the top are the names of variables that will be written to the file. They serve to define the file header. Next goes the data for the variables. This is a relatively small data collection consisting of only 99 double values per variable, and we may write them all to the file. Sometimes, however, the size of data to write may be very large, e.g. 50 000 or more values per variable. In that case we may choose to start not at the beginning, or write e.g. every hundredth data points. When we click OK button the program writes the data to the file, close and and save the file to the selected folder. The text box below shows a part of written file named x-Smoothed.txt. The data are written in the format which is directly accepted by MS Excel.

Text Box: A part of x-Smoothed.txt file

```
Time;x;Average_0.050000s;
5.000000000000e-003;1.135603064001e-003;3.385973791415e-003;
1.000000000000e-002;3.761887676000e-003;3.710893577939e-003;
1.500000000000e-002;6.362472912229e-003;3.876736002603e-003;
2.000000000000e-002;7.645009673929e-003;3.987904128317e-003;
2.500000000000e-002;7.153983275804e-003;4.161503088586e-003;
3.000000000000e-002;5.381135048369e-003;4.467214510037e-003;
3.500000000000e-002;3.388466810110e-003;4.767864989637e-003;
4.000000000000e-002;2.199587811176e-003;4.865467594678e-003;
4.500000000000e-002;2.294980831097e-003;4.739668677449e-003;
5.000000000000e-002;3.448204502296e-003;4.484276227005e-003;
5.500000000000e-002;4.937230054710e-003;4.244610391290e-003;
6.000000000000e-002;5.973270277294e-003;4.137911439034e-003;
.....
```

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- X -