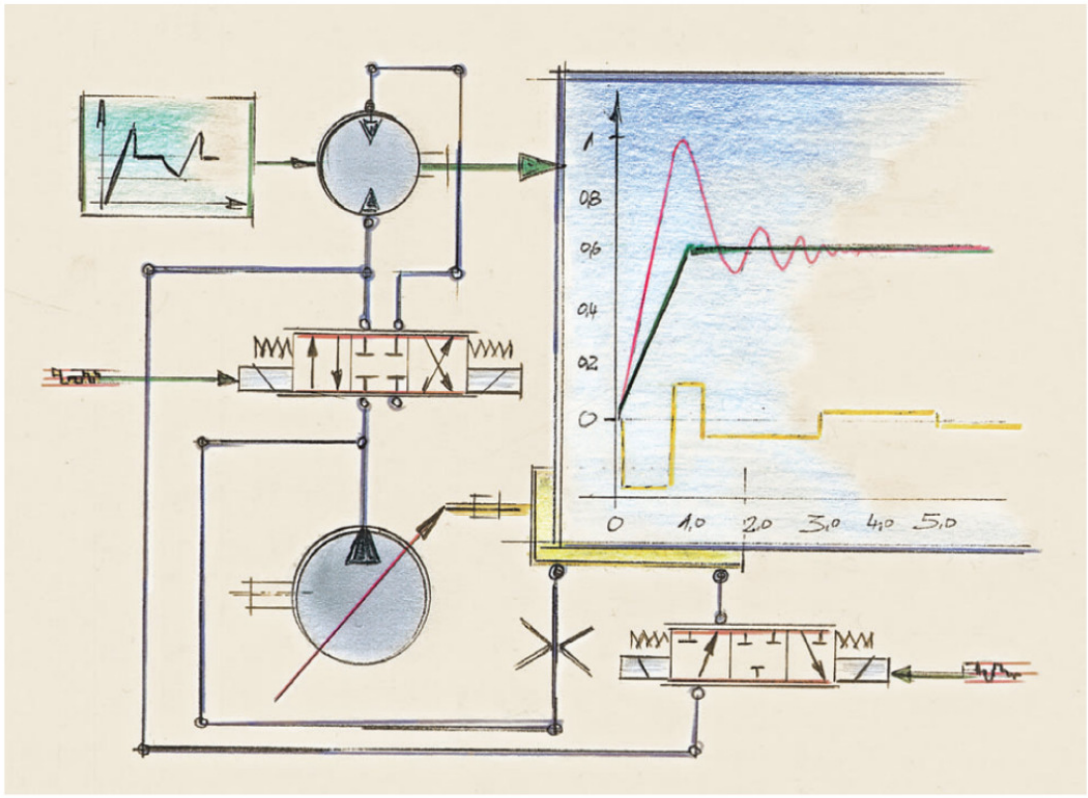
DSH*plus* 3.6

User Manual

Simulation, Analysis, and Optimisation



COMPETENCE IN FLUID POWER SYSTEMS SIMULATION

KOMPETENZ IN DER SIMULATION FLUIDTECHNISCHER SYSTEME

FLUIDON®
Gesellschaft für Fluidtechnik mbH

DSH*plus*

Copyright © 1995-2008 by

FLUIDON® Gesellschaft für Fluidtechnik mbH
Jülicher Straße 336
52070 Aachen
Tel. +49 (0) 241 - 9 60 92 60
Fax +49 (0) 241 - 9 60 92 62
www.fluidon.com
e-mail info@fluidon.com

No part of the manuals may be reproduced, duplicated or distributed in any form (printing, photoprint etc.) without permission of the FLUIDON GmbH.

The soft- and hardware names mentioned in this manual are in the most cases registered trademarks and are subject to the legal determination as such ones.

In the course of further development of the software, parts of this manual can lose their validity.

TABLE OF CONTENTS

1 INTRODUCTION 11

1 WHAT IS DSH*PLUS*? 11

2 STRUCTURE OF THE USER MANUAL 14

3 TYPOGRAPHICAL CONVENTIONS 15

4 MOUSE PROCEDURES 15

5 INDICATIONS TO THE OTHER MANUALS 16

2 INSTALLATION OF THE PROGRAM 17

1 SYSTEM REQUIREMENTS 17

2 INSTALL DSH*PLUS* 18

3 UNINSTALL DSH*PLUS* 19

3 BASIC PROGRAM FUNCTIONS 21

1 START DSH*PLUS* 21

2 CREATING A NEW PROJECT 25

3 DRAWING MODELS IN THE DESIGN MODE 25

3.1 The component library 26
3.2 Select components from the component library 29
3.3 Connection of components 31
3.4 Terminate model design 34

4 MODEL CREATION 34

5 SIMULATION MODE 36

5.1 Set parameters and starting values of the nodes and components 36
5.2 Set simulation parameters 40
5.3 Fluid options 41
5.4 Save parameters 42
5.5 Start and abortion of calculation 43
5.6 Control of simulation data during calculation 44
5.7 Save simulation results 48

6 CLOSE A PROJECT 48

7 EXIT DSH*PLUS* 49

4 THE INDIVIDUAL FUNCTIONS 51

1 PROJECT 51

1.1 New 52
1.2 New from Template 52
1.3 Open 53
1.4 Import 55
1.5 Close 57
1.6 Save 57
1.7 Save as 57
1.8 Info 58
1.9 Print 59
1.10 Exit 60

2 MODEL 60

2.1 Edit 61
2.2 Save 72
2.3 Stop edit 72
2.4 Library view 73
2.5 Graphic elements 73
2.6 Refresh 74
2.7 Select all 74
2.8 Copy 74
2.9 Paste 75
2.10 Undo 75
2.11 Redo 75
2.12 Create Model 75
2.13 Compile Model 78
2.14 Edit model source 79

3 PARAMETERS 80

3.1 Load 84
3.2 Save 87
3.3 Save as 88
3.4 Delete 90
3.5 Show 91
3.6 Show favourites 92
3.7 ASCII-Import 94
3.8 ASCII-Export 95

 © FLUIDON® GmbH 2008

3.9 Status 96
3.10 Protection 97
3.11 Look-up tables 99
3.12 Fluid options 107
3.13 Parameter selection 108

4 SIMULATION 109
4.1 Start 110
4.2 Start continuously 110
4.3 Stop 111
4.4 Parameters 111
4.5 Messages 115
4.6 Graphic 116
4.7 Current values 121
4.8 Animation 123
4.9 Save results 124
4.10 Load results 125
4.11 Create report 126
4.12 Create ePLAN report 127
4.13 Play animation 128
4.14 Final values as initials 128
4.15 Initial conditions 129
4.16 Selection of variables 132

5 MODULES 133
5.1 Variation of parameters 133
5.2 Batch simulation 136
5.3 Optimization 137
5.4 Waterfall plot 138
5.5 Frequency analysis 139
5.6 System identification 141
5.7 System eigenvalues 142
5.8 Editors 143
5.9 Export model 146
5.10 Create submodel 146
5.11 Import S-model 148
5.12 Four pole analysis 149

6 WINDOW 150
6.1 Miniature picture 151

7 EXTRAS 151
7.1 Options 152
7.2 Select licence file 161
8 HELP 161
8.1 Contents 162
8.2 Component library 162
8.3 About 162
8.4 Version info 162
8.5 Licence info 163
8.6 Hotline Support 164
9 CONTEXT MENUS 164
5 MODEL STRUCTURE 167
1 MATHEMATICAL PRINCIPLES 167
1.1 Mathematical description of technical systems 168
1.2 Calculation method of DSH*plus* 169
1.3 Numeric integration 173
2 DESCRIPTION OF THE MOST IMPORTANT FILES 178
2.1 Files of the projects 178
2.2 Files of the library 181
3 STRUCTURE OF THE SOURCE CODE OF A COMPONENT 183
3.1 Definition of category 184
3.2 Definition of parameters 184
3.3 Definition of auxiliary parameters 185
3.4 Definition of variables 186
3.5 Definition of auxiliary function 187
3.6 Definition of exit function 188
3.7 Definition of functions 189
3.8 End of component description 197
4 STANDARD PARAMETERS AND STANDARD STARTING VALUES 198
5 UNIT FILE 199
6 STRUCTURE OF THE SOURCE CODE OF A SIMULATION MODEL 199
6.1 Header of the model source code 201
6.2 Component descriptions in the model source code 201
6.3 Component definition 202

 © FLUIDON® GmbH 2008

6.4 Initialization 203
6.5 Exit function 204
6.6 Limit values 204
6.7 Reset update flag 205
6.8 Differential equations 205
6.9 Output function 208
6.10 Limitations 209
6.11 Data 209

6 PROGRAMMING OF NEW COMPONENTS 212

1 EXTENSION OF THE COMPONENT LIBRARY 212
1.1 Programming language of the component library 212
1.2 Programming of new components 212
1.3 Programming rules 215

2 DSH*PLUS* SPECIFIC DEFINITIONS AND FUNCTIONS 217
2.1 General 218
2.2 Deftyp2.h 220
2.3 dglstruc.h 224
2.4 dgltool.h 224
2.5 dglthermotool.h 240
2.6 dglpipes.h 241
2.7 pneutools.h 242
2.8 meldung.h 243

3 MODIFICATION OF THE SOURCE CODE OF THE MODEL 245
3.1 How to proceed when developing a component 245
3.2 Programming rules 247
3.3 Programming example 248

4 INTEGRATION OF EXTERNAL FUNCTIONS 265

APPENDIX A NODES 272

1 HYDRONODE 272

2 THERMOHYDRONODE 276

3 HYDROVLNODE 279

4 PNEUMATNODE 280

5 MECHTRANSNODE 281

6 MECHROTNODE 282

7 SIGNALNODE 283

8 DIGITALSIGNALNODE 284

APPENDIX B COMPONENT BLACKBOX 286

BLACKBOX: 286

APPENDIX C EXAMPLES OF SIMULATIONS 290

1 CYLINDER CONTROL 290

2 DIGITAL CYLINDER CONTROL 293

3 HYDROSTATIC DRIVE 295

4 HYDRO TRANSFORMER 297

5 DIESEL ENGINE DRIVEN PUMP 298

6 MECHANICAL DRIVE TRAIN 299

7 PRESSURIZATION AND VENTILATION OF A VOLUME 301

8 CYLINDER IN A CLOSED-LOOP CONTROL SYSTEM 303

9 MODELLING OF A PRESSURE REGULATING VALVE 305

APPENDIX D MISCELLANEOUS 310

1 CONFIGURATION OF COMPILER AND LINKER 310

INDEX 318

 © FLUIDON® GmbH 2008

1 Introduction

1 What is DSH*plus*?

DSH*plus* is a simulation software tool that has been developed to support a design engineer with virtual engineering functionality throughout the development of complex fluid technical systems.

To stay competitive new products have to be developed in the shortest possible time. For that reason, errors have to be determined in an early state of the development process, so that cost-consuming iteration loops in the product development are avoided.

With the help of a simulation-based design in the beginning of the product conception it is possible to get new products faster to the market - with a benefit in quality and at lower costs.

But the computer-based simulation not only shortens the expense for prototypes and testing cycles. With the usage of simulation the development process for the whole product life cycle can be optimized.

DSH*plus* is designed in a way to support the user comfortably from the first design of the system until the start-up of the system's operation.

Programs such as DSH*plus* contribute in a valuable way in securing one's own market position and open up new ways to enlarge it. The program therefore has been developed in the past few years into an indispensable working tool for system design.

What are the features of DSH*plus*?

As a graphic oriented program, DSH*plus* places the functional unit „component“ in the centre of the modelling concept when describing the system. Libraries which contain both hydraulic and pneumatic as well as electrical, control engineering and mechanical elements, are available to this effect. The circuit diagram analogy thus achieved enables a simple and convenient structure of the respective simulation model and

guarantees that the user immediately feels familiar with the simulation tool DSH*plus*.

A prerequisite for accepting simulation in the industrial practice is the simple parameterization of components. Catalogue data of the manufacturers therefore form the basis for modelling the components in DSH*plus*. The open mathematical description of the system as C++ Source Code enables the user in addition to include his own approaches into the component description and makes it possible to use the program in conjunction with other simulation tools.

Which possibilities does DSH*plus* offer?

With DSH*plus*, the system developer disposes of a design tool which offers planning safety already in an early project stage. The circuit diagrams and simulation results of the concept draft can also be used for communication between the project partners.

DSH*plus* enables to develop alternative system solutions and circuit concepts and to comfortably evaluate them. The result is a system design which is adapted in an optimum way to the technical problem. The reduction of the prototype expenditure thus achieved is also responsible for a considerable saving of time in planning or revision of the systems.

Another result of the dynamic system evaluation with DSH*plus* is the optimum component selection when converting the system solution developed. The use of DSH*plus* not only leads to an optimization of the costs for the manufacturer but also at the same time guarantees the user that the plant is optimized with regard to operating costs.

DSH*plus* simulation models can even be used as reference during commissioning and for training of the plant personal as the experience of well-versed users has shown.

 © FLUIDON® GmbH 2008

Working with DSH*plus*

The manual on hand will provide the user with a practical introduction in DSH*plus* and show how the program can be used for solving fluid-technical problems.

The system developer will appreciate very quickly the efficient and easy-to-use environment which enables him to carry out the numeric calculation of the system interactively and to visualize it.

With all these advantages, however, one may never forget that behind the circuit diagram displayed on the screen, a complex system of differential equation is hidden for which the individual program functions of DSH*plus* are only a comfortable handling tool.

The quality of the calculations made with DSH*plus* thus largely depends on a useful model structure for which the system developer is responsible. In order to achieve a stable numerical calculation, realistic data have to be available or be worked out for the parameterization of the components.

2 Structure of the user manual

Three types of manuals are supplied with DSH*plus*, a user's manual, component manuals, and a module manual.

The user's manual is made up of the following chapters:

Chapter 1 is this introduction chapter.

Chapter 2 carries out hardware and software requirements for the operation of DSH*plus* and explains the installation of DSH*plus*.

Chapter 3 provides a brief introduction to DSH*plus*. It explains the program elements which have to be carried out to run a simulation.

Chapter 4 complements this introduction with detailed information on all program functions. Their knowledge helps to use the comfortable possibilities of DSH*plus* and to carry out optimum simulation projects according to the individual requirements of the users.

Chapter 5 describes the technical and mathematical principles of DSH*plus*.

Chapter 6 describes the programming of new components. The functions presented here go beyond the simple use of the program. Their use requires programming knowledge from the user.

Appendix A contains an explanation of the nodes by which the individual components are linked.

Appendix B contains a description of the component BLACKBOX

Appendix C contains simulation examples

Appendix D installation of the compiler, etc..

 © FLUIDON® GmbH 2008

Index enables the user to use the manual as reference book.

At the end of each individual chapter, numerous reference is made to thematically linked chapters. These should facilitate the user to find a solution to individual cross-chapter questions.

3 Typographical conventions

In order to facilitate reading the test, significance and categories which repeat are optically marked. The following fonts are used in this manual:

MENUS, **MENU POINTS** and **KEYS** in the dialogs are stressed by bold type and small capitals.

Names of **Dialogs**, **Fields** and **Lists** of the individual dialogs are marked in bold type.

Source text is keyed in grey, indented and marked with its own type matter and type size.

```
This is an example for formatting a source code in the manual
```

4 Mouse procedures

In DSH*plus* the left mouse key is intended as primary key, the right mouse key as secondary key. The descriptions in the manual always refer to this designated function. Please mind that the primary and secondary function will be reversed if you have adjusted your mouse by hand, e. g. for left-handed operation.

The left mouse key offers you the following possibilities:

Showing: place the mouse pointer on a certain element of the screen

Clicking: press the mouse key briefly and release it immediately again without moving the mouse

Drawing keep the left mouse key pressed while shifting the mouse pointer

Marking: click on a single object with the left mouse key or draw the mouse over several objects

The right mouse key offers you the following possibilities

Clicking: press the right mouse key briefly and the so-called **CONTEXT MENU** is opened. For further information about context menus please see chapter 4, 8 Context menus.

5 Indications to the other manuals

The second manual type – the component manual (consisting of technology specific parts) – contains the documentation of all components and nodes that are available in DSH*plus*.

In the third manual – the module manual – the analysis functions and the program interfaces that are available in DSH*plus* are documented and illustrated by examples of application.

 © FLUIDON® GmbH 2008

2 Installation of the program

1 System requirements

Operation of DSH*plus* demands specific minimum requirements to the computer hardware resources. This minimum configuration can be enlarged by extensions which facilitate the work with DSH*plus* especially in interaction with other Windows applications. Some of these hardware requirements are determined by the platform used - Windows NT/2000, XP or Vista -, others are specific for DSH*plus.* The minimum required computer configuration recommended for an efficient and universal use is listed in the following.

Minimum computer configuration:

- Windows NT/2000 SP3
- Resolution 1024*768 pixels
- 150 MB free hard drive capacity

DSH*plus* should in any case be operated with a graphic oriented input device. The usual devices supported by Windows such as mouse, track ball or touch pad can be used.

All the printers and plotters supported by Windows can be used as output devices.

Use of DSH*plus* in a network

- DSH*plus* is not especially designed for multi-user deployment in a network environment. Although it also works via a network in most of the cases. However, not more than one user at a time should access a model directory. The appendix explains how DSH*plus* has to be configured for use in a network.

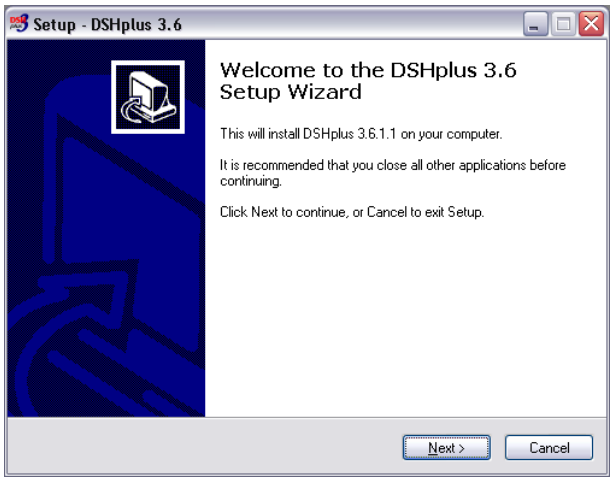
2 Install DSH*plus*

Preparations

- Make sure that you have administrator rights to install the program.
- Make sure that there is at least 150 MB free hard drive capacity is available on the disk on which you want to install DSH*plus*.

Installation

- Insert the DSH*plus*-CD into the CD drive.
- This application starts automatically. If there is no Autostart function please run "setup.exe" from the CD-ROM.
- The DSH*plus* welcoming screen is displayed.



Welcoming screen of DSHplus

- Follow the instructions of the installation program. An installation guide is available on the program CD

© FLUIDON® GmbH 2008

3 Uninstall DSH*plus*

In order to uninstall DSH*plus* completely, open the DSH*plus* folder from the start menu and run the DSH*plus* uninstall entry. You also can use the software de-installation function of the operating system.

 © FLUIDON® GmbH 2008

3 Basic program functions

This chapter is an introduction to the handling of DSH*plus*. In this chapter, the functions, which are absolutely necessary to perform a simulation, are explained.

Among others it is, creating and managing of projects, the graphic structure of the simulation model, the generation of an operational model, simulation calculation, and evaluation of the results.

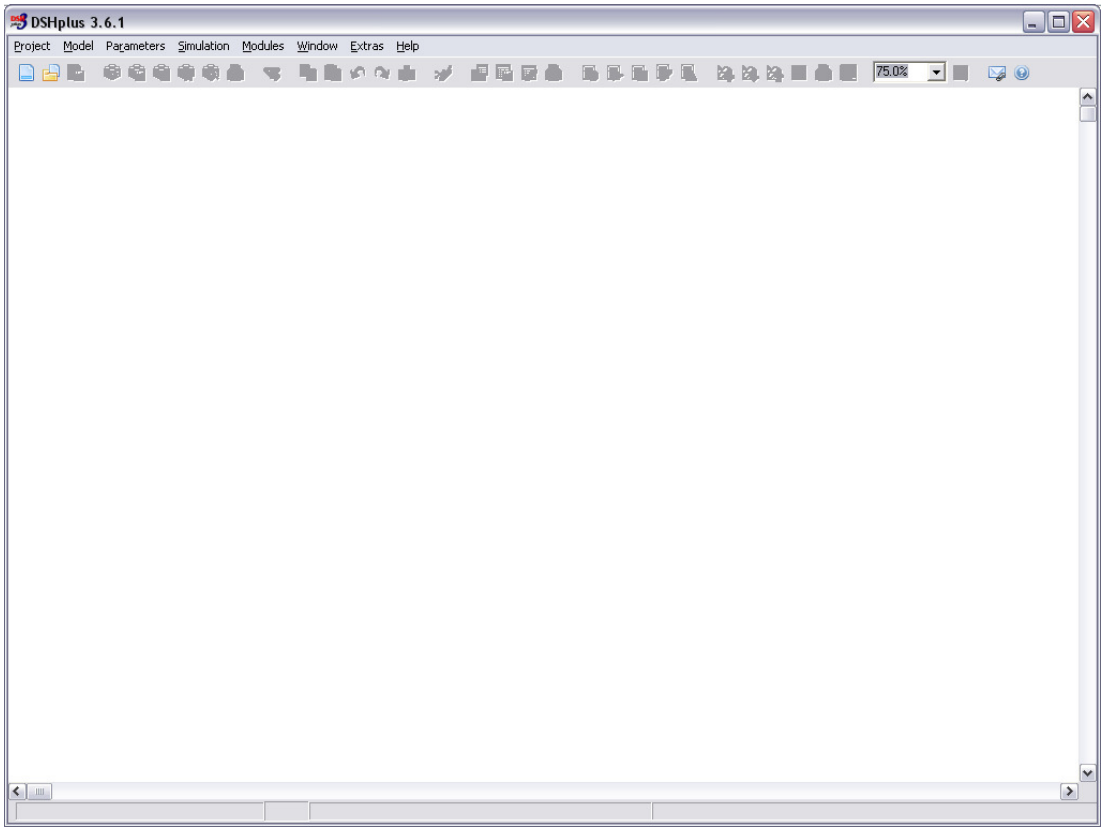
The sub-chapters are arranged in the order in which the steps described are to be used. Thus, users can take the structure of the chapter as a guideline for working with DSH*plus*.

1 Start DSH*plus*

Depending on the selected installation options, DSH*plus* can be started in the following ways.

- DSH*plus* is started with the **DESKTOP ICON**
- DSH*plus* is started with the **QUICK LAUNCH ICON** in the **START MENU.**
- DSH*plus* is started form the DSH*plus* program group, that is located under **PROGRAMS** in the **START MENU**.

During the start of DSH*plus* a splash screen shows the type of licence that is invoked.



Surface of DSHplus

Title bar

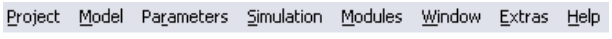
The title bar indicates besides the program name and the version the name of the project if a project is open.



Title bar

Menu bar

The menu bar contains the call of all functions with which DSH*plus* is operated. Operation and design is subject to the Windows guidelines.

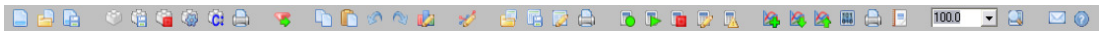


Menu bar

 © FLUIDON® GmbH 2008

Toolbar

The toolbar is situated below the main menu line on the upper edge of the DSH*plus* main window. The control spaces in this line symbolize frequently needed menu commands. It is possible to call these functions by simply clicking on the mouse without opening the menus themselves.



Toolbar

When the mouse pointer points onto a control space of a function line, a small quick info window with the name of the corresponding function is displayed after a short time.

The significance of the individual control space is explained in the appendix.

Status bar

The status line on the lower edge of the DSH*plus* window takes over information tasks.



Status bar

- Indications to the menu function you have selected are displayed in the left display space.

- During simulation, the progress of calculation is displayed in the control window. When the progress bar occupies the entire field, simulation is finished.

- The field on the right next to the progress bar displays the current simulation step size and remaining term.

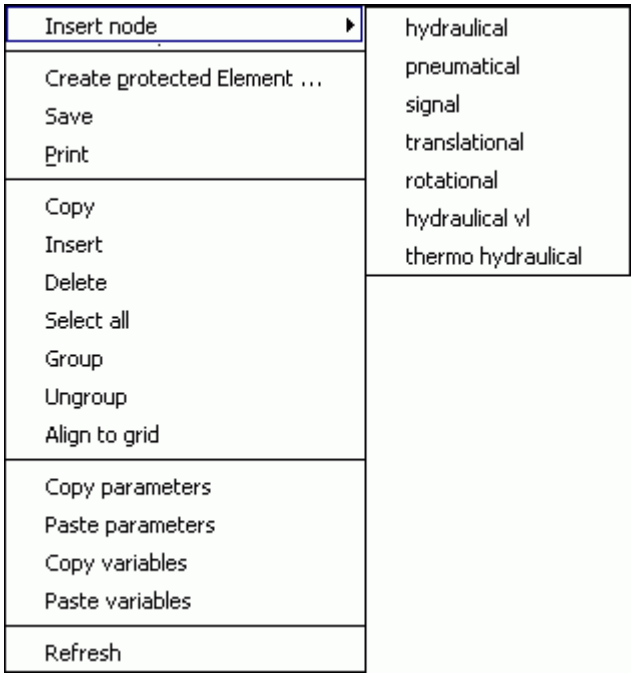
- The field on the right displays the name of the currently loaded parameter set.

Drawing area

The space between the function line and the status line is the drawing area. It is used for designing and processing the model.

Context menus

The main menu is complemented by context menus which contain commands available only for a specific application or a window and which only relate to this specific application. Context menus can be opened by a mouse-click with the right mouse key on the object or the area that is to be edited.



Context menu

 © FLUIDON® GmbH 2008

2 Creating a new project

Within the scope of a project, the model is designed using the component libraries, the component parameters are adjusted, and the dynamic behaviour of the model is calculated by DSH*plus*.

At first a new project has to be created. There are two possibilities to do this:

- A empty project is created by the menu point New in the Project menu. That a new project was created is visible by the fact, that the component library is open. The program is in the design mode.

- By means of the menu point New from Template a project template is used to create a new project by means of. To edit the model, the library has to be opened manually.

For further information on:	Please see:
Creating of projects	Chapter 4, 1.1 New
	Chapter 4, 1.2 New from Template
Opening of projects	Chapter 4, 1.3 Open
Store projects under different name	Chapter 4, 1.7 Save as
Importing and exporting of projects	Chapter 4, 1.4 Import
Files of a project	Chapter 5 Model structure, 2.1 Files of the projects

3 Drawing models in the design mode

After a project has been created, the program is in the input mode. This mode can be seen by the fact that the component library is open.

In the input mode, the simulation models are graphically designed on the drawing area. To this effect, components from the component library are placed onto the drawing area and linked.

The following chapters describe how to design models.

For further information on:	Please see:
Designing and processing of models	Chapter 4, 2 Model

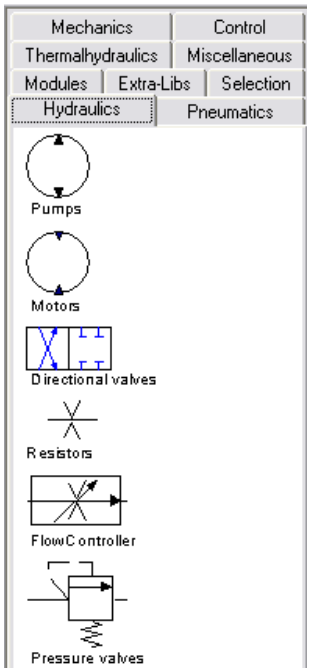
3.1 The component library

In the component library, depending on the licence version DSH*plus* provides hydraulic, pneumatic, mechanic, and thermal-hydraulic as well as components of the automatic control engineering which simulate the technical behaviour of real components by means of mathematical functions.

The component library consists of several categories which on their part are subdivided in different component assemblies. In these component assemblies, either component subassemblies or the individual components can be found.

The category names are displayed on the upper edge of the component library. The component assemblies, subassemblies and components are displayed in the library window as symbols.

 © FLUIDON® GmbH 2008



Component library (full license version)
View hydraulic category

In the following the most important categories, component assemblies and subassemblies are listed for example:

Categories	Component assemblies	Component subassemblies
Hydraulics	Pumps	
	Motors	Rotational motors Linear motors
	Directional valves	2-x-valves 3-x-valves 4-x-valves Miscellaneous valves
	Resistors	
	Flow controller	
	Pressure valves	
	Check valves	
	Accumulators	
	Lines and reservoirs	Automotive lines

Categories	Component assemblies	Component subassemblies
	Hydro2signal	
	Fluidsensor	
Pneumatics	Motors	Linear motors
		Rotational motors
	Directional valves	2-x-valves 3-x-valves 4-x-valves 5-x-valves
	Check valves	
	Pressure valves	
	Resistors	
	PneuAccumulator	
	Pneusensor	
	Pneu2Signal	
	pT2pneu	
Mechanics	Masses	
	Spring-damper components	
	Drive technology	
	Miscellaneous	
ThermalHydraulics		
Control	Sources	FunctionGenerator
		Constant
		OnlineDataImport
		TrackCurve345
		Timebase
	Mathematical functions	

© FLUIDON® GmbH 2008

Categories	Component assemblies	Component subassemblies
	Functions and tables	
	Transfer functions	
	Time discrete transfer functions	
	Nonlinear functions	
	Extras	
	Digital signals	
	Logic elements	
Extra-Libs		
Miscellaneous	Nodes	
	Steering components	
	Miscellaneous	

For further information on:	Please see:
Components	Component manual
Mathematical principles of components	Chapter 5, 1 Mathematical principles

3.2 Select components from the component library

In order to design a simulation model graphically the required components have to be selected from the component library and to be placed on the drawing area on the left next to the library.

How to proceed:

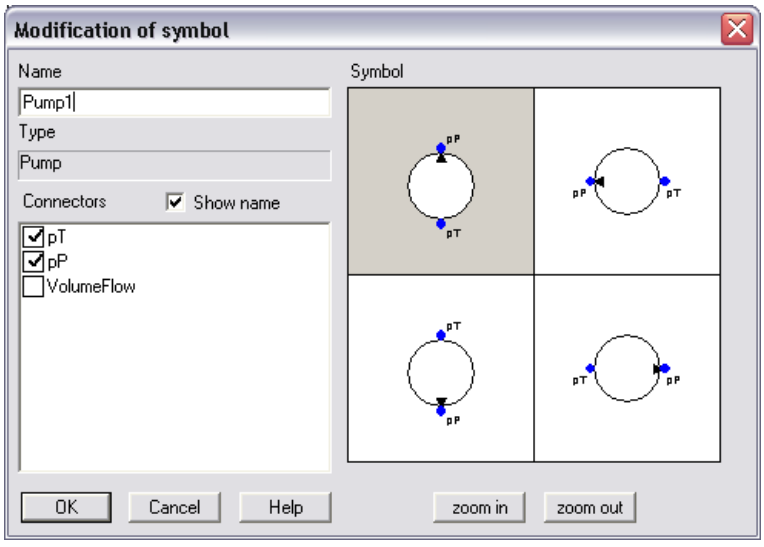
1. If you click on the category name in the headline of the component library, the corresponding component assemblies are displayed in the library window.

2. Click onto the symbols of the component assemblies and subassemblies until the assembly with the component required is opened.

3. Select the component by clicking it on. The mouse pointer transfers into a cross.

4. Click onto the space in the drawing area on which the component is to be placed.

5. When you have clicked on the drawing area the dialog **MODIFICATION OF SYMBOL** is opened. A name which consists of the type designation and a number is registered in the field name automatically. You are able to maintain this name or to allocate a definite name of your own choice. This name must not be allocated to another component in the same project.



Component dialog

6. If you click on **OK**, the selected component is displayed on the drawing area.

 © FLUIDON® GmbH 2008

For any further component which is to become part of the model, repeat steps one to six.

If you click on another category after selecting a component assembly, but you have not selected a new component assembly yet, it is possible to open again the last selected component assembly by clicking on **SELECTION** in the headline of the component library.

For further information on:	Please see:
Assignment and connections of components	Chapter 4, 2.1.3 Adapt component alignment and component connections

3.3 Connection of components

The components now placed on the drawing area have different connections. The same connections of different components can be linked by lines.

Survey of component connections and their corresponding nodes:

- Hydraulic connections are marked as small blue dots. In general, they are linked by hydraulic nodes (HYDRAULICAL).
- Signal connections are marked as small black arrows. They are linked by signal nodes (SIGNAL).
- Pneumatic connections are marked as small dark blue dots. They are linked by pneumatic nodes (PNEUMATICAL).
- Mechanic connections for translatory movements are marked as dark gray arrows. They are linked by mechanical-translatory nodes (TRANSLATIONAL).

- Mechanic connections for rotation movements are marked as light grey arrows. They are linked by mechanic-rotatory nodes (ROTATIONAL).

- Thermal-hydraulic connections are marked as dark red points.

- Hydraulic connections without volume are marked as lilac points.

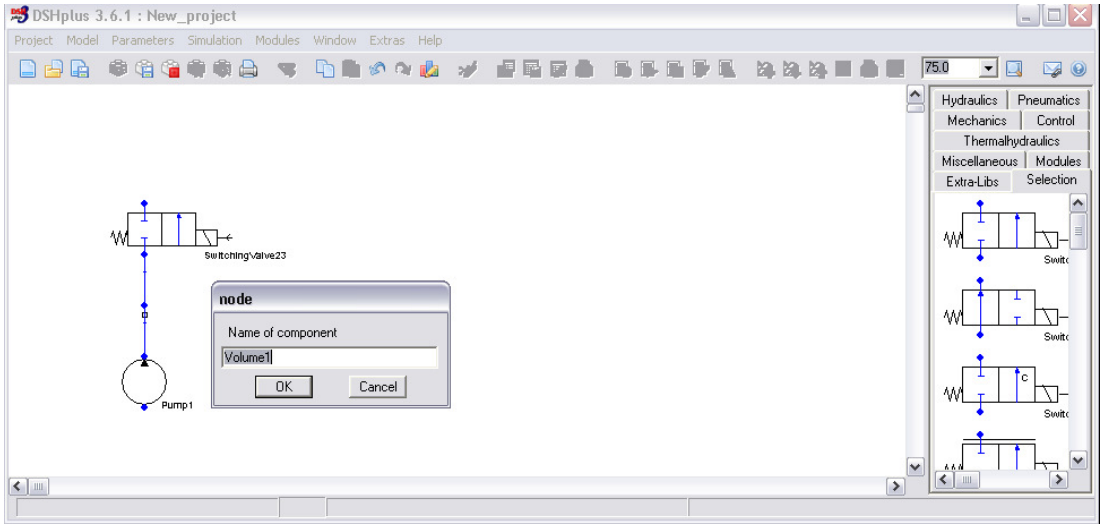
- Signal-digital connections are marked as dark green points.

In order to link the components via the respective nodes, proceed as follows:

1. Use the left mouse key to click on a component connection and keep the key pressed.

2. Draw the mouse pointer to a component connection of the same type of another component.

   Please see that only outputs can be connected to inputs and not outputs to outputs or inputs to inputs. DSH*plus* checks the connection logic and avoids wrong connections.



*DSHplus in the input mode with an active **node** dialog*

3. If you release the mouse pointer exactly on the second component connection, the suitable node is automatically created and the **node** dialog is displayed.

 © FLUIDON® GmbH 2008

4. In this dialog you have to allocate a definite name to the node or to confirm the proposed name.

5. The dialog is closed with **OK** and the name allocated is displayed on the drawing area below the connection node.

Repeat steps one to five for all components you want to connect.

Note:

- It is possible to abort the connection process by ESC.
- The model can be stored while it is designed.
- You can use the undo function in order to undo steps of the model design.
- If you insert names for the nodes by hand, a unit and a value range according to the pre-settings for the initial letters are allocated to these nodes. The pre-settings can be done under **OPTIONS** in the menu **EXTRAS** on the file-card **Graphics Options** in the area **Special**

For further information on:	Please see:
Deletion of connection nodes	Chapter 4, 2.1.8 Deleting of components and links
Extension of connections	Chapter 4, 2.1.2.1 Connection links
Connect several components with a component connection	Chapter 4, 2.1.2.2 Link of several components
Component connections	Chapter 4, 2.1.3 Adapt component alignment and component connections

3.4 Terminate model design

After the simulation circuit has been designed graphically the input mode has to be closed in order to generate the simulation model.

How to proceed:

1. Click on **STOP EDIT** in the **MODEL** menu.

2. Click on **YES** in the **Save changes** dialog.

The input mode is closed now. In order to be able to simulate with the model, the model creation of the source code has to be carried out, so that the graphic model can be transferred into a mathematical model. This is done automatically if the standard program configuration has not been changed.

If a new model wasn't saved before, the **Projects save as** Dialog will open to do so.

Note:

- The **YES** key in the **Save changes** dialog means that the current model configuration is stored. **NO** means that the changes are rejected after they have been last stored and the old circuit diagram is loaded again.

4 Model creation

The following steps are only required, if the standard program configuration has been changed on the **Other options** page of the **Options** dialog.

How to proceed:

1. Open the dialog **Status model build up** in the menu **MODEL** with **CREATE MODEL**.

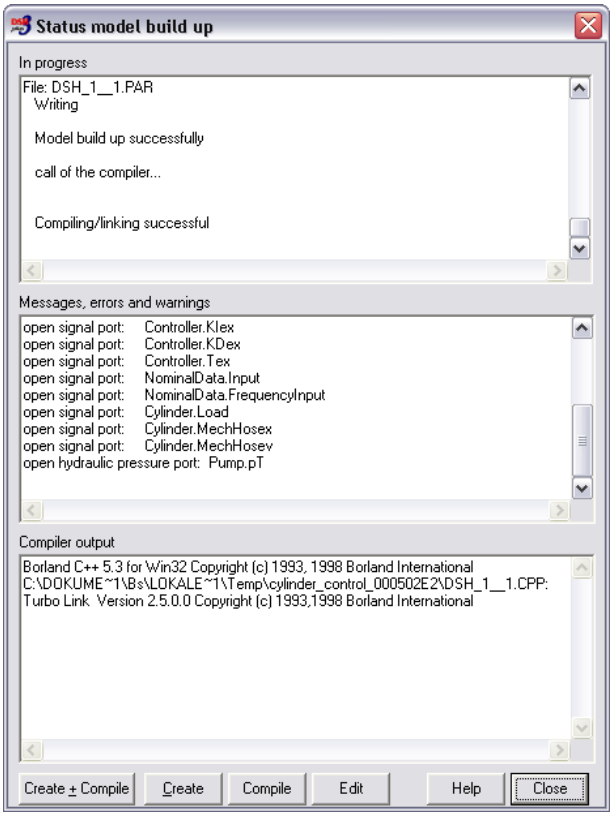
 © FLUIDON® GmbH 2008

2. By clicking on **Create + Compile** in the dialog, the source code of the model is created and compiled.

   The lists indicate the process of creating and compiling.

3. When creating and compiling is terminated, click on **CLOSE**.

Note:



Dialog for status of model formation

The files required for the calculation of the system behaviour are now created. The program is in the simulation mode.

For further information on: Please see:

Creating a model Chapter 4, 2.3 Stop edit

5 Simulation mode

After model formation, the program is in the simulation mode.

In this mode, the following steps are to be carried out:

1. Input of component parameters.
2. Input of starting values or variables.
3. Input of the calculation period and the simulation step size.
4. Calculation of the dynamic behavior of the model.
5. Evaluation of the simulation results.
6. Repeating steps one to five.

The following chapters describe how to carry out these individual steps.

5.1 Set parameters and starting values of the nodes and components

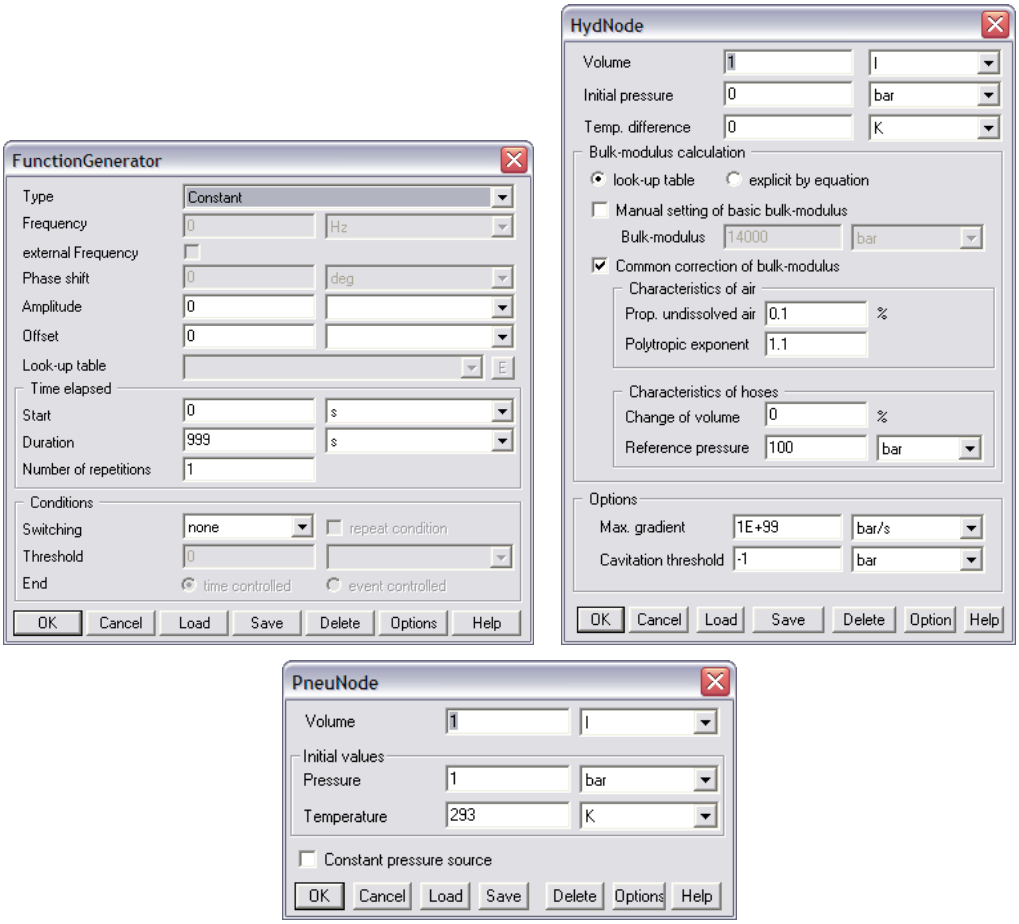
The parameters and variables of the components and nodes are set in the respective dialogs. In these dialogs, the parameters and variables are preset with standard values which are changed according to individual requirements.

The dialogs are opened by double clicking on the respective component. When the desired changes have been made they are confirmed by clicking the **OK** button. The dialog closes again. The names of the components whose parameters have not been edited yet, are coloured violet. After the processing of the parameter set the name of the component is displayed in black.

 © FLUIDON® GmbH 2008

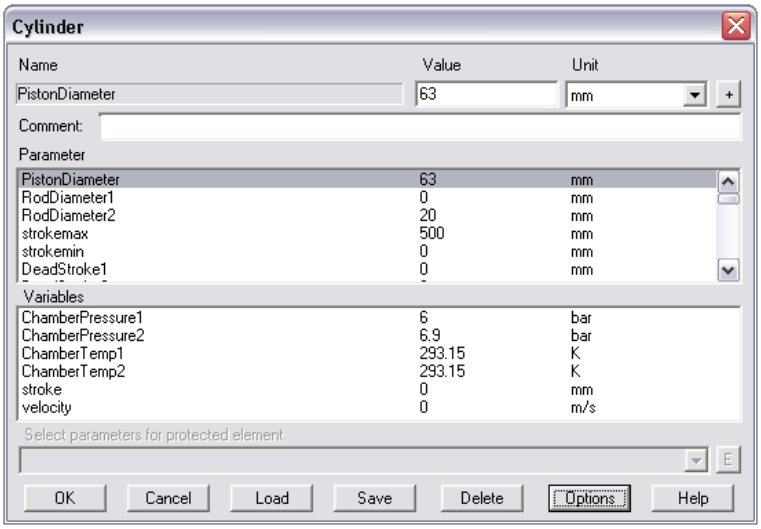
Function generators, hydraulic and pneumatic nodes have their own dialog types in which every single parameter has its own input line for putting in a value and the unit, if necessary. For further details on the parameters, please see the component manual.

Settings of the other nodes and components are carried out in standardized dialog types. In order to change variables and parameters, proceed as follows:



On the left the parameterization dialog of a function generator, on the right the dialog of a hydraulic node, below the dialog of a pneumatic node

1. Open the dialog by double clicking on the component.



Parameterization dialog of a cylinder

2. If you click on an entry in the list **Parameter** or the list **Variables**, this entry is displayed in the input field.

   The parameters indicate the component specific characteristic values which remain constant during simulation. The starting values of the variables at the beginning of a simulation are shown under Variables.

3. If you want to change the unit of a parameter, click on the arrow of the drop down list with the units and select the unit desired. This list mentions all units belonging to the same unit group as the one originally mentioned.

   If the desired unit is not in the list, click on the **+** button. All the units made available by DSH*plus* are now displayed in the drop down list. The number in the **Value** field is converted to the suitable unit.

4. Put in the desired numerical value in the input field **Value**.

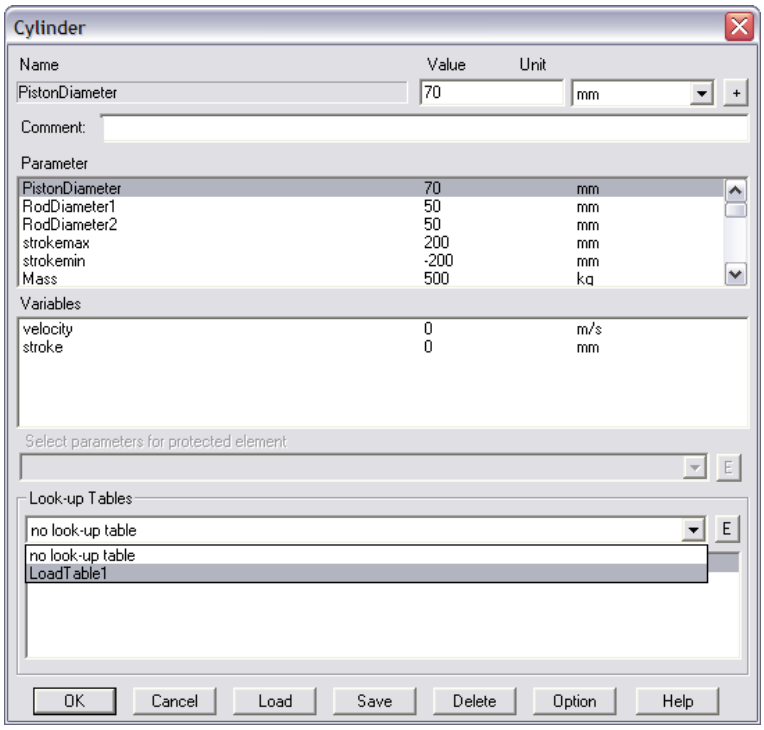
5. If you put in the value first and then adjust the unit, the value is adapted to the new unit.

6. Repeat steps one to four until all parameters and variables have the desired unit and the desired value.

© FLUIDON® GmbH 2008

7. If you click on **OK**, the parameterization dialog is closed. The settings are used in the following simulations.

These steps also have to be carried out for setting the other components.



Parameterization dialog of a component with look-up tables

If the component contains look-up tables, an extended dialog is used. The parameterization dialog contains an additional list with look-up tables which can be allocated to the look-up tables parameters.

The selected look-up tables in the look-up tables editor is opened by the **E** key.

For further information on:	Please see:
Individual parameters	Component manual
Save component parameters	Chapter 4, 3.2 Save

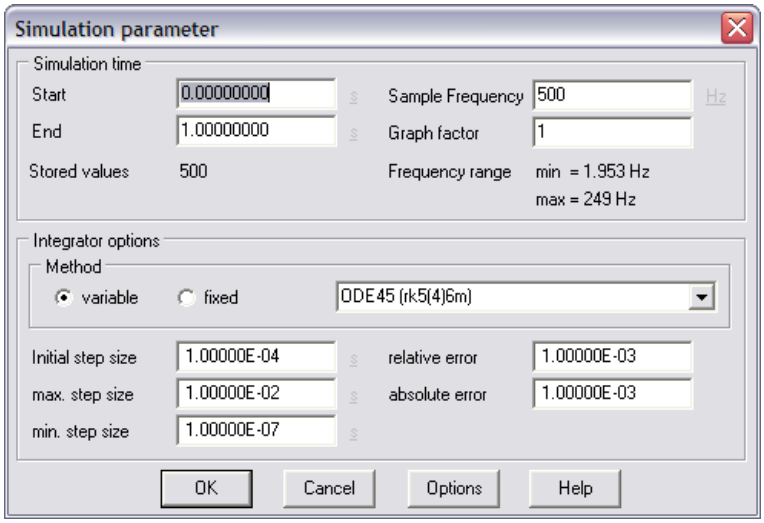
For further information on:	Please see:
Load component parameters	Chapter 4, 3.1 Load
Look-up tables editor	Chapter 4, 3.11 Look-up tables

5.2 Set simulation parameters

The numeric limiting conditions of the simulation calculation are set in the dialog **Parameters of Simulation** .

To do this, proceed as follows:

1. In order to open the **Parameters of Simulation** dialog, click on **PARAMETERS** in the **SIMULATION** menu.



Dialog for simulation parameters

2. Make the desired settings.

3. Click on **OK** in order to take over the settings for the simulation.

Now, you have determined the numeric limiting conditions of the simulation. The values adjusted are used when calculating the dynamic behavior of the model and are also saved in the parameter set.

© FLUIDON® GmbH 2008

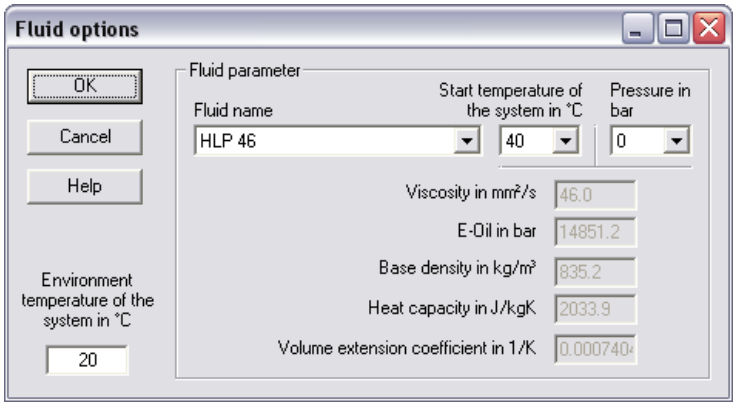
For further information on:	Please see:
the individual parameters	Chapter 4, 4.4 Parameters
Save simulation parameters	Chapter 4, 3.2 Save
Load simulation parameters	Chapter 4, 3.1 Load

5.3 Fluid options

In the dialog **Fluid Options** the specifications of the employed oil type and the environment temperature can be set.

For setting the fluid options please proceed as follows:

1. Select the point **FLUID OPTIONS** in the menu **PARAMETERS**.



Dialog for Fluid options

2. Make the desired adjustments for the oil specification and the environment temperature.

3. Click on **OK.**

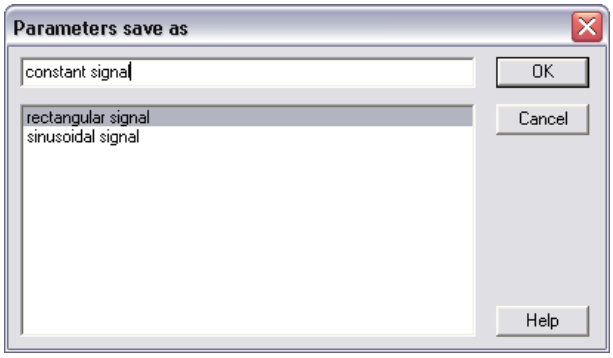
Now you have made the settings for oil type and environment temperature.

For further information on:	Please see:
Fluid Options	Chapter 4, 3.12 Fluid options
Save Simulation Parameters	Chapter 4, 3.2 Save
Load Simulation Parameters	Chapter 4, 3.1 Load

5.4 Save parameters

Component parameters, variables and simulation parameters can be saved together and provided with an identifiable comment. To do this, proceed as follows:

1. Click on **SAVE AS** in the **PARAMETER** menu in order to open the dialog **Parameters save as**.



*Dialog **Parameters save as***

2. Put in a comment in the input line or select a comment already available from the list below the input line and click on **OK**.

3. If you have entered a new comment, you have thus created a new parameter set and saving is now terminated. If you use a comment already available, a dialog with the note **The set of parameters exists** opens. In this dialog, it is possible to choose between **OVERWRITE** and **NEW**. With **OVERWRITE** the old parameter set to which the comment has already been allocated, is deleted and replaced by the new set. **NEW** means that the new parameter set has the same

 © FLUIDON® GmbH 2008

comment as the already existing one. In all parameter set lists, the comment can then be found several times.

For further information on:	Please see:
Save parameter set	Chapter 4, 3.2 Save
Load parameter set	Chapter 4, 3.1 Load
Delete parameter set	Chapter 4, 3.4 Delete

5.5 Start and abortion of calculation

Calculation of the dynamic behaviour of the model is started with **START** in the **SIMULATION** menu.

Calculation of the model is based on the adjusted parameters. Calculation of the variables begins with the starting values adjusted. The calculation period and the numeric limiting conditions of the calculation depend on the settings of the simulation parameters.

The blue progress bar in the status line on the lower edge of the program window shows the progress of the simulation. The simulation is interrupted automatically if the setting of one or several components is defective. In that case the name of the defective components is marked in red. In addition the dialog **Messages** is opened. In this dialog you find information about the question which defect occurred in which component.

If a warning for a certain component occurs the name of this component turns violet. The simulation is not interrupted by warnings. The dialog **Messages** is not opened automatically. Via the menu **SIMULATION** and the sub-point **MESSAGES** the dialog can be opened in order to get detailed information about the warning.

With **STOP** in the **SIMULATION** menu, calculation can be aborted before the adjusted simulation time has elapsed.

5.6 Control of simulation data during calculation

DSH*plus* enables to see the data on the model dynamic behaviour already during calculation. The data can be represented in a graphical manner as well as in a numerical one.

This gives the possibility to judge the development of variable values at an early stage and to abort simulation, if necessary.

5.6.1 Graphical control

By means of the **Graphics** window, the values of the variables can be put out graphically during calculation over the simulation time.

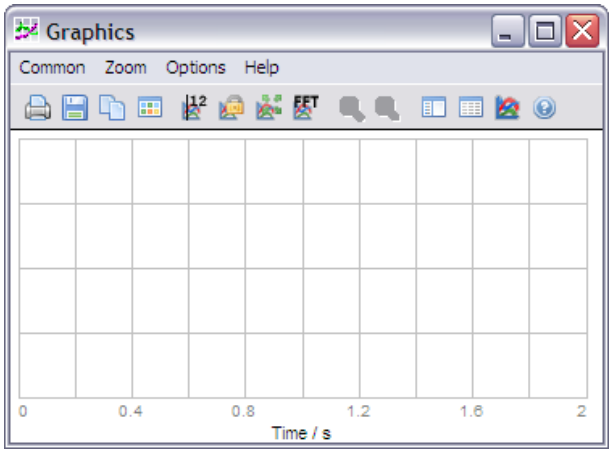


Simulation graphics

Proceed as follows in order to call and to configure a simulation graphics:

1. Open a new graphic window by selecting **GRAPHIC->NEW WINDOW** in the **SIMULATION** menu.

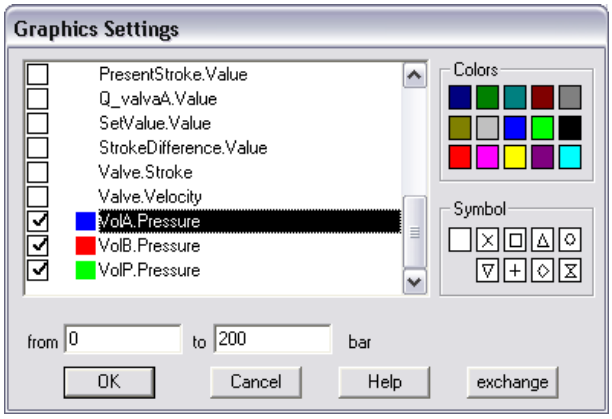
 © FLUIDON® GmbH 2008



New simulation graphics

Repeat this step if you want to open further simulation graphics.

2. By double clicking on the grid in the **simulation graphics** window, the dialog **Graphics Settings** is opened.



*Dialog **Graphics settings***

3. Under **GRAPHICS SETTINGS** mark the variables which are to be shown.

4. In order to distinguish them better, the variables are shown in different colours which are preset in series. It is also possible to allocate other colours to these variables by selecting another colour under **Colours**. The selected colour is allocated to the variable which is marked.

5. In addition, markers can be allocated to the variables. They can be selected under **Symbol**. The selected marker is allocated to the variable which is marked.

6. Certain value ranges are allocated in series to the individual variables. These value ranges can be adapted to your requirements in the input fields **from** and **to**. In these fields, the variable which is marked is always processed.

7. Click on **OK** when you have made all adjustments.

Your selection and your settings are taken over for the individual simulation graphics. For every graph of a variable, the names and the value range are shown in the same colour in the simulation graphics window.

If you want to revise the settings of a certain variable, double click on these settings in the caption. By doing this, you open the **Graphics Settings** dialog. In this dialog, the variable which has previously been selected by double clicking, is automatically marked so that this variable can be processed immediately.

In DSH*plus,* up to ten simulation graphics can be shown at the same time. In every simulation graphics, up to eight variables with different colours and markings as well as their own value ranges can be shown. By means of the configuration of a simulation graphics, display of the variables can be made very clearly. A simulation graphics can be called and configured prior to, during or even after a calculation has been made.

Note:

- The time controlled resolution of the graphical display depends on the setting **factor graphics**. This setting is determined under **PARAMETERS** in the **SIMULATION** menu.

- The units of the variables cannot be changed by means of the **Graphics** window. If a unit is to be allocated to a variable or a variable is to be displayed in another unit, this change has to be effected under **INITIAL CONDITIONS** in the **PARAMETERS** menu or in the parameterization dialog of the component.

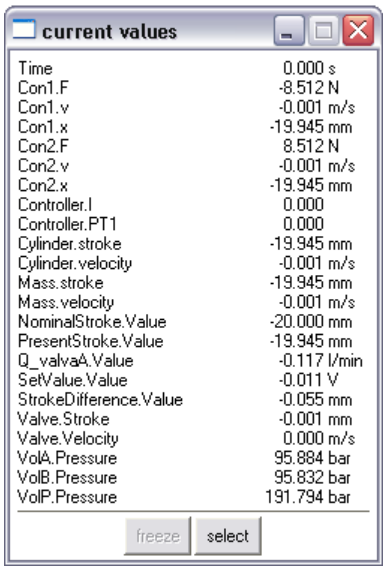
 © FLUIDON® GmbH 2008

- Presetting of the colours is determined in the options dialog.

For further information on:	Please see:
Simulation graphics	Chapter 4, 4.6 Graphic
Adapt simulation graphics	Chapter 4, 4.6.2 Adapt simulation graphics
Animation of simulation results	Chapter 4, 4.13 Play animation

5.6.2 Numeric control

The current simulation data are numerically displayed in the **current values** window. In this window, the current values of all variables or only of the selected variables are displayed.



Display of current values

Proceed as follows in order to call the **current values** window:

1. Click on **CURRENT VALUES** in the **SIMULATION** menu in order to open the **current values** window.

For further information on:	Please see:
Current values	Chapter 4, 4.7 Current values

5.7 Save simulation results

In order to process the simulation results further by means of other program functions, they have to be saved.

Under **SAVE RESULTS** in the menu **SIMULATION** the **Result save Dialog** opens. In this dialog the storage place, file name and type of file can be set.

For further information on:	Please see:
Process simulation results	Chapter 4, 4.9 Save results

6 Close a project

When the work on a project is terminated, close the project by clicking on **CLOSE** in the **PROJECT** menu.

Prior to closing, the project is checked for data that have not yet been stored. If necessary, a dialog is called in which the data can be saved.

Afterwards, you can start a new project or continue an old project.

Note:

- The project is also closed when a new project is created, another project is opened or DSH*plus* is terminated.

 © FLUIDON® GmbH 2008

- The project which has been processed at last within a project directory is marked as pre-selection in the dialog **Open project**.

For further information on:	Please see:
Creating of projects	Chapter 4, 1.1 New
Opening of projects	Chapter 4, 1.3 Open
Save projects with another name	Chapter 4, 1.7 Save as
Importing and exporting of projects	Chapter 4, 1.4 Import

7 Exit DSH*plus*

Select **EXIT** in the **PROJECT** menu in order to quit DSH*plus*. If a project is still open, it is now closed.

 © FLUIDON® GmbH 2008

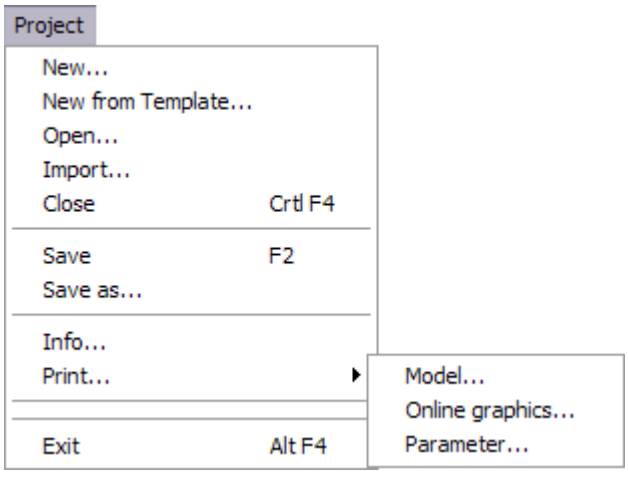
4 The individual functions

This chapter contains detailed information on all program functions. Their knowledge helps to use the comfortable possibilities of DSH*plus* completely and to carry out optimum simulation projects depending on the individual requirements. This chapter is structured as per the structure of the menus of DSH*plus*. Every menu is allocated a subchapter in which the related menu points are described. Furthermore, functions are described in the following which, however, are not mentioned in the menus as menu point but whose content is related to the menu to which they are allocated.

1 Project

Within DSH*plus,* all data belonging to a simulation model are managed via a project. A project is defined via its name and releases the user from handling the files.

The files which are created while working with a simulation model are automatically inserted into the project and marked via a comment put in by the user. The functions for project management can be seen in the **PROJECT** menu.

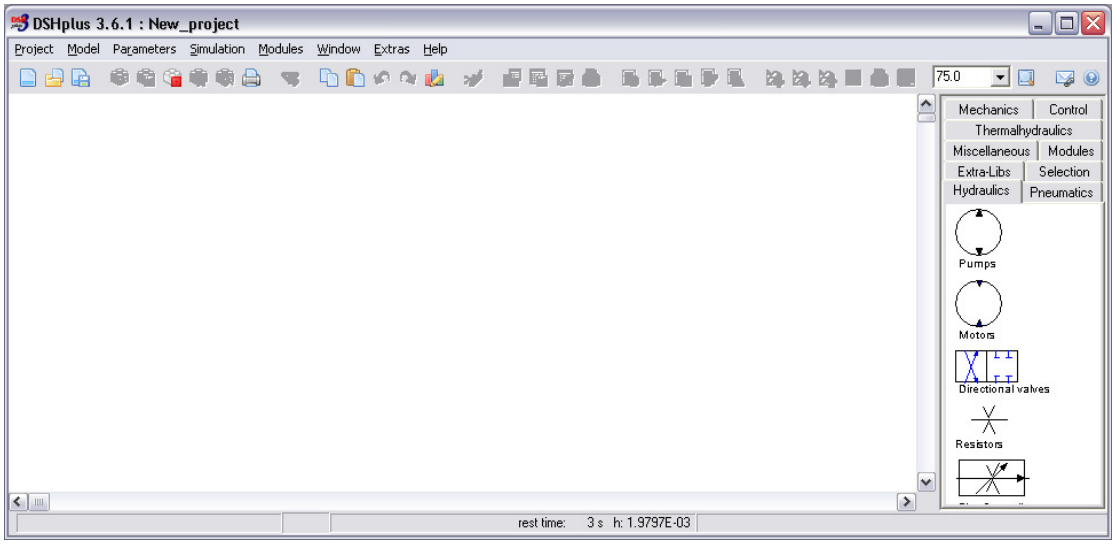


*Main menu **Project***

For further information on:	Please see:
Create projects	Chapter 3, 2 Creating a new project
Close projects	Chapter 3, 6 Close a project

1.1 New

You make a new **PROJECT** over the menu item **NEW** project in the menu project.



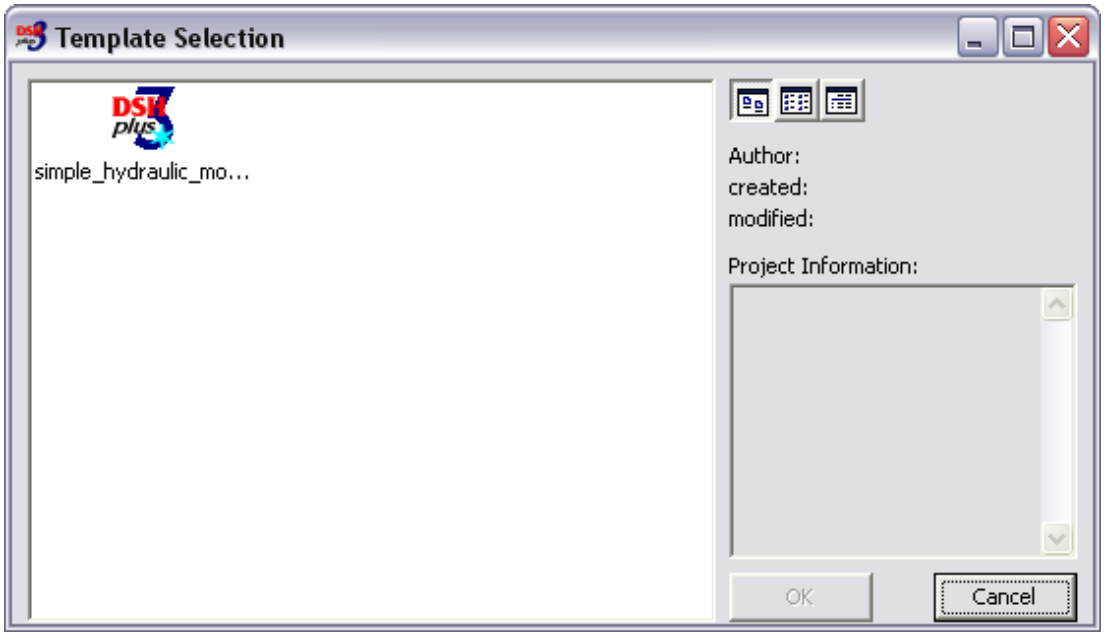
New project „untitled“

You have created a new project without name and the program is in the design mode. This mode can be identified by the fact that the component library is open. Additional project-relevant information such as titles, Author and model information is questioned when storing for the project the first time.

1.2 New from Template

If you want to create a new project from template open the dialog **Template Selection** by means of the menu point **NEW FROM TEMPLATE** in the **PROJECT** menu.

 © FLUIDON® GmbH 2008



*Dialog **Template Selection***

In the dialog all available project templates are displayed. Stored information about the chosen template is printed on the right next to the selection window.

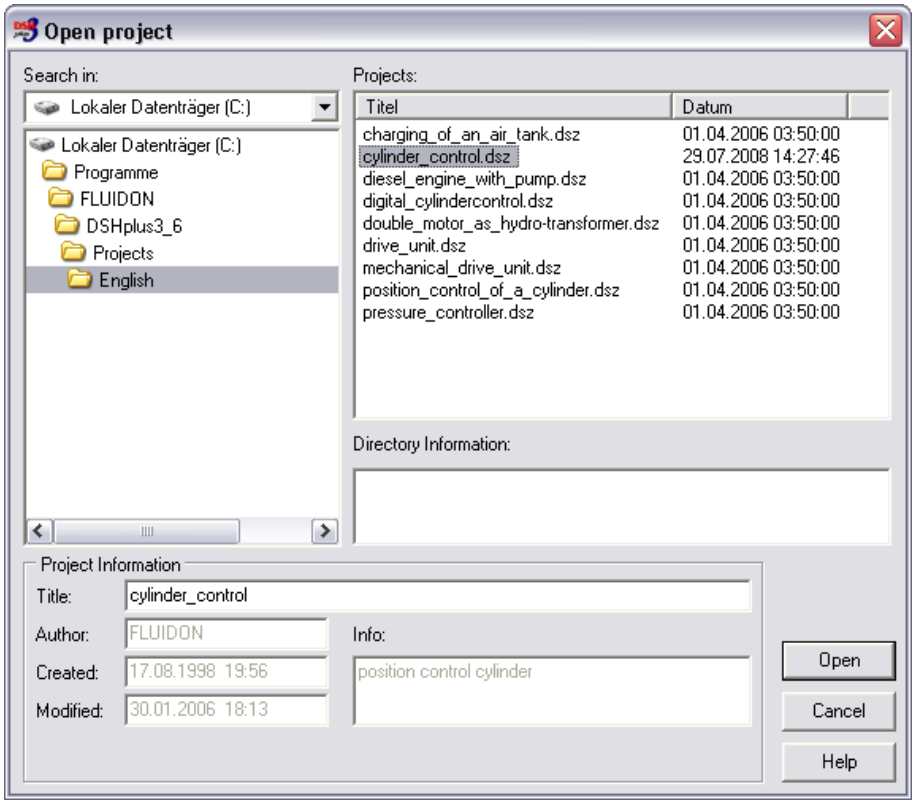
For further information on:	Please see:
Creating of projects	Chapter 3, 2 Creating a new project

1.3 Open

A project which has been created and saved at an earlier stage can again be called for processing by **OPEN** under **PROJECT**.

In order to open a project, proceed as follows:

1. Open the dialog **Open project** via the menu point **OPEN** in the **PROJECT** menu. Under **search in** the last used directory is displayed.



Dialog ***Open project***

2. Select the directory which contains the project that is to be opened.

3. Under Projects choose the project you want to open.

Under Project info information which facilitate the identification of a certain project is displayed.

4. Click on **OPEN**.

Now you have opened a project and you are able to continue the processing.

For further information on:	Please see:
Creating of projects	Chapter 3, 2 Creating a new pro-

© FLUIDON® GmbH 2008

	ject
Saving of projects	Chapter 3, 6 Close a project

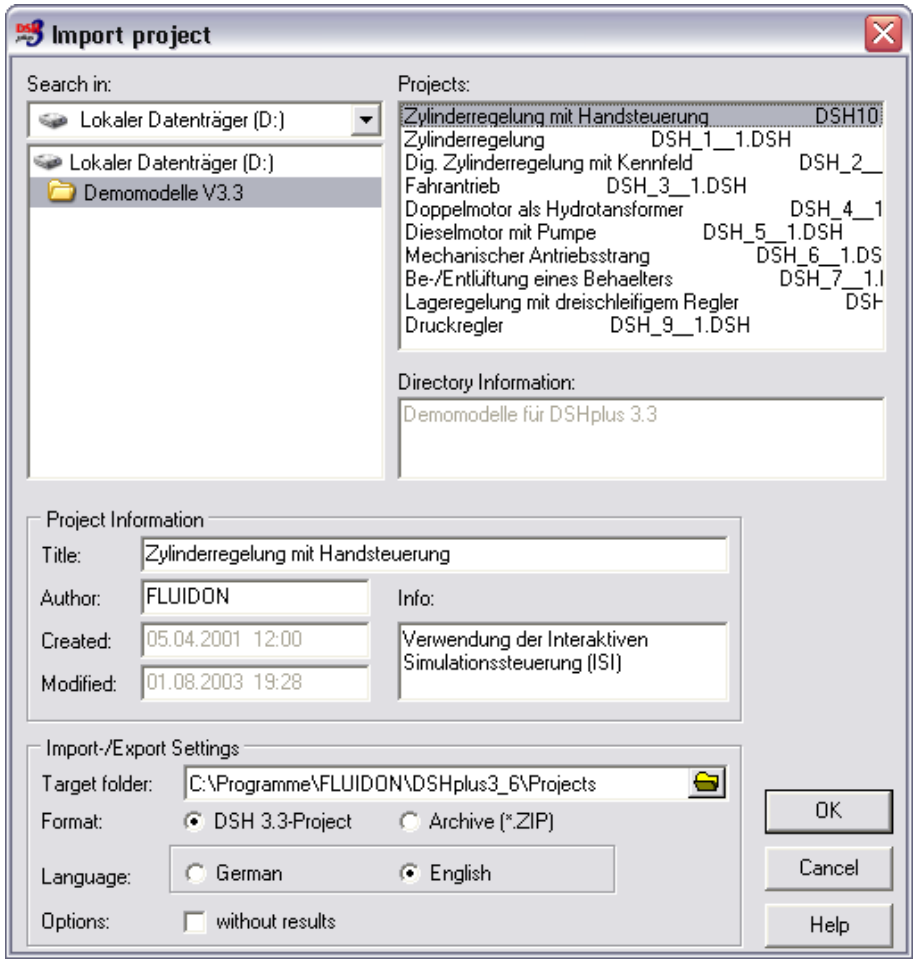
1.4 Import

Projects which have been built with program versions prior to Version 3.4 can be imported with the **Import project** dialog.

Projects, which have been built with program version 3.4 or higher, will be converted automatically during **OPEN**.

Project files both existing in original format and condensed in Zip-format can be imported.

The imported projects are sorted in the existing project list and the files are renamed if necessary.



*Dialog **Import project** (DSHplus 3.3 format)*

In order to import a project existing in original format, proceed as follows:

1. The dialog **Import project** is opened by means of **IMPORT** in the **PROJECT** menu**.** Under **search in** the last used directory is displayed.

2. Select the directory containing the project that is to be imported.

3. In the field **target folder** put in the directory into which you want to import the project(s). You can also click on the opposite button and determine the target directory in the dialog Path selection.

4. If you mark **without results** under **options**, the results are not imported.

5. Mark under **Format** the option **DSH 3.3-Project**.

   In the upper part of the right side of the dialog you now see the field **Projects**, in the lower part of the right side of the dialog the field **Directory info**.

6. Under **projects** select that project or the projects that are to be imported. Multiple selection is possible by means of the shift or the control key.

7. Click on **Ok** in order to transfer the selected projects into the target directory.

In order to import projects that are stored in a standard Zip-format-archive, proceed as follows:

1. Follow steps 1 to 5 for the importing of projects in the original format.

2. Mark under **Format** the option **Archive (*.ZIP)**. In the upper part of the right side of the dialog you now see the field **import file** and below the field **projects**.

 © FLUIDON® GmbH 2008

3. In the field **import file** click on the Zip-files which contain the projects you want to import. In the field **Projects** all DSH*plus* projects existing in the clicked Zip-files are displayed.

4. Click on **IMPORT** in order to import all displayed projects into the target directory.

5. If you have completed the importing procedure close the dialog by clicking on **CLOSE**.

When a project has been imported, it can be opened by means of **OPEN** in the **PROJECT** menu.

1.5 Close

The entry Close in the menu Project ends the currently open project without the main programme to close.

Prior to closing, the project is checked for data that have not yet been stored. If necessary, dialogs are called to save the data.

1.6 Save

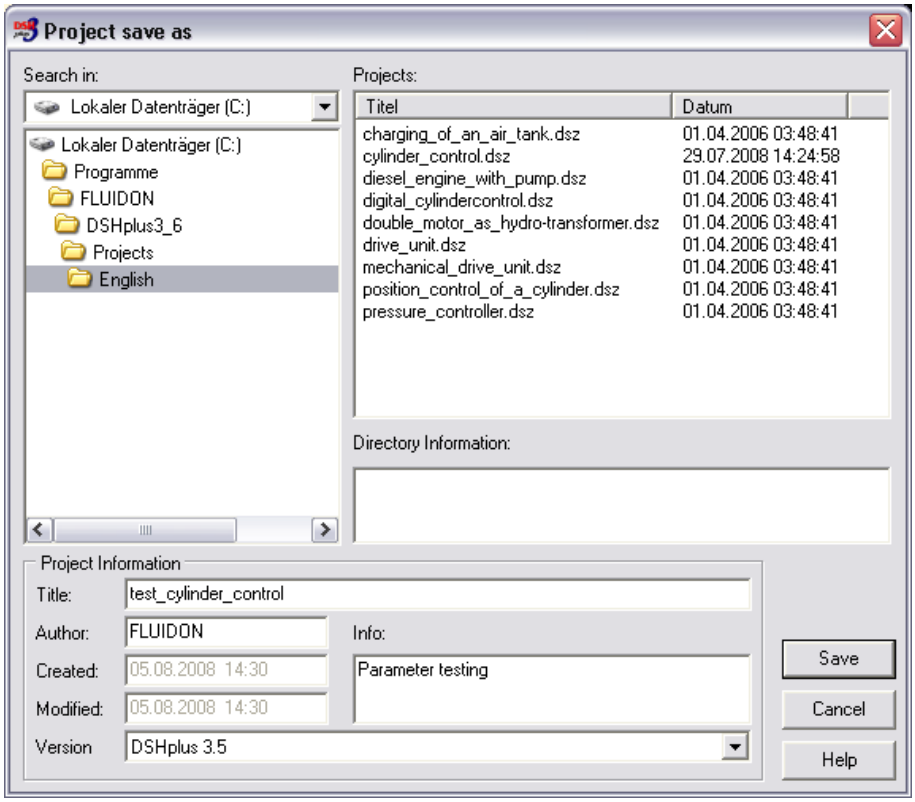
The open project is stored over the entry **SAVE** in the menu **PROJECT** with the current attitudes.

All project relevant files are archived in the DSZ-project-file. The old DSZ-project-file is changed into a BAK file.

1.7 Save as

A project can be duplicated by means of the option **SAVE AS** in the **PROJECT** menu.

The old project is maintained, the duplicate is allocated a new **Name**, a new **Author** and new **Info**.



Dialog ***Project save as***

The application of **SAVE AS** is useful if a new model is to be created on the basis of an existing model and the old model shall be maintained.

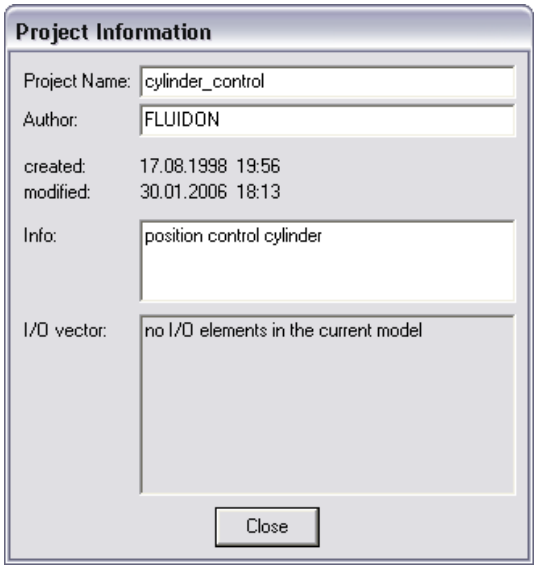
The option **Version** allows you to convert your project to older DSH*plus* versions. This is necessary, because the parameter file format of a current release is not downwards compatible to older releases.

1.8 Info

The dialog **PROJECT INFORMATION** is opened by the menu point **INFO** in the **PROJECT** menu. It displays the information which the user has given when creating a new project (Project title, Author, Remarks). The information can be changed if necessary.

In addition the dialog contains information about date and time the project has been created and date and time of the last model change.

 © FLUIDON® GmbH 2008



Dialog ***Project Information***

If the project contains input or output components for DSH*plus*-STC the dialog also displays a mapping list of those elements to the input and output vector of the simulation model.

For further information on:	Please see:
Create project information	Chapter 3, 2 Creating a new project

1.9 Print

The menu point **PRINT** offers three options, to print the simulation **model**, the opened **online graphics** and the current **parameters** of the model.

Besides the parameters, the printout also contains the indications which are displayed in the info on the current project and the names of the parameter set.

Before starting the printout, the settings for the current printer are displayed in the printer dialog. With the first printout, the settings of the standard printer are taken over by Windows, then the changes you made are maintained. Printout is started by **OK**.

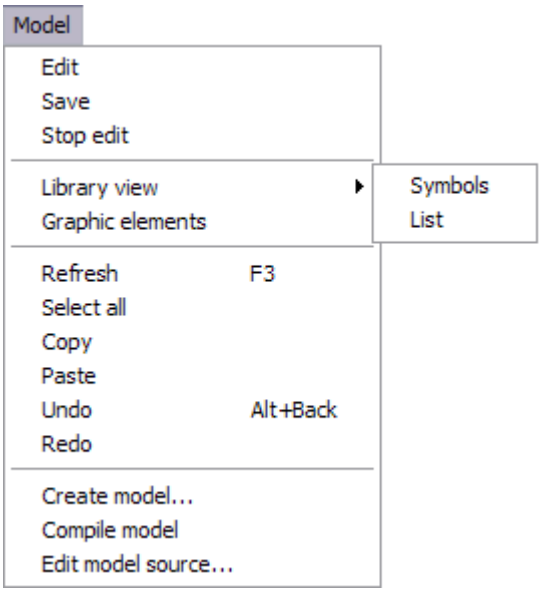
1.10 Exit

To end the DSH*plus* session click on **EXIT** in the **PROJECT** menu.

2 Model

After creating a new project, the DSH*plus* is in the design mode in which the simulation model is designed. In the menu **MODEL**, functions related to the design mode of the model management and model creation as well as a call for the source code editor are located.

Besides these menu points, this chapter describes a number of functions for the model design which are not to be found in the **MODEL** menu.



*Main menu **Model***

 © FLUIDON® GmbH 2008

For further information on:	Please see:
Designing of models	Chapter 3, 3 Drawing models in the design mode

2.1 Edit

The function **EDIT** in the **MODEL** menu is used to transfer from the simulation mode to the design mode. Transfer can be recognized by the fact that the component library is open.

2.1.1 Select and place components

For designing a model, components are selected from the component library and placed on the drawing area.

DSH*plus* is using “pick” and “place” technique for selecting and placing the components.

2.1.2 Drawing functions

The functions described here are used for designing the model.

2.1.2.1 Connection links

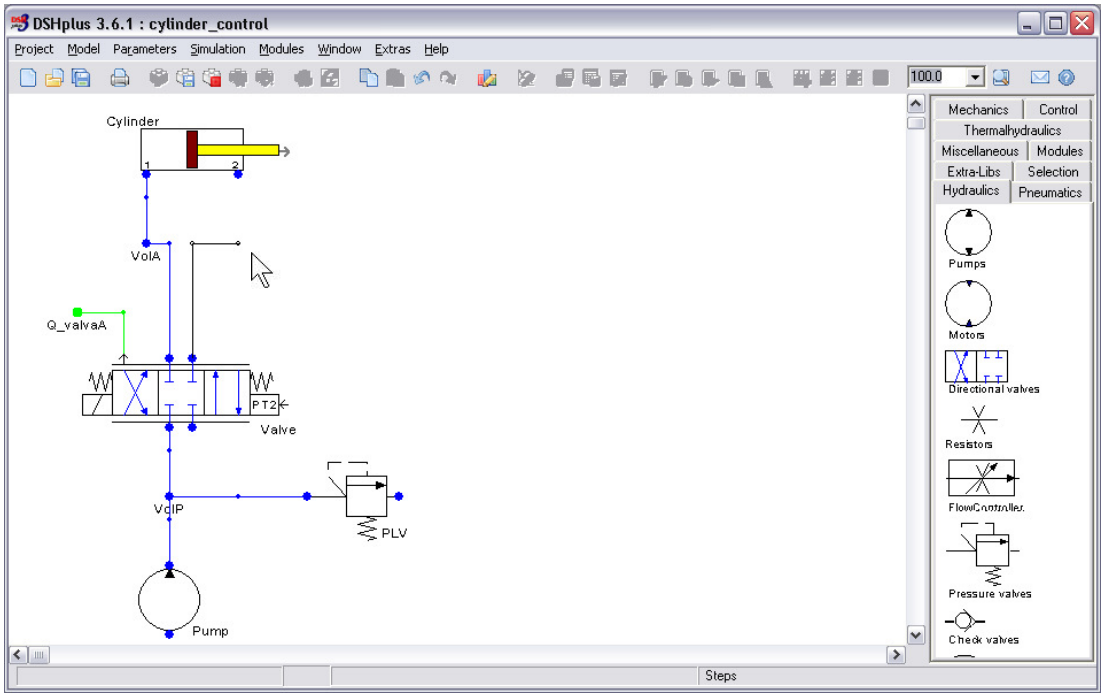
The link between two component connections is created in the simplest case by direct link.

In order to link two components via additional intermediate points, proceed as follows:

1. Use the left mouse button to click on a component connection and keep the key pressed.

2. Draw the mouse pointer to any direction on the drawing area but not to a second component connection.

3. If you release the left mouse button here, a black line is formed with a small circle at the end.



Link of two connectors by intermediate points

4. From this circle, it is possible to draw the line in any direction.

5. You can pass as many intermediate points as you want until you produce the link with the second component.

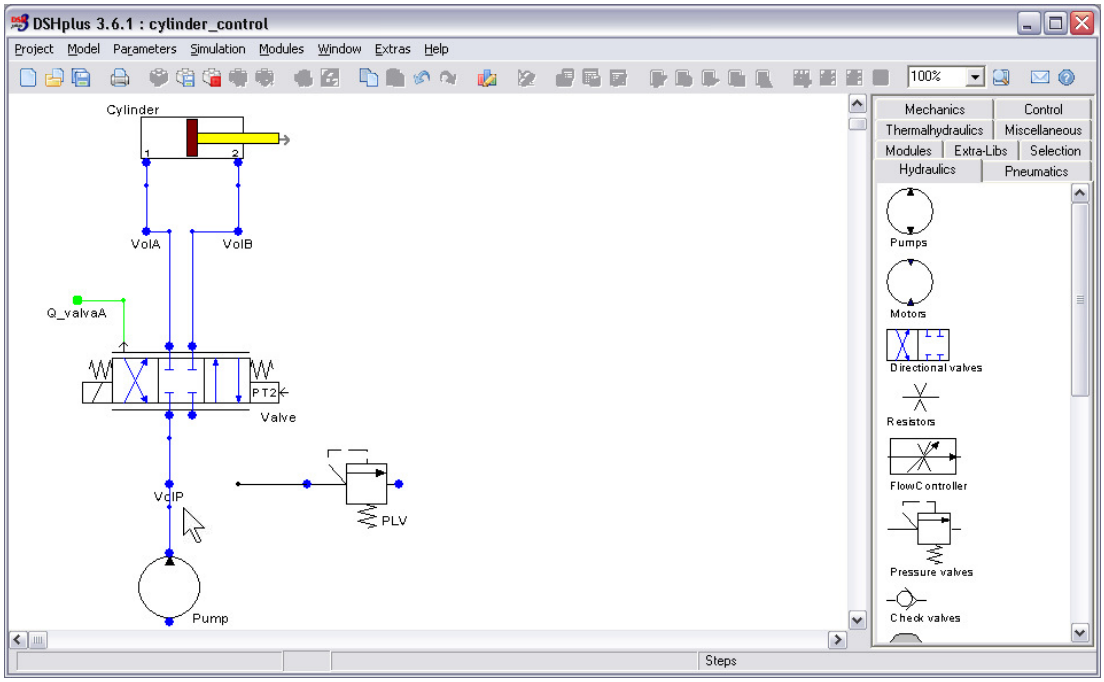
6. It is possible to abort the generation of a link at any time by means of the **Esc** key.

For further information on:	Please see:
Link of components	Chapter 3, 3.3 Connection of components

© FLUIDON® GmbH 2008

2.1.2.2 Link of several components

If you link two components or their connections, a node is automatically added. For linking further components, draw a line from a connection to an existing node.



Draw an additional link to the node

Nodes can also be placed individually on the drawing area in order to be able to create all links from the elements. You can find the nodes as well as the other components in the component library under **Miscellaneous**. A further possibility is to insert a node by means of **INSERT NODE** in the context menu. The new node is inserted at the position of the mouse.

Please note, a link line, starting from a node, is not possible as this may cause logical errors. Instead of creating a link line, the node is displaced.

For further information on:	Please see:
Link of components	Chapter 3, 3.3 Connection of components

2.1.3 Adapt component alignment and component connections

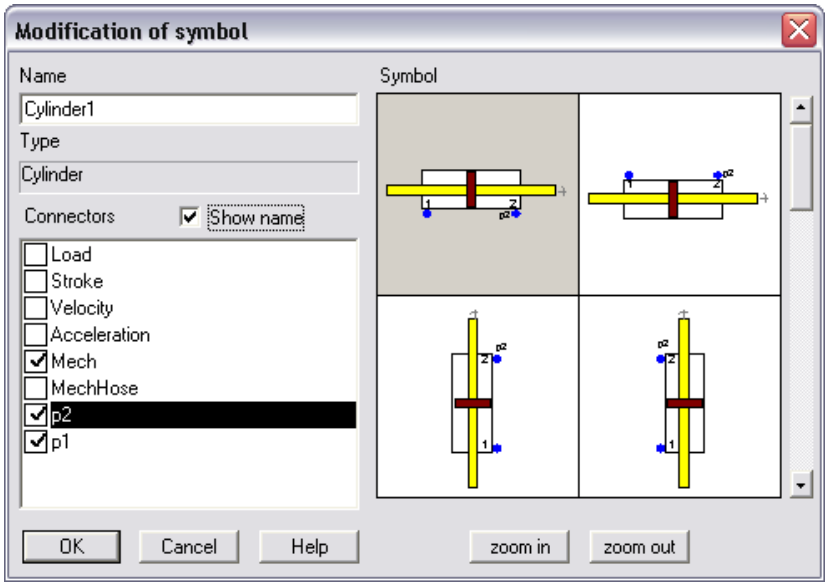
The symbol for a component shown in the library is only one possible view. After taking over a component from the library to the drawing area, put in the name of the new component in the dialog **Modification of symbol**.

This name is only the displayed name of the component and can be changed later on. DSH*plus* is assigning unmistakable component names for the model source code automatically.

Via selection on the options page **Other options**, the system is forced, to accepted only unmistakable identical component names. For the model source code and the display name.

It is also possible to change the appearance of the symbol. Which views of the symbol are shown depends on the fact which views are stored in the library. This varies from component to component. A cylinder for example is shown as synchronous cylinder and as differential cylinder in four different positions each. The signal element symbols, however, are only shown in two views, first with the signal flow from left to right and then with the signal flow from right to left.

 © FLUIDON® GmbH 2008



Dialog Modification of symbol (example: Cylinder)

Select the component connections you need in your model from the list of connections. The designations of the connections are masked out in series. The name of the marked connection is masked in by means of **SHOW NAME**.

During model input, it is possible at any time to open the dialog **Modification of symbol** by double clicking on the component and to carry out changes.

For further information on:	Please see:
Dialog Symbol modification	Chapter 3, 3.2 Select components from the component library

2.1.4 Marking of components

Selection of components for common processing can be carried out in different ways.

1. Marking of individual elements

An individual element is marked by clicking it on by means of the mouse.

2. Marking of several elements

   Several elements can be marked by means of a rectangle. Draw the rectangle round the desired elements when the left mouse button is pressed. If you draw beyond the window frame, the represented section follows.

3. Marking of individual additional elements

   Individual elements are taken up into the selection by keeping the shift key pressed and clicking them on at the same time.

4. Marking of all elements

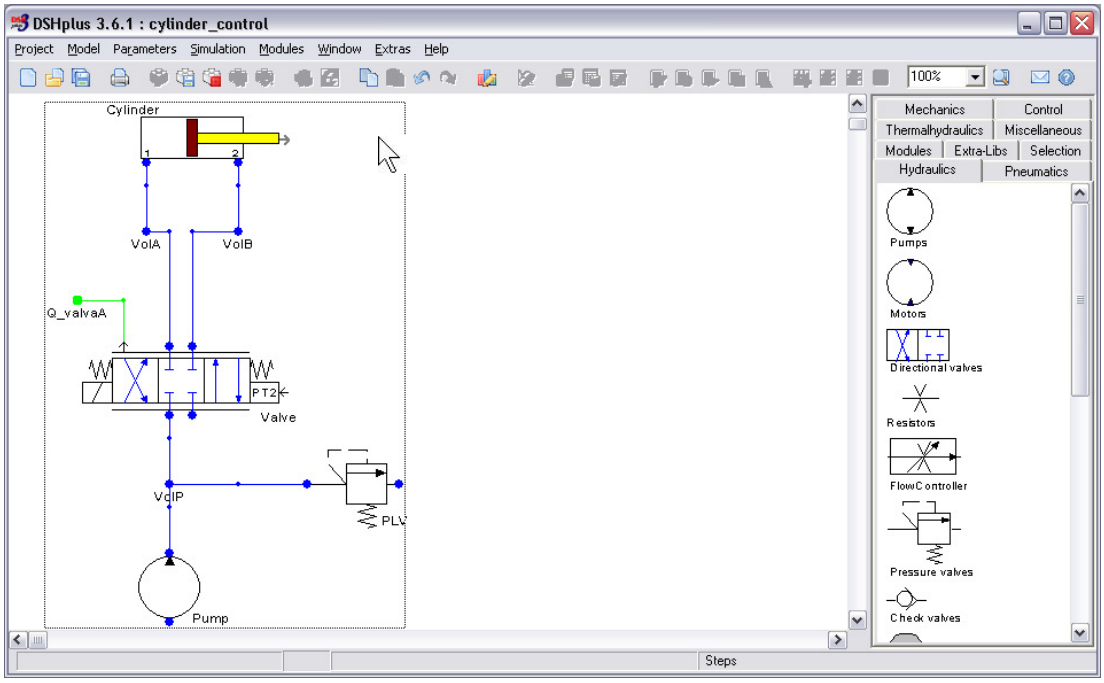
   All elements which are on the drawing area (also those which are outside the visible range) are marked by means of **SELECT ALL** in the **MODEL** menu or by **SELECT ALL** in the context menu.

2.1.5 Displacing of components

In the design mode, components which are already located on the drawing area can be displaced .

First mark the components which are to be displaced. Then, these elements are displaced while keeping the left mouse button pressed at the same time.

 © FLUIDON® GmbH 2008



Displacing of marking to SELECT ALL

For further information on:	Please see:
Marking of components	Chapter 4, 2.1.4 Marking of components

2.1.6 Copying and inserting

In the design mode, components already located on the drawing area can be copied and inserted again into the model or another project.

Proceed as follows, in order to copy an individual component:

1. Mark the component you want to copy.

2. Select **COPY** from the context menu.

3. Open the context menu again and select **INSERT** and the copied symbol is inserted onto the drawing area at another place than the original.

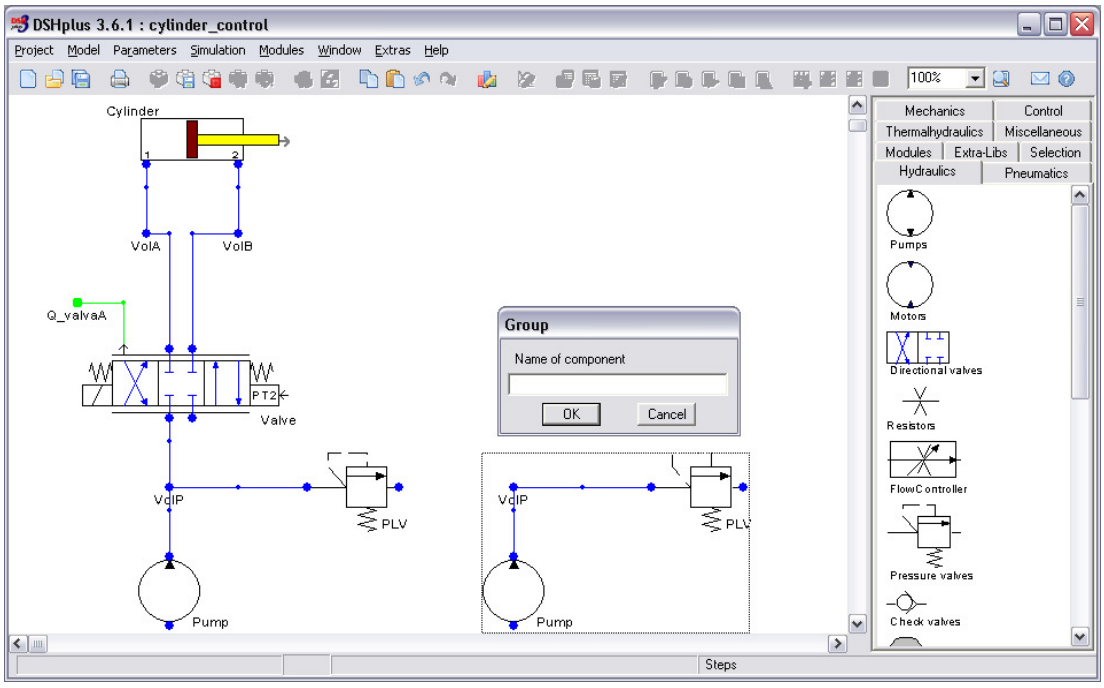
4. The symbol modification dialog in which you have to give the component a definite name, is displayed.

5. **OK** terminates the copying process.

Proceed as follows in order to copy several components at the same time:

1. Mark the desired components you want to copy.

2. Select **COPY** and **INSERT** in the context menu one after the other as with copying an individual component.



Copy of two elements and Dialog Group

3. Now, the dialog **Group** is displayed. The name you put in here is placed before the names of the initial components so that the new components get a definite name.

 © FLUIDON® GmbH 2008

4. When clicking on **OK,** the copied components with the new name are placed onto the drawing area.

5. The copied elements are combined to a group.

In order to insert copied elements into another project, close the current project after copying, open the project in which the elements are to be inserted and insert the copied elements.

For further information on:	Please see:
Cancellation of groups	Chapter 4, 2.1.10 Group and ungroup
Marking of components	Chapter 4, 2.1.4 Marking of components

2.1.7 Copy in clipboard

The model can be copied in the clipboard of Windows as a Windows-Metafile. The model is taken over in its current state, i.e. with all the changes you have made since your work has been last stored when DSH*plus* is in the design mode. The model itself is not changed because of copying.

Calling is made by means of **COPY** in the **MODEL** menu or the design mode by means of **COPY** from the context menu.

From the clipboard, it is possible to take over the model for further processing into other Windows applications such as text processing programs or drawing programs for example. The graphics can also be changed or complemented at a later date in a graphics program.

Please note, it is not possible to paste a Windows-Metafile into a DSH*plus* simulation model. However, it is possible to use Windows-Metafiles as icon templates in the DSH*plus* Component Editor.

2.1.8 Deleting of components and links

In the design mode, individual or several elements can be deleted.

First, the elements which are to be deleted, have to be marked. The selection is deleted by means of the **DEL** key or by means of **DELETE** in the context menu.

When you delete a node, all the links made between the node and other elements are deleted. An individual link can be deleted by marking an intermediate point of the link first and then by deleting the marking by means of the **DEL** key or by **DELETE** in the context menu.

For further information on:	Please see:
Marking of components	Chapter 4, 2.1.4 Marking of components

2.1.9 Undoing and restoring of component design

By means of the function **UNDO** in the **MODEL** menu single steps of the model design can be undone in the design mode.

The undoing of input steps can be revoked by the function **REDO** in the **MODEL** menu.

2.1.10 Group and ungroup

In the design mode, components can be combined into a group. The elements of the group are treated as one unit when being further processed, e.g. all elements are displaced together when displacing is carried out and the arrangement of the group remains the same.

 © FLUIDON® GmbH 2008

Proceed as follows in order to create a group:

1. Mark the components you want to group.

2. Select **GROUP** from the context menu.

Formation of groups can be effected on several levels, i.e. it is possible to combine a group with other components to a new group, etc.

Formation of groups can be cancelled by **UNGROUP** from the context menu. This means that only one group level is released, the groups within the level below are maintained.

For further information on:	Please see:
Marking of components	Chapter 4, 2.1.4 Marking of components

2.1.11 Align elements at the grid

In the design mode, marked components can be aligned at a grid. The grid size can be modified on the options page **Other options**. The function **ALIGN TO GRID** is located in the context menu.

For further information on:	Please see:
Setting of grid	Chapter 4, 7.1.5 Other options
Marking of components	Chapter 4, 2.1.4 Marking of components

2.1.12 Refresh model graphics

The model graphics can be drawn again by means of the **F3** key or by **REFRESH** in the context menu or in the **Model** menu. This is useful if, for example residues of deleted lines remain on the drawing area.

2.2 Save

While creating a simulation model, save your work from time to time. This is done either by means of the menu point **SAVE** in the **MODEL** menu or by means of **SAVE** from the context menu. The function save is only active if a modification has been made since the work has been last stored.

On the options page **Other options**, there is also the possibility to switch on an auto-save mode.

2.3 Stop edit

When you have graphically created a simulation model, close the design mode by means of the menu point **STOP EDIT** in the **MODEL** menu.

Should you not have saved the input, DSH*plus* asks you whether you want to save the last changes. If you select **YES**, your input is saved. If you select **NO**, the changes you made since you last stored your work, are rejected, the last status of the model is restored and the design mode is terminated. With **CANCEL** you cancel the termination of the design mode and remain in this mode.

If the design mode is left, the model is automatically created and compiled. This function can be disabled on the options page **Other options**.

If compilation was successful, the simulation model is loaded immediately after closing the dialog. If errors occur while the model is created or compiled, they are displayed in the window **Messages, errors and warnings**.

 © FLUIDON® GmbH 2008

For further information on:	Please see:
Setting of grid size	Chapter 4, 7.1.5 Other options
Marking of components	Chapter 4, 2.1.4 Marking of components

2.4 Library view

In the design mode two possibilities for the representation of the library are available.

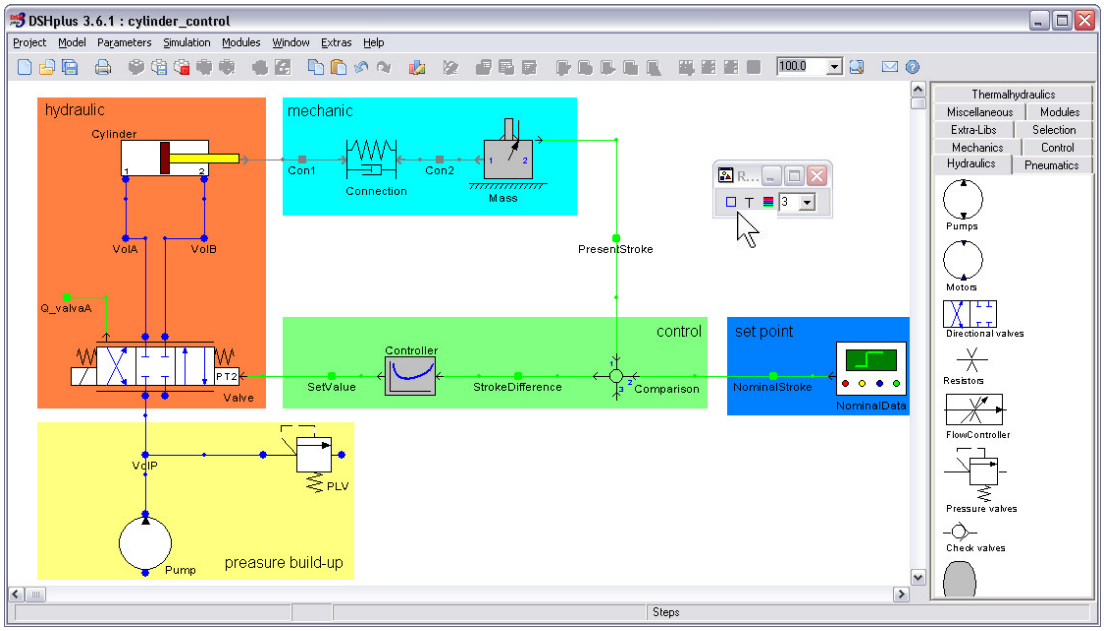
Besides the symbolic standard view a classic text based list view is also available. If the mouse pointer is put over a component name, then the accompanying component symbol is displayed.

The view change is carried out via the function **LIBRARY VIEW** in the menu **MODEL**.

2.5 Graphic elements

A simple drawing function and a text-field are available for the identification and highlighting of single components or assemblies.

The dialog is called over the field **GRAPHIC ELEMENT** in the menu **MODEL**.



With color underlayed simulation model

Attributes like filling area colour and text field parameter can be changed.

2.6 Refresh

The model graphics can be drawn again by means of the **F3** key or by **REFRESH** in the context menu or in the **MODEL** menu. This is useful if, for example residues of deleted lines remain on the drawing area.

2.7 Select all

During the input mode all elements which are on the drawing area (also those which are outside the visible range) are marked by means of **SELECT ALL** in the **MODEL** menu.

2.8 Copy

With this function the highlighted components are inserted in the clipboard.

 © FLUIDON® GmbH 2008

2.9 Paste

With this function the copied components are inserted from the clipboard into the drawing area.

2.10 Undo

By means of the function **UNDO** in the **MODEL** menu single steps of the model design can be undone in the design mode.

The number of available Undo step is specified on the options page **Other options**.

2.11 Redo

The undoing of input steps can be revoked by the function **REDO** in the **MODEL** menu.

2.12 Create Model

When a model is created, the until this time pure graphical model is transferred into a simulation model.

At this stage the descriptions of the individual components are loaded from the library, the model-specific set of differential equations is generated and the resulting source code is compiled to a simulation model. At the same time, a parameter set is created which contains the base parameters of all components.

After calling the **CREATE MODEL** in the **MODEL** menu, the corresponding dialog is displayed only, if the **Automatically model build** function is disabled on the option page **Other options**. Otherwise the simulation model is created automatically.

In case there are errors the dialog will appear even if the **Automatically model build** function is enabled on the option page **Other options**.

When the model is created, messages on the individual steps which are processed, are displayed in the upper window of the dialog.

In the middle window, messages, errors and warnings are displayed which occur when the graphical model is created or transferred into an operational program.

The bottom window displays the compiler output.

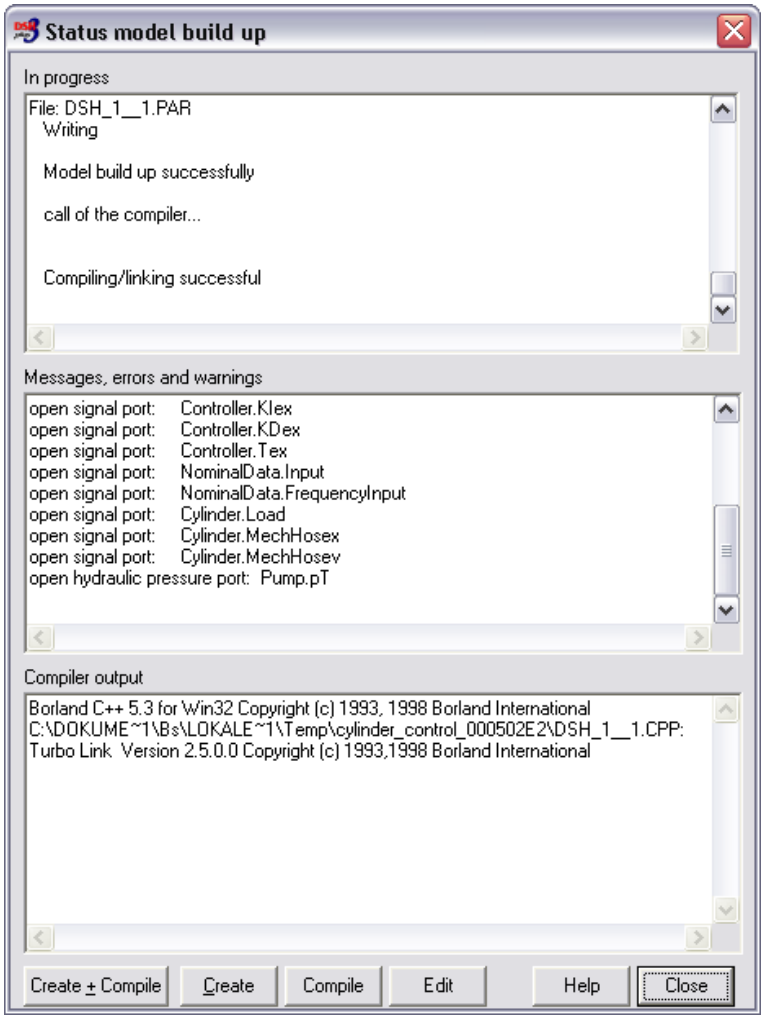


Dialog ***Status model build up***

The combination of creation and compilation can be called by **CREATE + COMPILE**. The model creation is started first.

If this was successful, DSH*plus* automatically calls the compiler.

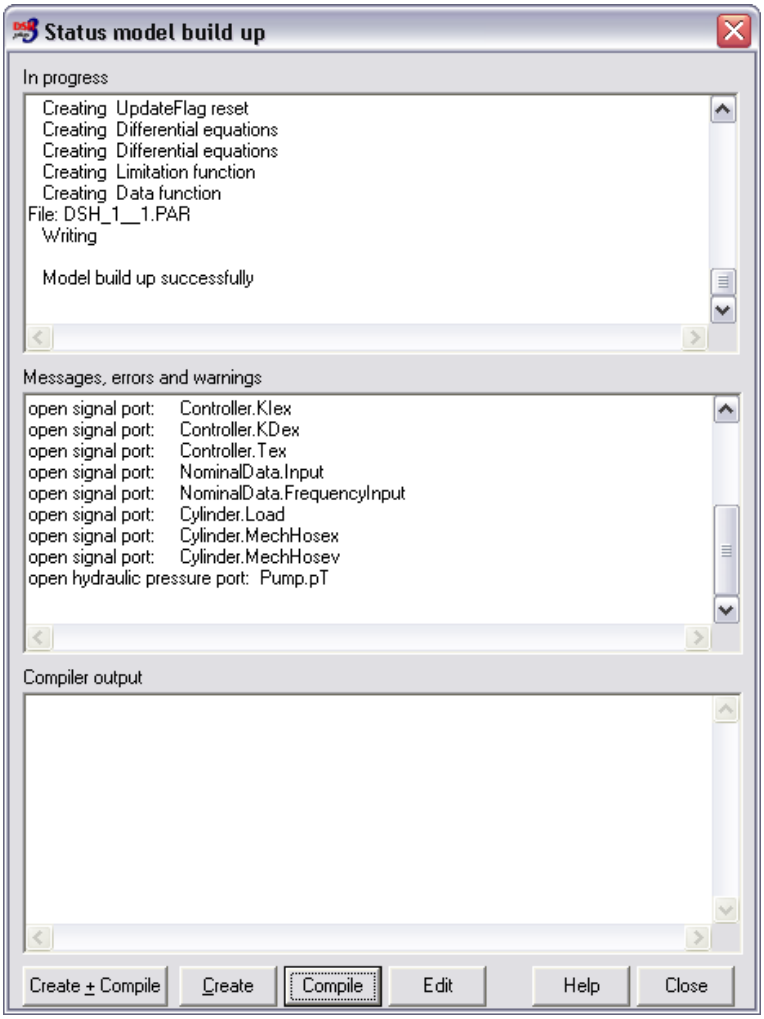
 © FLUIDON® GmbH 2008



*Dialog **Status model build up** after successful model create + compile*

If compilation was successful too, the simulation model is loaded immediately after closing the dialog.

The button **CREATE**, initiates a manual creation of the model source code and the parameter set. The compiler is not yet called in this step.



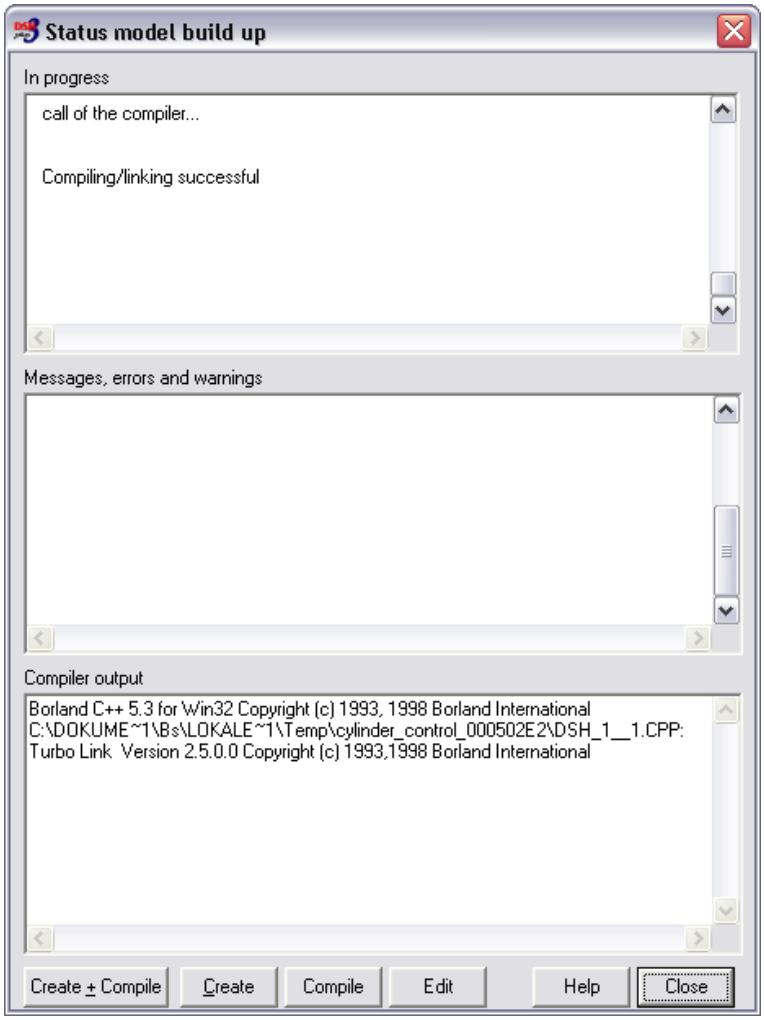
*Dialog **Status model build up** after successful model creation*

2.13 Compile Model

Starting of the compiler and the linker is effected by means of the **COMPILE** button. This function is useful when you have changed the model source code and want to takeover the modification into the simulation model.

The model source code can be accessed by means of the **EDIT** button.

 © FLUIDON® GmbH 2008



*Dialog **Status model build up** after successful model compile*

2.14 Edit model source

The mathematical models of the simulation models are stored as C++ source code.

It is possible to see and edit this source code. To do this, select **EDIT SOURCE** of the **MODEL** menu.

If you carry out any changes at the source code, the model has to be re-compiled so that the changes can be effective.

The selection of the editor that is assigned to the DSH*plus* source code can be changed on the option page **Other options**.

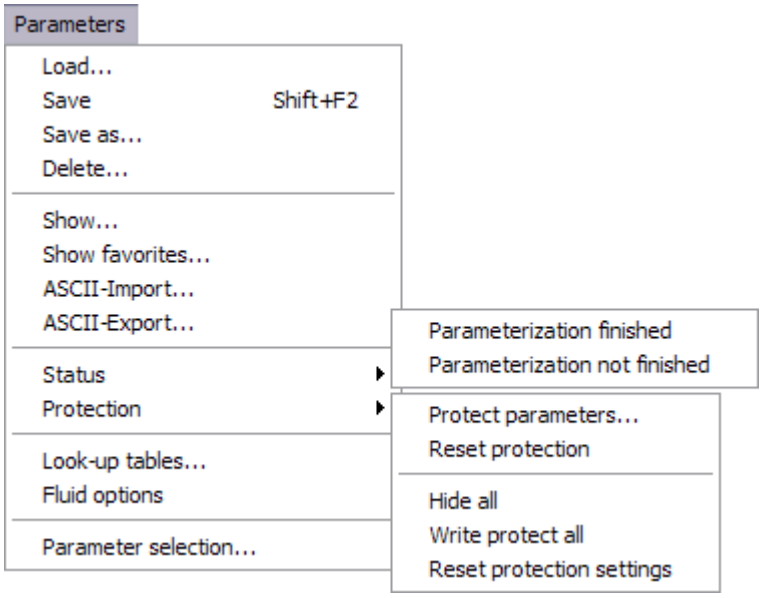
For further information on:	Please see:
Structure of source code	Chapter 5, 3 Structure of the source code of a component
	Chapter 5, 6 Structure of the source code of a simulation model
Selection of the editor	Chapter 4, 7.1.5 Other options
Creating a model	Chapter 4, 2.12 Create Model

3 Parameters

Each model is in principle characterized by its mathematical description which is made out of its graphical structure. Furthermore, by the settings of the individual component parameters the model behaviour is adapted to that of a real system.

In DSH*plus,* difference is made between three parameter types: component parameters, simulation parameters and starting values. The functions with which a parameter set, the general name for the quantity of all the parameters, are processed, are located in the menu **PARAMETERS**.

 © FLUIDON® GmbH 2008



*Main menu **Parameters***

For processing the individual component parameters, functions are available in the respective component dialogs which can be opened by double clicking on a component.

Component parameters

Component parameters are the constant (geometric or physical) values which characterize a component.

Examples are diameter, length or mass.

Component parameter set

This is the sum of all parameters that are required for the components description or determine the boundary conditions of nodes. The values are adjusted in dialogs that open by double-clicking on the respective component.

Component parameter sets can be archived into a database for a later reuse.

Short view of component parameters

If in the simulation mode the mouse is placed on a component, a yellow flyer appears which displays the individual component parameter set.

The flyer also display the source code class name of the component and the internal automatically assigned component name.

The flyer can be turned on or off on the **Other options** page in the dialog **Options**.

Initial values

Starting values are the initial values of the systems state variables.

Two different kinds of variables are calculated:

1. Variable values are obtained from the integration of differential equations. The starting values represent only the initial variable values at the beginning of the simulation.

2. Algebraic values are calculated according to a set of algebraic, non integrated function (process values). The calculation of the individual values is independent from the starting value. Therefore the initial values do not have any effect on the calculation.

Parameters of Simulation

The simulation parameters define the boundary conditions of the calculation (start and end time) and the numeric procedure with which the simulation model is calculated.

For further information on:	Please see:
Significance and function of the indi-	Component manual

© FLUIDON® GmbH 2008

vidual component parameters	
Individual simulation parameters	Chapter 4, 4.4 Parameters
Setting of component parameters	Chapter 3, 5.1 Set parameters and starting values of the nodes and components
	Chapter 4, 3.3.2 Component parameter set
Setting of starting values	Chapter 3, 5.1 Set parameters and starting values of the nodes and components
	Chapter 4, 4.15 Initial conditions
Setting of simulation parameters	Chapter 3, 5.2 Set simulation parameters
Loading of parameter sets	Chapter 4, 3.1.1 Parameter set
Loading of component parameter sets	Chapter 4, 3.1.2 Component parameter set
Saving of parameter sets	Chapter 4, 3.2 Save
Saving of new component parameter sets	Chapter 4, 3.3.2 Component parameter set

Transfer of parameters and variables

In the simulation mode the parameter sets and variable values of a component can be transferred with drag and drop to an other component.

In order to do this, click with the left mouse button on the component whose parameters shall be transferred. Keep the button pressed until a P-symbol is displayed. Then drag the pointer onto the component onto which the parameters are to be transferred. When the mouse pointer is positioned on this component release the mouse button. The parameter transfer is complete.

The transfer of parameters can also be done by means of the functions **COPY PARAMETERS** and **PASTE PARAMETERS**. These functions are in the context menu which is opened when a component is clicked on with the right mouse button.

To copy variable values press the **Strg** key and the left mouse button. Instead of the P-symbol a V-symbol appears.

The transfer of variable values can also be done by means of the functions **COPY VARIABLES** and **PASTE VARIABLES**. These functions are in the context menu which is opened when a component is clicked on with the right mouse button.

3.1 Load

The loading functions for parameter assign stored values to the model . There are complete parameter sets and component specific parameter sets.

3.1.1 Parameter set

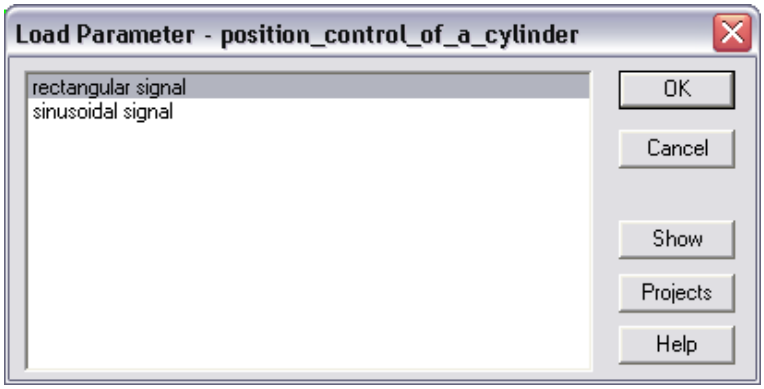
A parameter set represents the entire quantity of component parameter, initial values, and simulation parameters of a model. An alternative names for a parameter set could be “solution set”.

Both parameters sets of the current project and those of other projects can be loaded.

Typically a project contains numerous parameter sets. By means of the menu point **LOAD** in the menu **PARAMETERS** these parameter sets can be re-assigned to the model.

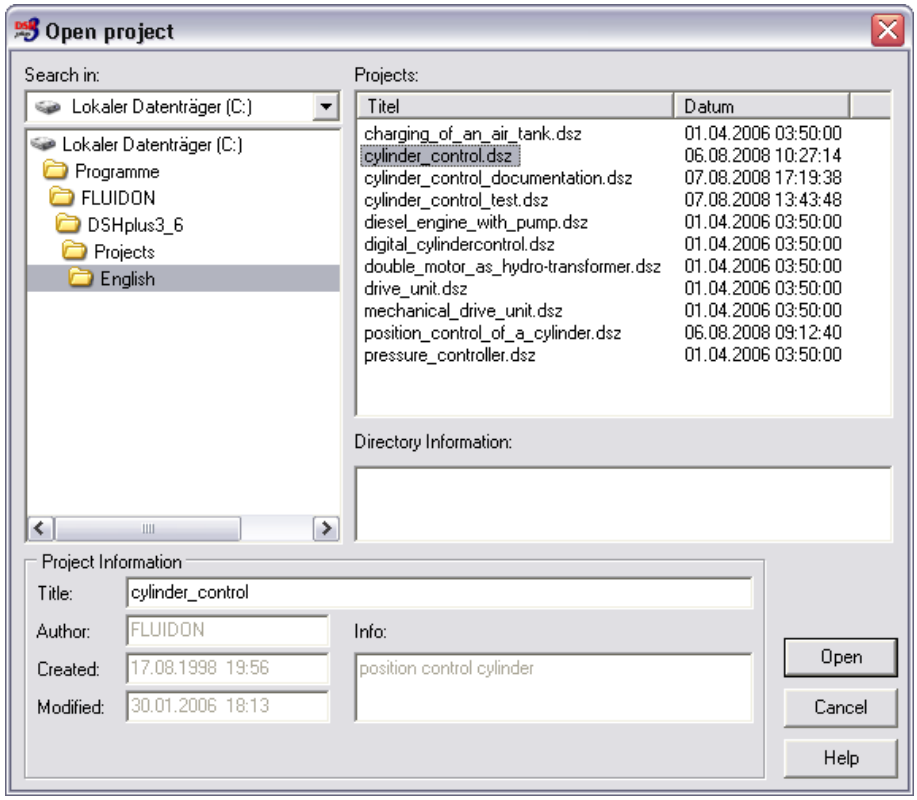
A specific parameter set can be selected and loaded through the dialog **LOAD PARAMETER**.

 © FLUIDON® GmbH 2008



*Dialog **Load parameter***

If a parameter set of another project is to be loaded, the **PROJECTS** button is actuated. The project from which the parameter set is to be loaded can be selected in the dialog **Open project**.



*Dialog **Open project***

A parameter set of the selected project is now loaded in the dialog **Load parameter**.

Simulation parameters, component parameters and component variables are loaded (= Starting values). If the structure of the selected pa-

rameter set differs from the current parameters – this can be possible if a model has been enlarged by one component for example – only parameter values of such parameters are loaded whose name is identical in both parameter sets. Identity has to be available both for the component and for the parameter. All other parameters are not changed.

3.1.2 Component parameter set

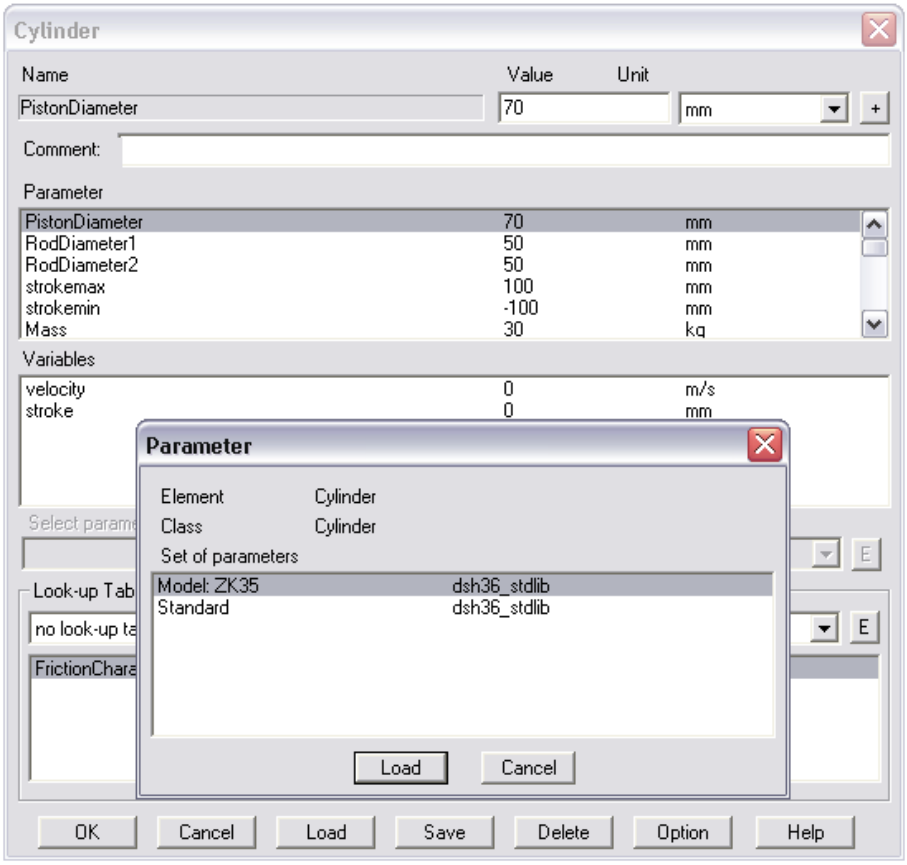
It is also possible to load component parameter sets of individual components if they

- have been archived by a component dialog first of the current project or another project

- and belong to a component which is based on the same initial component class as the component for which the parameterization is to be loaded.

Double-click on the component for which the parameter is to be loaded. Then actuate the **LOAD** key in the specific component dialog. A component specific parameter set can now be selected and loaded in the **Parameter** dialog.

 © FLUIDON® GmbH 2008



Component dialog with dialog for loading of parameters

Note:

- The component parameterization dialogs are only available in the simulation mode.

- The parameter values of standard DSH*plus* components are stored into the ASCII-file “dsh36_stdlib.spa” that is located in the user library folder.

For further information on:	Please see:
Saving of component parameter sets	Chapter 4, 3.3.2 Component parameter set

3.2 Save

In order not to loose any parameterizations, they have to be saved. Both parameter sets and individual component parameter sets can be

saved. Only saved parameter sets and component parameter sets can be loaded and used for parameterization of other models or individual components.

Clicking on **OK** in the various parameterization dialogs for components, starting values and simulation parameters means that the adjusted parameters are used in the current simulation set. This does not mean that the new settings are automatically saved. Changes of parameters thus are lost if the project is closed without saving the parameter set.

The parameterization changes are saved by means of the function **SAVE** in the **PARAMETERS** menu under the current parameter set. This function is only active when changes have been made in the current work session.

On the **OTHER OPTIONS** page of the **OPTIONS** dialog there is also the possibility to switch on an auto save mode with a three minutes saving interval.

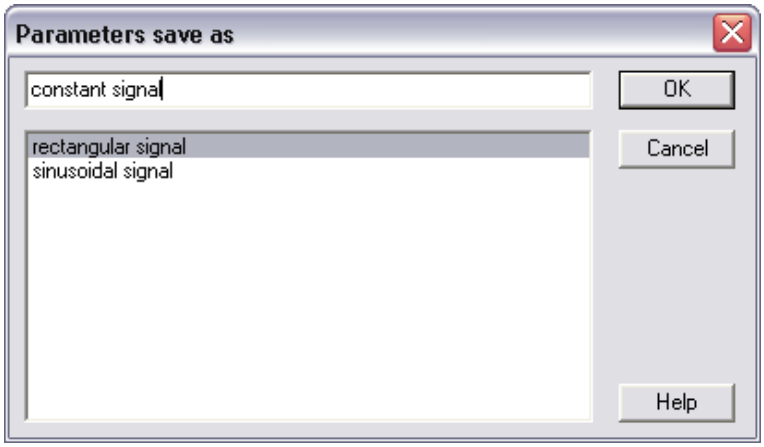
3.3 Save as

3.3.1 Parameter set

In order to save the parameter set and to provide it with an identified comment, proceed as follows:

1. Click on the menu point **SAVE AS** in the **PARAMETERS** menu in order to open the dialog **Parameters save as**.

 © FLUIDON® GmbH 2008



*Dialog **Parameters save as***

2. Put in a parameter set name in the input line or select an already available comment from the list below the input line and click on **OK**.

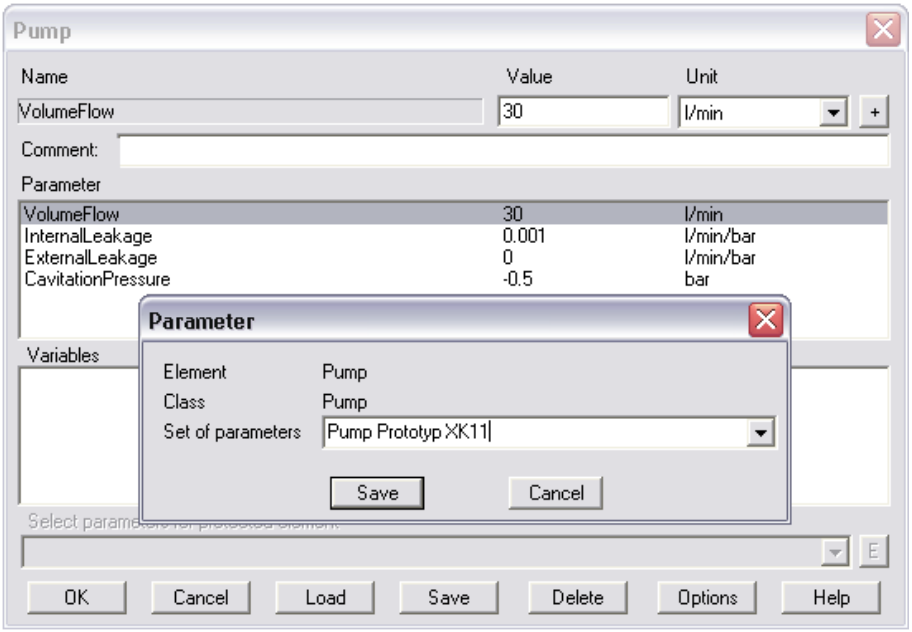
3. If you have entered a new comment, you have thus created a new parameter set that is automatically the active parameter set. If you use a comment already available, a dialog with the note "**… exists. Overwrite?** Yes/No/Cancel"

Note:

- If a parameter set is saved after **Final values as initials** has been selected at the end of a simulation, DSH*plus* saves the calculated variable values and not the starting values originally set.

3.3.2 Component parameter set

The parameters saved for an individual component are available for all components based on the same component class, e.g. for all pumps based on the type **PumpRotationalSpeed** of the group **Pumps**.



Component dialog with dialog for saving of parameters

Saving of the parameterization of the individual component is made in the input dialog of the component parameters which is opened by double-clicking on the respective component. The **Parameter** dialog is opened by means of the key **SAVE**.

The parameter set of the specific component is allocated a name in this dialog and saving is carried out by means of **SAVE**.

Note:

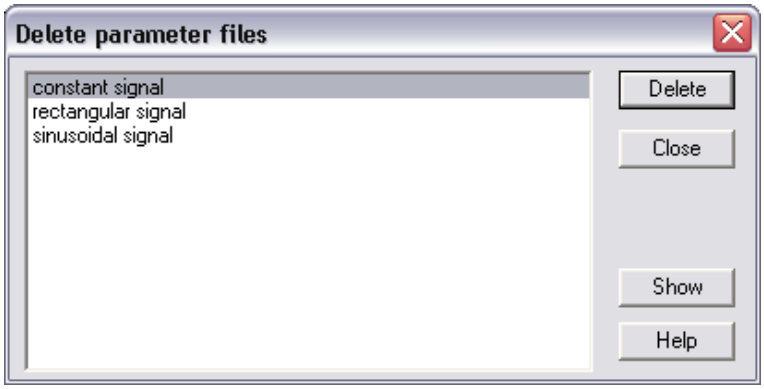
- Saving of parameters is only possible in the simulation mode, not in the design mode.

- The parameter values of standard DSH*plus* components are stored into the ASCII-file “dsh36_stdlib.spa” that is located in the user library folder. This guarantees that the sets are maintained even after an update of the library has been made.

3.4 Delete

For deleting a parameter set, the menu point **DELETE** has to be selected under **PARAMETERS**.

 © FLUIDON® GmbH 2008



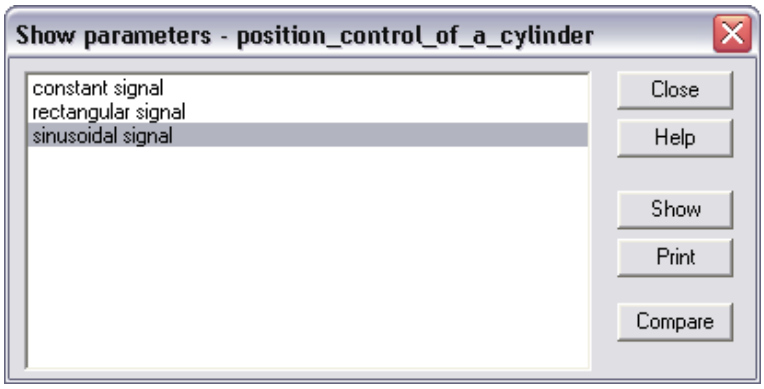
*Dialog **Delete parameter files***

One or several items are selected in the list with the parameter sets. They are deleted by clicking on the **DELETE** button. If only one parameter set is selected, it can be seen by means of **SHOW**. In this way you can control if the selected parameter set is actually to be deleted.

The active parameter set of a project cannot be deleted even if it is selected.

3.5 Show

The dialog **Show parameters** is opened by means of **SHOW** in the **PARAMETER** menu.

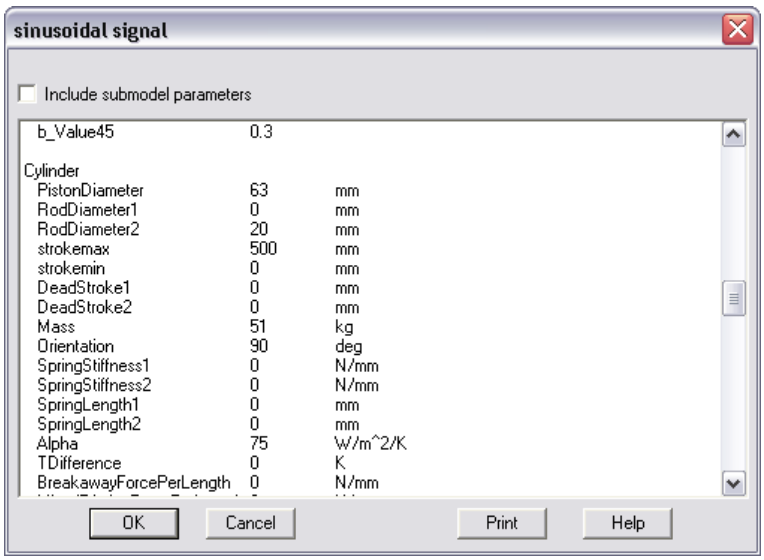


*Dialog **Show parameters***

This dialog combines several functions which are described in the following.

Show:

This function displays a selected parameter set. Direct printout can be effected from the display dialog.



Show of parameters

If several parameter sets are marked in the dialog **Show parameters**, the function **SHOW** cannot be used.

Print:

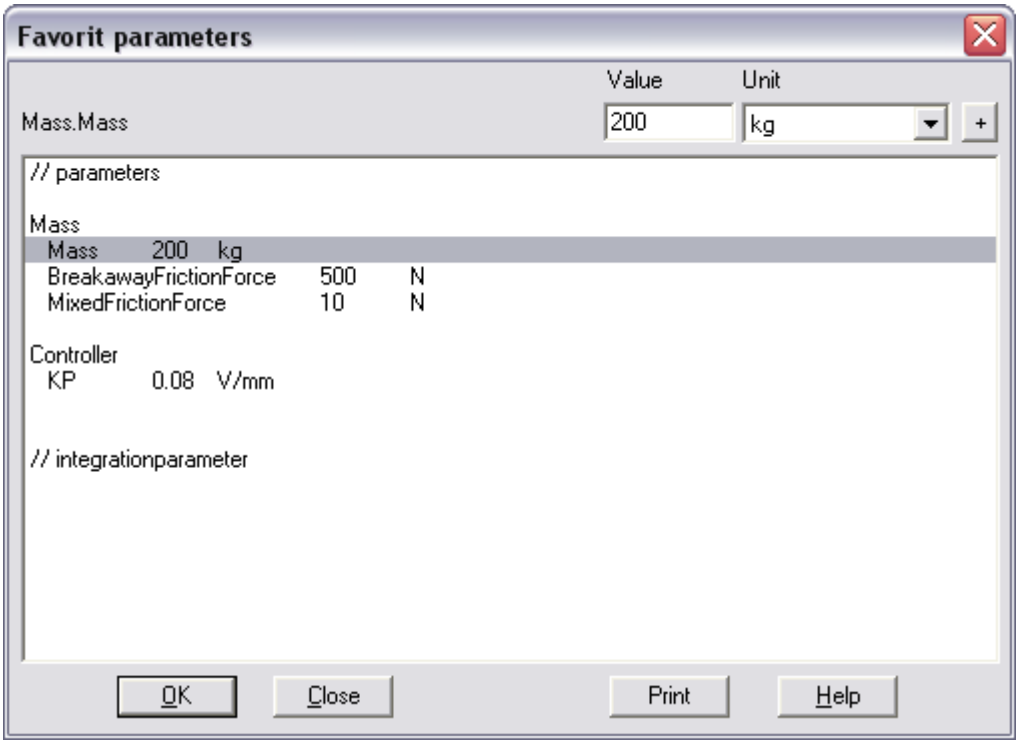
The **PRINT** function enables to print all parameter sets which are marked in the dialog **Show parameters**. If several parameter sets are marked, they are printed out one after the other.

3.6 Show favourites

Parameters which are frequently changed can be included in a favourites list. This dialog makes it possible to quickly change parameters without calling the corresponding parameter dialogs.

The Dialog **FAVOURITE PARAMETERS** is called via the function **SHOW FAVORITES** in the menu **PARAMETERS**.

 © FLUIDON® GmbH 2008

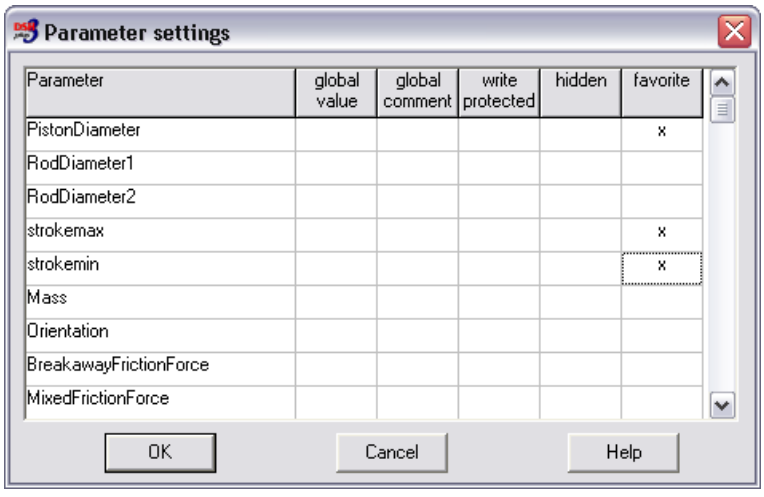


*Dialog **Favourite parameters***

To include component parameters in the list, you proceed as follows:

1. Open the component parameter dialog of the component.

2. The button **Options** opens the **PARAMETER SETTINGS dialog.**



*Dialog **Parameter settings***

3. Mark the desired parameters in the column **favourite**.

4. With OK the highlighted parameters are inserted in the favourites list.

To include simulation parameter in the list, open the dialog **SIMULATION PARAMETER** and execute the steps 2 to 4 mentioned above.

3.7 ASCII-Import

By means of the function **ASCII-IMPORT** in the **PARAMETERS** menu, parameters, existing in a specific ASCII-format, can be imported into a current project. The easiest way to get a parameter set file in the right ASCII-format, is to export a parameter set via the **ASCII-EXPORT** dialog.

If you want to import ASCII-files, proceed as follows:

1. Open the project into which the parameters are to be imported.

2. Select in the **PARAMETERS** menu the point **ASCII-IMPORT**.

3. Import the ASCII-file containing the parameters by means of the dialog **Open Import File**.

It is also possible to import only partial parameter set files.

This is helpful if the files are generated automatically by external programs. However, all parameter section headlines, indicated through "//" characters, have to be included into the file. Even if the corresponding parameter section is empty.

```
Parameter set name
DSHplus Version     x.y

// Parameter

Component name1.Parameter name1       Value       Unit
Component name1.Parameter name2       Value       Unit
Component name2.Parameter name1       Value       Unit
Component name2.Parameter name2       Value       Unit

// Variablen

// Integrationsparameter
```

 © FLUIDON® GmbH 2008

There is also the possibility to have an automatically parameter update, previous of an individual simulation run. To activate the automatic parameter update select **AUTOMATICAL UPDATE OF PARAMETERS** on the **Other options** page of the **Options** dialog.

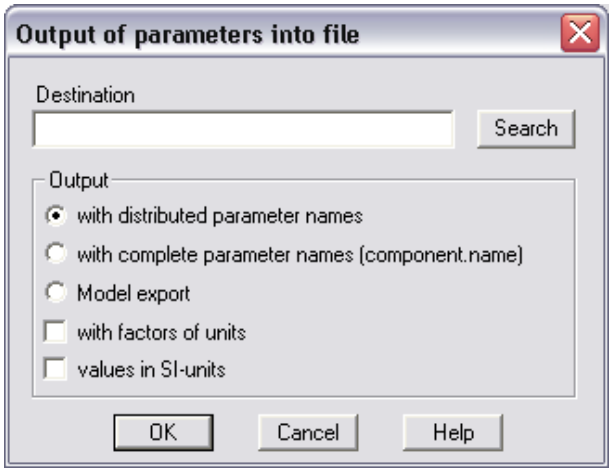
Note:

- The ASCII-parameter file for the automated parameter update must have the same name as the DSH*plus* simulation model.

3.8 ASCII-Export

With **-> FILE,** the parameter sets are written in files in the ASCII-Format. The parameter sets thus exported can be transferred into a text processing program for example. Furthermore they can be imported via **ASCII-IMPORT** in the **PARAMETER** menu into a DSH*plus* project.

If **FILE** is clicked on, the dialog **Output of parameter into file** opens. The target file can directly be put in the file or determined by **SEARCH**.



*Dialog **Parameter output into file***

Output of parameters can be selected between the alternatives **with distributed parameter names** and **with complete parameter names (component.name)**.

The alternative **with distributed parameter names** means that the component name is put out above and the related parameters are put out below and indented:

```
Parameter set name

// Parameter


Component name1
     Parameter name1      Value       Unit
     Parameter name2      Value       Unit

Component name2
     Parameter name1      Value       Unit
     Parameter name2      Value       Unit
```

On the other hand, **with complete parameter names (component.name)** means that component name and parameter name are always completely indicated:

With **Model export** the whole parameter set of the model is exported as follows.

```
Parameter set name

// Parameter

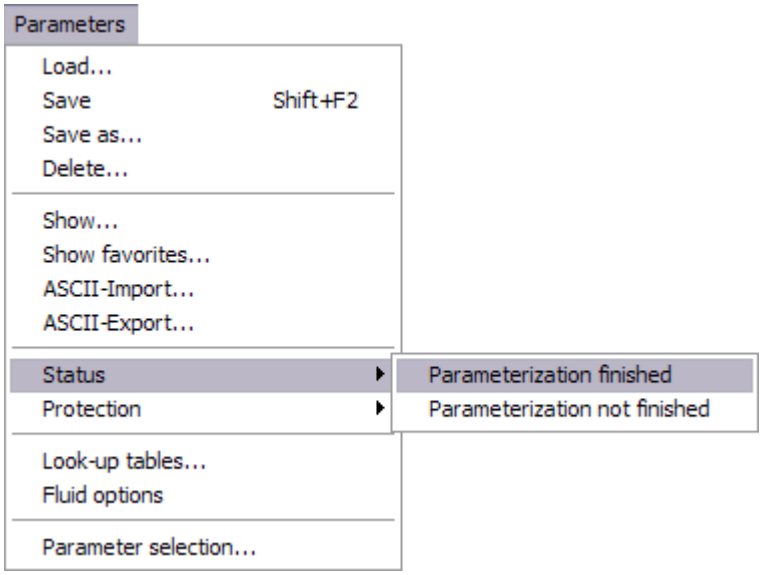
Component name1.Parameter name1      Value       Unit
Component name1.Parameter name2      Value       Unit
Component name2.Parameter name1      Value       Unit
Component name2.Parameter name2      Value       Unit
```

Alternatively to the adjusted units, all values can also be indicated in SI units. Furthermore, the conversion factors of the units can be put out.

3.9 Status

The menu point **STATUS** in the **PARAMETERS** menu provides the two functions **PARAMETERIZATION FINISHED** and **PARAMETERIZATION NOT FINISHED**. These functions are intended to support the user setting the parameters.

 © FLUIDON® GmbH 2008

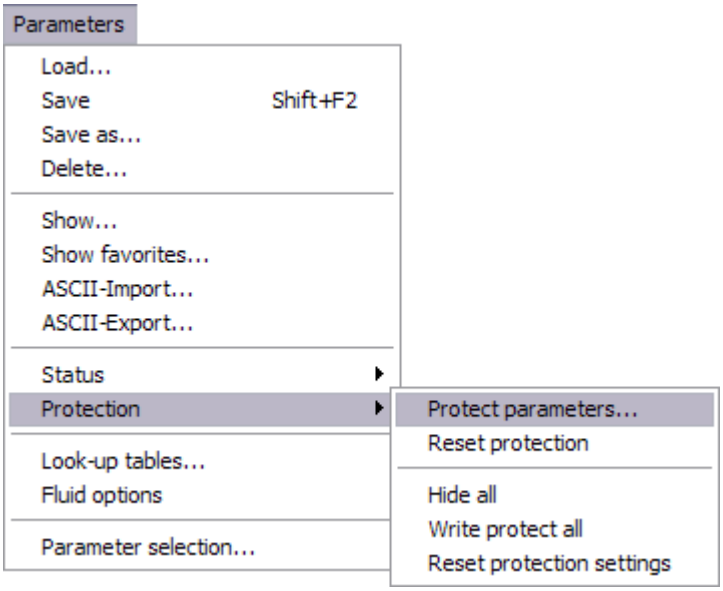


*Menu **Parameter Status***

The names of the components containing parameters which have not been processed are coloured violet. When the processing of the component parameter set is completed the component name is presented in black. If all components are to be processed again the colouring of the components can be changed back into violet by means of the function **PARAMETERIZATION NOT FINISHED**. Thus you can always see which components have been processed already and which have not. Clicking on **PARAMETERIZATION FINISHED** actuates that all component names are displayed in black.

3.10 Protection

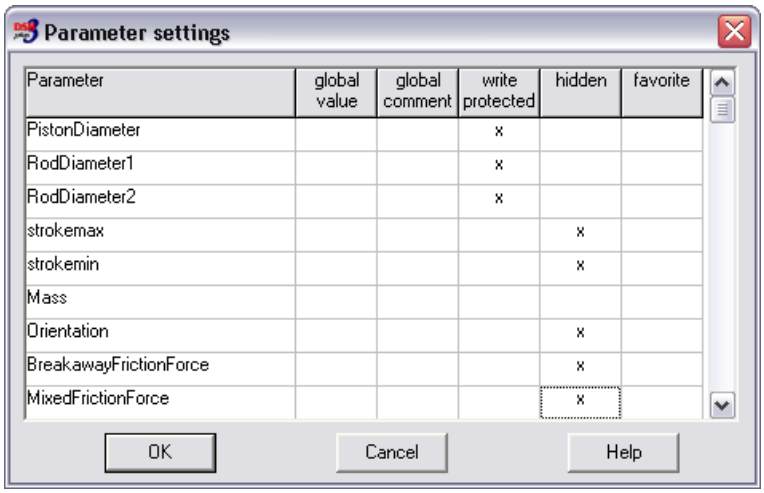
The function **PROTECTION** in the menu **PARAMETER** activates and deactivates the parameter protection functions.



Menu Parameter Protection

By means of **PROTECT PARAMETERS** … the protection is activated, after a password has been assigned. This password is questioned at changes of the parameter attitudes and at the termination of the protection.

Parameters that are marked as "hidden" are not listed in the component dialogs as well as under parameter advertisements. Parameters that are marked as "write protected" are visible but can not be changed.



Dialog Parameter settings: Hidden and write protect

By means of **RESET PARAMETERS** … the protection is deactivated after the password inquiry was answered successfully.

 © FLUIDON® GmbH 2008

The sub-menu function **HIDE ALL** changes all parameter setting to the status hidden.

The sub-menu function **WRITE PROTECT ALL** changes all parameter setting to the status write protected.

By means of **RESET PROTECTION SETTINGS** all parameter settings are reverted to standard status.

Note:

- Protection attitudes refer to the current project and not to the current parameter set. All attitudes are stored with the project.

- The function **RESET PROTECTION SETTING** resets only the attributes of the component parameters. Protection attributes of the simulation parameters remain unchanged and must be removed manually in the simulation parameter attributes dialog.

- Protection settings are kept active if the project is converted into a DSH*plus* sub-model.

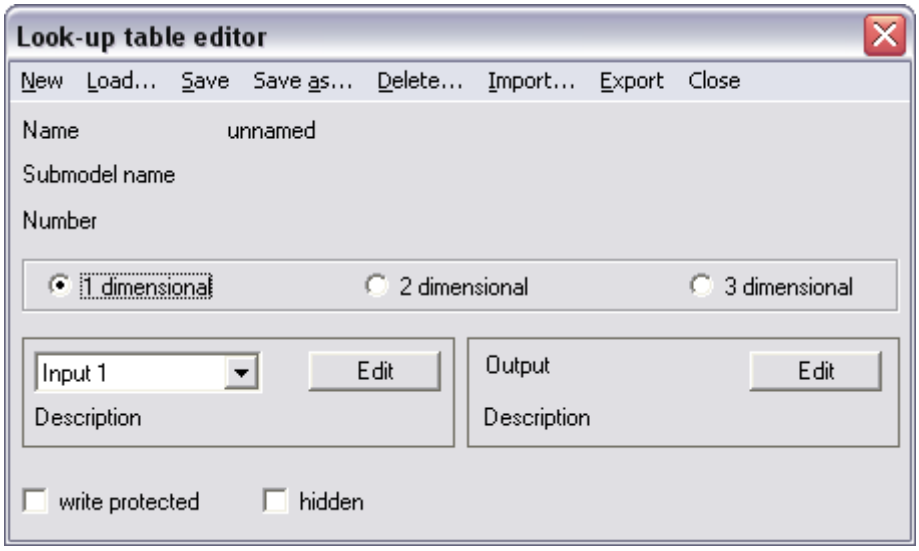
- If the project is exported via DSH*plus*-STC hidden parameters are included directly into the compiled model. Thus these parameters cannot be made visible in the target programs.

3.11 Look-up tables

In practice, a number of components are used whose physical behaviour can only be described with mathematical functions which take a lot of time. In order to integrate such components, however, in the simulation with less expenditure, DSH*plus* offers the possibility to characterize their physical behaviour by means of look-up tables.

Look-up tables form the link between one to three inputs and one output .

The editor for processing the look-up tables is called by means of **LOOK-UP TABLES** in the **PARAMETERS** menu.



Dialog ***Look-up table editor***

The **look-up table editor** is used for creating, changing, loading, deleting, exporting, importing and managing one, two, or three dimensional characteristics.

3.11.1 Creating of look-up tables

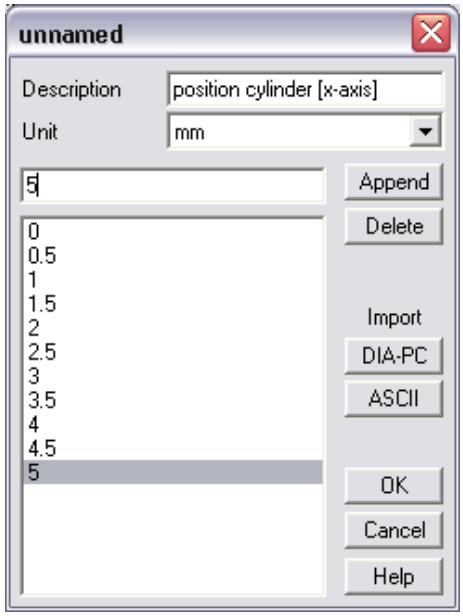
Proceed as follows in order to create look-up tables:

1. Select the dimension you want to use. The number of inputs (channels) increases according to the dimensionality selected.

2. From the drop down list select the input which is to be edited.

3. Open the dialog for setting the values and the unit of the input by means of the **EDIT** button on the left of the dialog.

 © FLUIDON® GmbH 2008



Edit input

4. Put in a description in the first two fields and select a unit for the input.

5. The third field is used for putting in the values. If you have put in a value in this field and click on **APPEND**, this value is added to the list. Repeat this step until all required values are put into the list.

   If you have inserted a wrong value, click onto this value and delete it by means of **DELETE**.

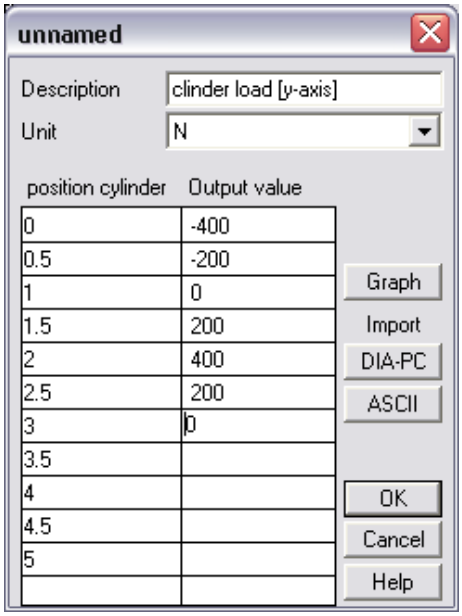
   The values are automatically sorted. Only inputs which monotonously increase can be created.

6. Processing of the input is terminated by **OK**.

7. Repeat steps two to six for every input dimension you use.

8. The parameterization of the Output channel is done by similar editor that opens after pressing the **EDIT** Button in the field **Output**.

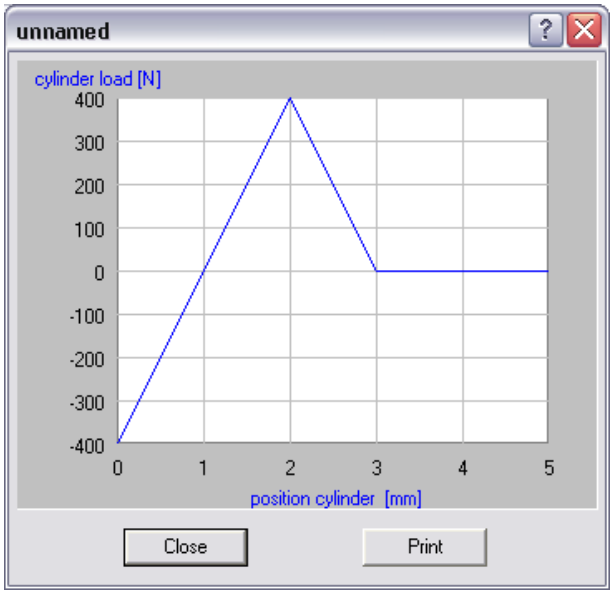
9. Put in a description in the first two fields and select a unit for the output.



Edit output

10. The values for the inputs are shown in the list. In order to allocate output values to the inputs, click on a line in the list and put in a value in the field for the output values. If you click on a line in the list then, the output value is taken over. By clicking on **OK,** editing of the output value as well as creating a look-up tables is terminated.

11. A graphical representation of an one-dimensional look-up table can be achieved by clicking on the Graph button.



Graphical representation of a one-dimensional look-up table

 © FLUIDON® GmbH 2008

12. In order to be able to use the look-up tables for parameterization of components, you have to save it in the **LOOK-UP TABLES** menu by means of **SAVE** or **SAVE AS**.

With multi-dimensional look-up tables, the data of the last input are combined with every value of the previous input. The same is done for every further input channel. A combination of $5^3 = 125$ output values is thus formed from a three-dimensional field with 5 input values in each dimension.

Note:

- The units can be selected in the editing dialogs for inputs and outputs even after the values have been put in without converting the values. This is very useful when values are imported.

3.11.1.1 Import measured values in look-up tables

If the values of the look-up tables inputs and outputs are available in the form of measured value files, they can be imported as **DIA-PC** and **ASCII** files via the corresponding keys in the dialogs for putting in the inputs and outputs. To do this, the following formats have to be made available:

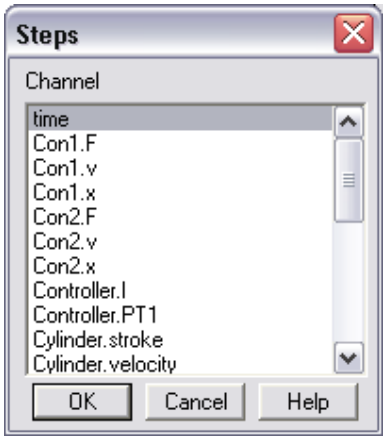
For loading the DIA-PC files, the data have to be stored in a 32 Bit format (R32) channel by channel or block by block. The number of values per channel is limited to 6000.

The individual data channels in the ASCII files have to be created in columns.

Import DIA-PC

A DIA-PC file can be loaded in a look-up tables by clicking on the control space **DIA-PC** in the editing window of the respective input and output in the look-up tables editor. A dialog window for selecting the file is displayed.

As soon as a file has been selected, another dialog field with the channel designations of the DIA-Header is displayed.

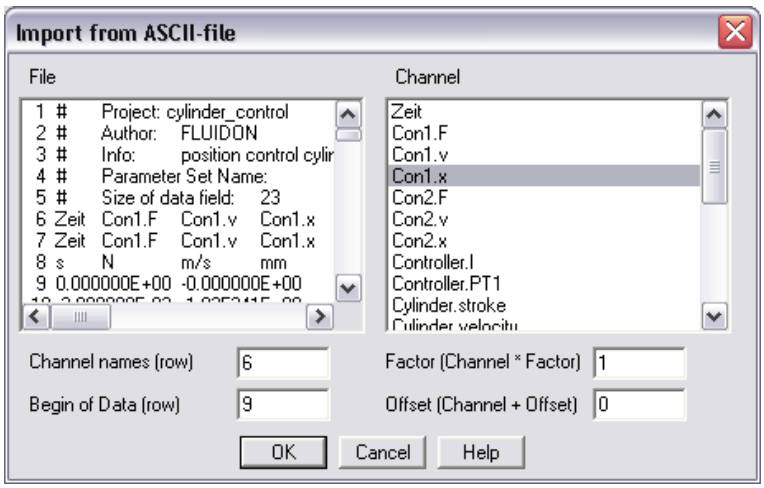


DIA Header

Here, it is possible to select the corresponding channel and to allocate it to the look-up tables channel by means of the control space **OK**. The values of the file are now displayed in the editor window of the look-up tables where they can be adapted at a later date.

Import ASCII

If you want to load an ASCII file, actuate the control space **ASCII** in the editing window of the respective input or output. A dialog window for selecting a file is displayed. As soon as an ASCII file has been chosen, the dialog **Import from ASCII file** opens.



Channel assignment of ASCII file

 © FLUIDON® GmbH 2008

The line in which the channel names are to be displayed is adjusted under **channel names (row)**. A blank is used as separation between the channel names which means that the names themselves are not to have any blanks. The list with the look-up tables names is automatically updated after a change.

The first line of the column the data which are used for the respective input or output, is adjusted under **Begin of Data (row)**.

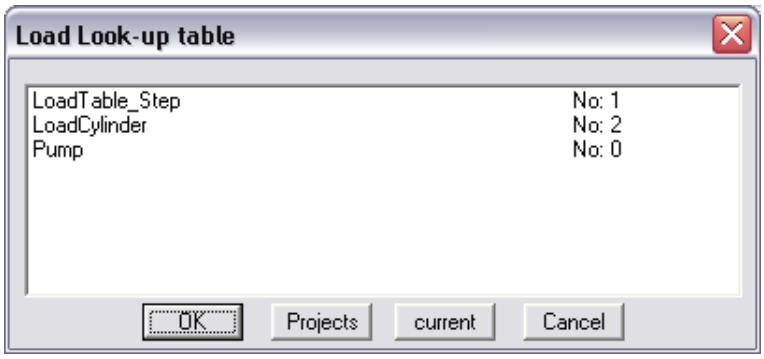
Under **Factor (Channel * Factor)**, a factor is put in with which all values of a channel are multiplied.

Under **Offset (Channel + Offset)**, a constant is put in which is added to every value of the channel.

When you click on **OK** in the dialog **Import from ASCII file**, the channel values are put in the respective input dialog. The values can be further processed here.

3.11.2 Load look-up tables

Already existing look-up tables can be loaded by means of the **LOAD** function in the menu look-up tables in the look-up tables editor and processed at a later date, if required.



*Dialog **Load look-up table***

A look-up table can be imported from another project. A project list can be called by means of the control space **PROJECTS**. If a project is selected from that list, the related look-up tables are listed.

The look-up tables of the current project are again displayed in the list by means of **CURRENT**.

In general, the look-up tables can be changed at any time. The procedure corresponds to the procedure used for creating new look-up tables. If the dimensional number is reduced by one channel, the channel with the highest number is no longer taken into account.

Note:

- Changes to a look-up table that is assigned to a component of the model are affecting directly the behaviour of this component during the next simulation run.

3.11.3 Save

The data which have been put in for the current look-up tables are saved by means of **SAVE** in the **LOOK-UP TABLES** menu. If a new look-up table is saved for the first time, **SAVE** corresponds to the function **SAVE AS**.

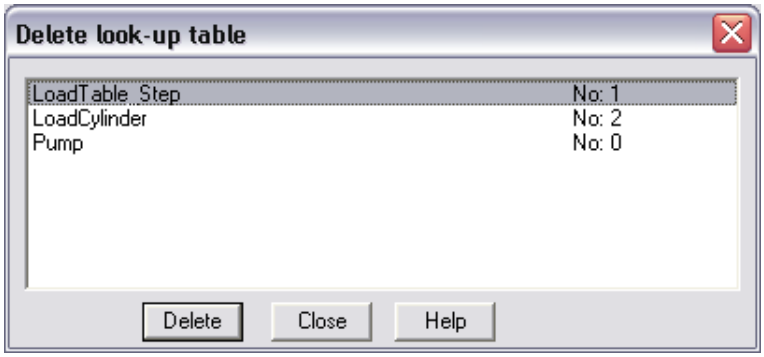
3.11.4 Save as

SAVE AS in the **LOOK-UP TABLES** menu offers the possibility to save changes carried out at the current look-up table under a new name.

3.11.5 Delete

Look-up tables belonging to the current project are deleted by means of **DELETE** in the **LOOK-UP TABLES** menu.

 © FLUIDON® GmbH 2008



*Dialog **Delete look-up table***

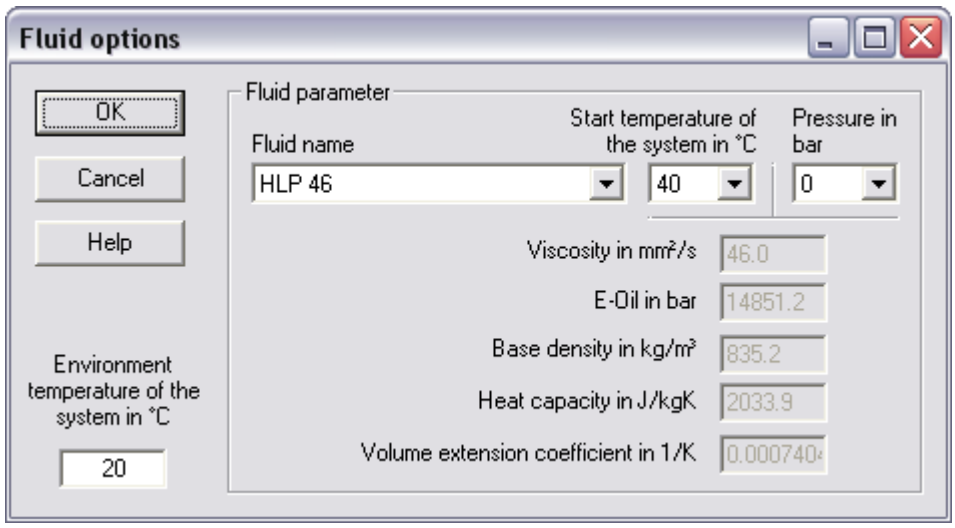
Mark the look-up tables which are to be deleted. Then click on **DELETE**.

3.12 Fluid options

In the dialog **Fluid options** the specifications of the employed oil type and the environment temperature of the system are adjusted.

In order to set the fluid options, proceed as follows:

1. Select **FLUID OPTIONS** in the **PARAMETER** menu.



*Dialog **Fluid options***

2. Make the desired settings for the oil specification and the environment temperature.

3. Click on **OK**.

Now, you have made the settings for oil type and surrounding temperature.

Note:

- The parameter **Start temperature of the system in °C** represents the tank temperature in a conventional hydraulic system. Physically modelled components that use fluid property values during the calculation do have a special parameter **TDIFFERENCE** to adjust the temperature difference between tank temperature and temperature at the component.

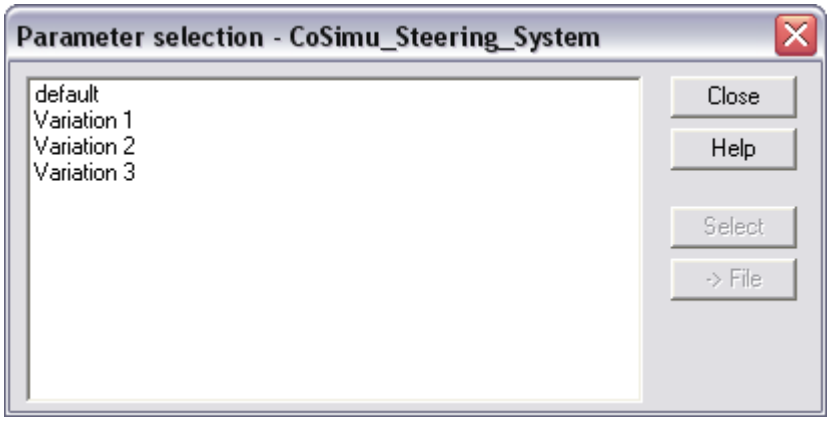
- A change of the pressure in the dialog only influences the values displayed in the dialog. If one of the values is varied, the specific oil values are adjusted automatically and the changes of the parameters during the simulation can be estimated.

3.13 Parameter selection

Via the function **PARAMATER SELECTION** in the **PARAMETER** menu individual parameters of a parameter set can be exported into a ASCII-file.

Please proceed as follows:

1. Choose **PARAMATER SELECTION** in the **PARAMETER** menu in order to open the dialog **Parameter selection**.

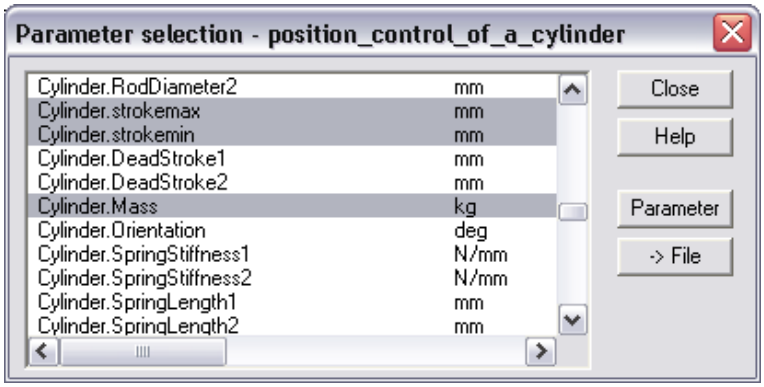


*Dialog **Parameter selection***

 © FLUIDON® GmbH 2008

2. Choose one or more parameter sets.

3. Click on **SELECT**.



*Dialog **Parameter selection***

4. Choose the parameters you want to export. Multiple selection is possible by means of the control- and shift keys.

5. Click on **-> FILE** in order to open the saving dialog.

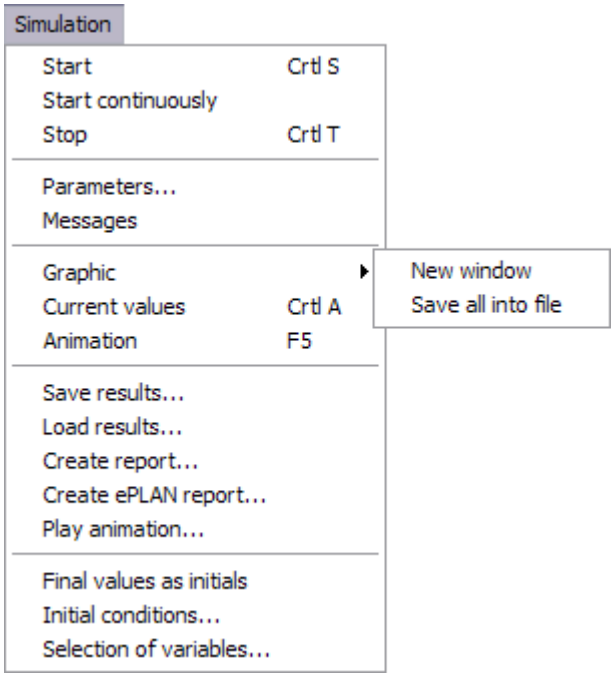
6. Put in a file name and choose a folder and a format to save as.

7. Click on **SAVE**.

Now, you have exported individual parameters. If you do not want to continue the exporting procedure, you can close the dialog **Parameter selection**.

4 Simulation

The functions which directly depend on the calculation of the dynamic system behaviour are summarized in the **SIMULATION** menu.



*Main menu **Simulation***

Here, the functions for starting and stopping the calculation, for setting the boundary conditions of the simulation calculation, for the graphic and numeric control of the simulation process and for saving the simulation data as well as a function with which the calculation of the final point of the previous simulation can be continued, are available

4.1 Start

Calculation of the dynamic behaviour of the model is started with **START** of the **SIMULATION** menu.

Progress of simulation can be seen at the progress bar in the status line. Process of the simulation can be seen in the graphic windows and the window **Current values** as these windows are currently up-dated during simulation.

4.2 Start continuously

The function **START CONTINUOUSLY** in the **SIMULATION** menu, initiates a simulation run in endless mode.

 © FLUIDON® GmbH 2008

In this case the simulation time specified in the simulation setting dialog is used as time interval for the x-axis scaling of the online graphic.

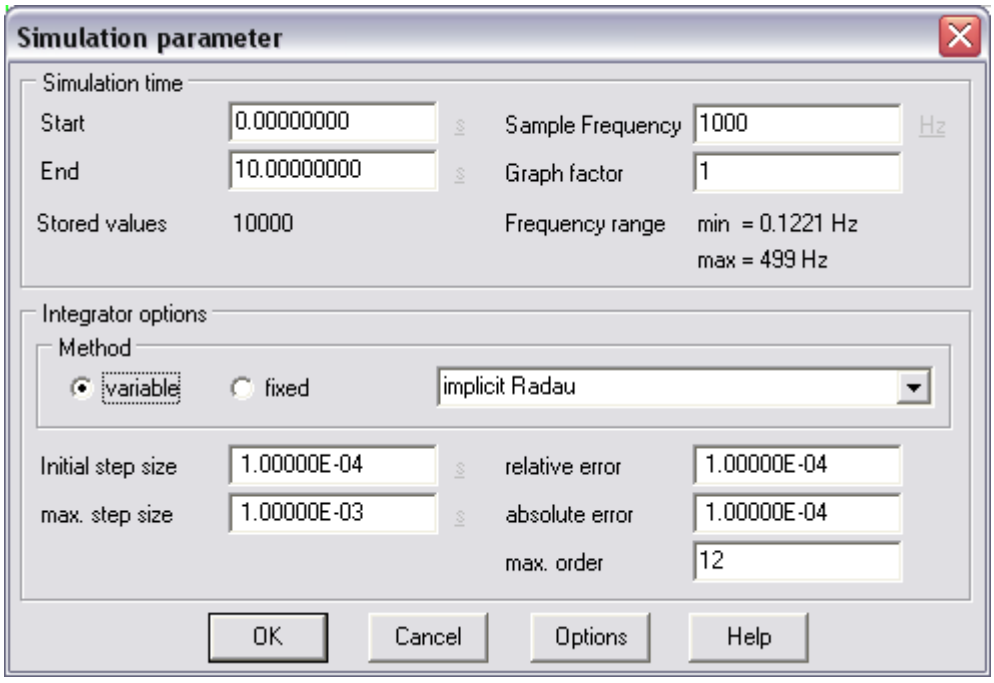
4.3 Stop

With **STOP** in the **SIMULATION** menu, calculation can be aborted before the adjusted simulation time has run out.

The values calculated up to the time of abortion, can be saved like the results of a simulation which has been completely carried out. Simulation results are saved in the **SIMULATION** menu under the menu point **SAVE RESULTS**.

4.4 Parameters

The boundary conditions of the numerical calculation can be adjusted in the **Simulation Parameters** dialog. This dialog is opened in the **SIMULATION** menu with **PARAMETERS**.



*Dialog **Simulation parameter***

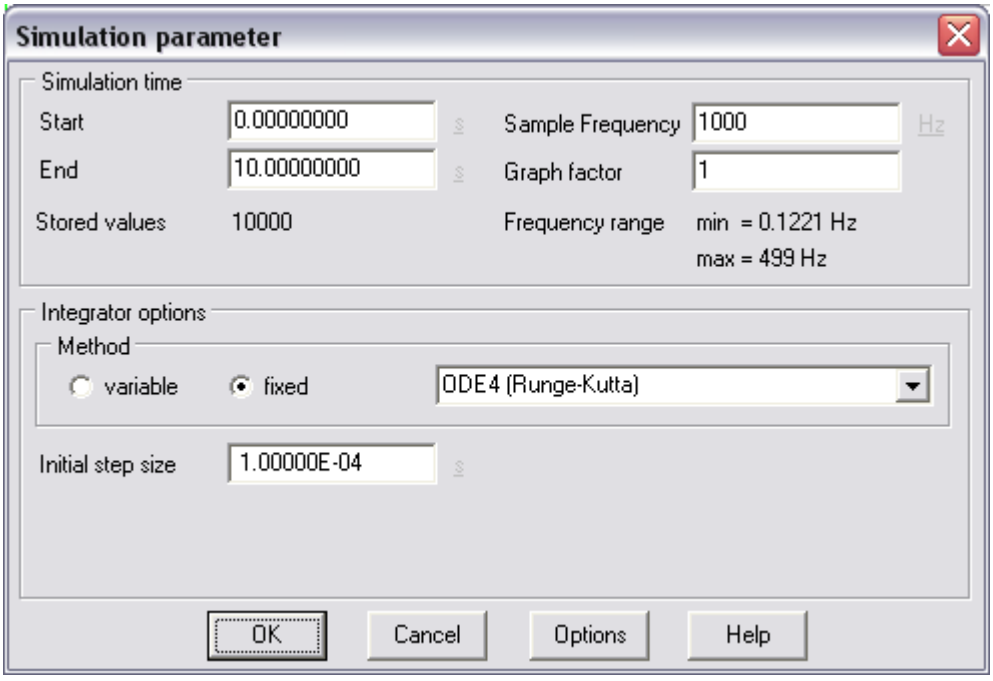
The simulation parameters define the boundary conditions of the calculation and the numeric procedure with which the simulation model is calculated.

The dialog **Parameters of Simulation** is divided into two sections:

- **Simulation time**

- **Integrator options**

Within the **Integrator options** you can choose between **variable** and **fixed** methods. The initial selection is **variable**. If, however, you choose **fixed**, the list of the simulation procedures in the drop-down list changes and the input possibilities within the calculating options are reduced to **Initial step size**.



Dialog for fixed simulation parameters

In the procedures with variable step size the step size is automatically adjusted to the model dynamic during simulation. The following methods are at your disposition:

- ODE23 (rk2(3) RK) parameter ID.-No. = 200

 © FLUIDON® GmbH 2008

- ODE45 (rk5(4)6m PD 45) parameter ID.-No. = 201
- Dormand-Prince 45 parameter ID.-No. = 202
- Euler implicit parameter ID.-No. = 203
- Rosenbrock implicit parameter ID.-No. = 204
- Radau implicit parameter ID.-No. = 205

In the procedures with fixed step size the step size is maintained constant throughout the simulation. The following methods are at your disposition:

- ODE1 (Euler-Cauchy) parameter ID.-No. = 100
- ODE2 (Heun) parameter ID.-No. = 101
- ODE3 (RK I) parameter ID.-No. = 102
- ODE4 (Runge-Kutta) parameter ID.-No. = 103
- ODE5 (Prince-Dormand) parameter ID.-No. = 104
- ODE4 (Runge-Kutta RCH) parameter ID.-No. = 105
- ODE5 (Prince-Dormand RCH) parameter ID.-No. = 106

The following parameters are determined in the dialog **simulation parameters**.

Parameters	Function
• Start	Starting time of simulation.
• Stop	Ending time of simulation.
• Sample frequency	Determines the time interval in which the result data is stored.
• Graph factor	Indicates which value of the indicated number of memory values is actually displayed in the graphics.

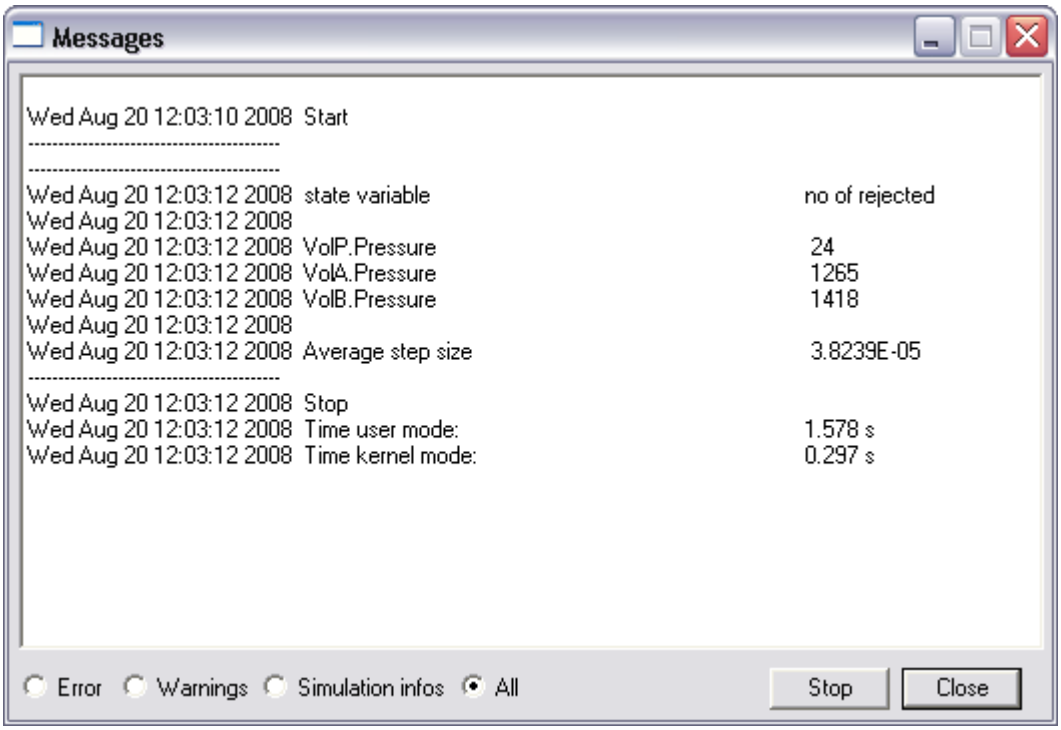
Parameters	Function
	If the parameter five is selected for example, only 200 of the indicated 1.000 values are put out.
• Initial step size	Time interval with which the integration procedure starts/works. With the variable step-size procedures, the interval is automatically varied during the simulation run.
• max. step size	Largest possible interval with the interval-controlled procedures with which calculation is made.
• min. step size	Smallest possible interval with the interval-controlled procedures with which calculation is made. This limit avoids that the procedure can get into an endless loop because of intervals which are too small.
• relative error	This evaluation criteria for the interval-controlled procedures states the relative faults which can be made at every simulation step.
• absolute error	This evaluation criteria for the interval-controlled procedures states the absolute faults which can be made at every simulation step.
• max. order	Determines the maximal order the integration procedure can select

For further information on:	Please see:
What are Simulation parameters	Chapter 4, 4.4 Parameters
Handling of the simulation parameters dialog	Chapter 3, 5.2 Set simulation parameters
Explanation of the integration procedures	Chapter 5, 1.2 Calculation method of DSH*plus*

© FLUIDON® GmbH 2008

4.5 Messages

The dialog **Messages** is opened with **MESSAGES** in the **SIMULATION** menu. In this dialog errors and warnings occurring with respect to the setting of the component parameters as well as the simulation process are recorded.



*Dialog **Messages***

At the bottom of the dialog it is possible to determine whether errors, warnings, and the simulation process are displayed singly or simultaneously.

Besides information about the model, the type of error or the type of the warning, the record also reveals which component the respective message relates to. On the drawing area the name of a component featuring an error is presented in red. The name of a component with a warning at issue is presented in dark violet.

4.6 Graphic

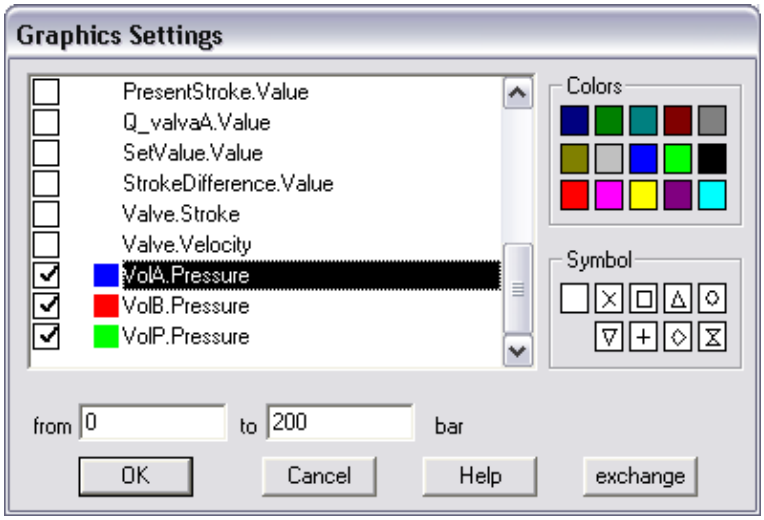
The **Graphic** window in which the simulation data are graphically put out during calculation is opened by means of **GRAPHICS** in the **SIMULATION** menu. Up to ten **Graphics** can be opened at the same time.

For further information on:	Please see:
Starting values	Chapter 4, 4.15 Initial conditions
Set starting values	Chapter 3, 5.1 Set parameters and starting values of the nodes and components

4.6.1 Selection of graphs

Selection of the graphs displayed in the simulation graphics is made with the following steps:

1. By double clicking on the grid in the graphics window, the dialog **GRAPHICS SETTINGS** is opened.



*Dialog **Graphics settings***

2. Under **GRAPHICS SETTINGS** mark the variables which are to be shown.

 © FLUIDON® GmbH 2008

3. In order to distinguish them better, the variables are shown in different colours which are preset in series. It is also possible to allocate other colours to these variables by selecting another colour under **Colours**. A selected colour is allocated to the variable which is marked.

4. In addition, markers can be allocated to the variables. They can be selected under **Symbol**. A selected marker is allocated to the variable which is marked.

5. Certain value ranges are allocated in series to the individual variables. These value ranges can be adapted to your requirements in the input fields **from** and **to**. In these fields, the variable which is marked is always processed.

6. Click on **OK** when you have made all adjustments.

Your selection and your settings are taken over for the individual simulation graphics. For every graph of a variable, the names and the value range are shown in the same colour in the simulation graphics window.

If you want to revise the settings of a specific variable, double click on these settings in the caption. By doing this, the **Graphics settings** dialog is opened. In this dialog, the variable which has previously been selected by double clicking, is automatically marked so that this variable can be processed immediately.

Note:

- The time controlled resolution of the graphical display depends on the setting **factor graphic**. This setting is determined under **PARAMETERS** in the **SIMULATION** menu.

- The units of the variables cannot be changed by means of the **Graphics** window. If a unit is to be allocated to a variable or a variable is to be displayed in another unit, this change has to be done under **INITIAL CONDITIONS** in the **SIMULATION** menu or in the parameterization dialog of the component.

- Presetting of the automatically chosen colours is determined on the **Graphics options** page of the **Options** dialog.

4.6.2 Adapt simulation graphics

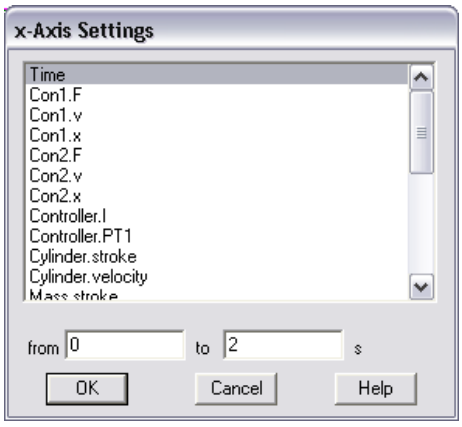
The dynamic behaviour of the model can be controlled and judged during simulation by means of the simulation graphics.

For an optimum control of the individually relevant data, DSH*plus* offers a number of possibilities to adapt the **Graphics**:

- Zoom: In order to zoom a section of the graphics, a frame is to be drawn round the corresponding section by means of the mouse. After the key is released, the selected section is enlarged. This step can be repeated several times.

- By means of the Shift-key in combination with the left mouse button the zoomed section can shifted in the graphic window. Pushing down the ctrl-key the y-axis will be kept constant and only the x-axis is scrolled.

- Selection of a different X-axis: By double-clicking with the left mouse-key directly beneath the X-axis the dialog **X-Axis Settings** is opened.

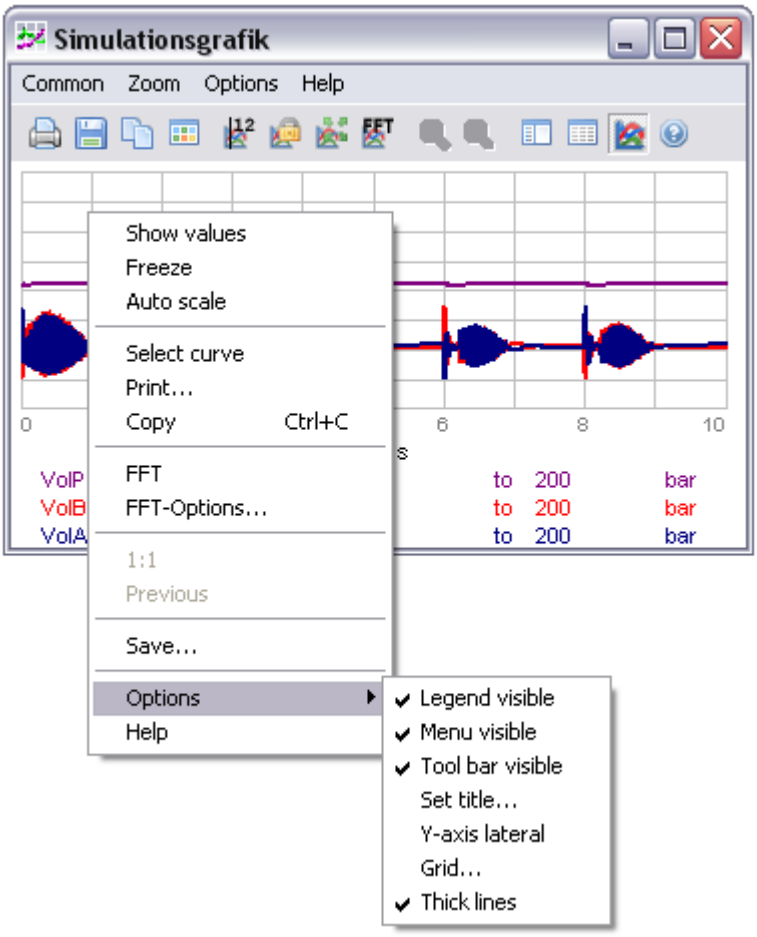


*Dialog **x-Axis Settings***

- In this dialog a different X-axis can be selected.

 © FLUIDON® GmbH 2008

All other adoption possibilities have to be carried out by means of the menus **Common**, **Zoom** and **Options** or by the context menu.



Context menu of the graphic window

The following functions are available:

- **SHOW VALUES** is used for displaying and tracking the values along the time axis. If this option is selected, a point on the time axis can be fixed by the mouse for which the values of the selected variables are displayed.

- **FREEZE** effects that the graph of the last simulation is not deleted by the following simulation but is still displayed. Thus, the direct comparison of two simulations is possible.

- **AUTO SCALE** effects that the indicated area is adjusted to the maximal values during simulation so that the total graph is always visible.

- **SELECT CURVE** opens the graphic settings window.

- **PRINT** is used for printing the active simulation graphic.

- **COPY** copies the graphic into the clipboard.

- **SAVE** allows to save the graphic as Windows Meta File (*.WMF).

- **FFT** performs a Fast-Fourier-Transformation.

- **FFT-OPTIONS** opens the dialog for the processing of the Fast-Fourier-Transformation settings.

- **1:1** cancels the zoom. The initial size is restored.

- **PREVIOUS** cancels the zoom. The zoom selected before is restored.

- **LEGEND** switches the legend on or off. The section for the grid is enlarged or reduced.

- **TITLE BAR** switches the title line on or off. The section for the grid is enlarged or reduced.

- **MENU** switches the menu on and off

- **SET TITLE…** opens a dialog to assign a title to the graphic window

- **Y-AXIS LATERAL** is used to display the y-axis names and scaling right and left at the y-axis in the graphic. In standard setting y-axis names and scaling is displayed below the graphic windows.

- **GRID…** opens a dialog to adjust the grid of the graphic window drawing area. Permanent changes to the grid can be made on the **Graphics options** page of the **Options** dialog.

 © FLUIDON® GmbH 2008

- **THICK LINES** increases the thickness of the displayed curves.

For further information on:	Please see:
Change units of the variables	Chapter 3, 5.1 Set parameters and starting values of the nodes and components Chapter 4, 4.15 Initial conditions

4.6.3 Copy simulation graphics into the clipboard

Simulation graphics can be inserted into graphics or text processing programs by means of the clipboard. Graphics can be copied into the clipboard in two different ways:

1. Selection of **COPY** from the context menu of the simulation graphics,

2. By means of the key combination **Crtl-C,** the active window is copied into the clipboard.

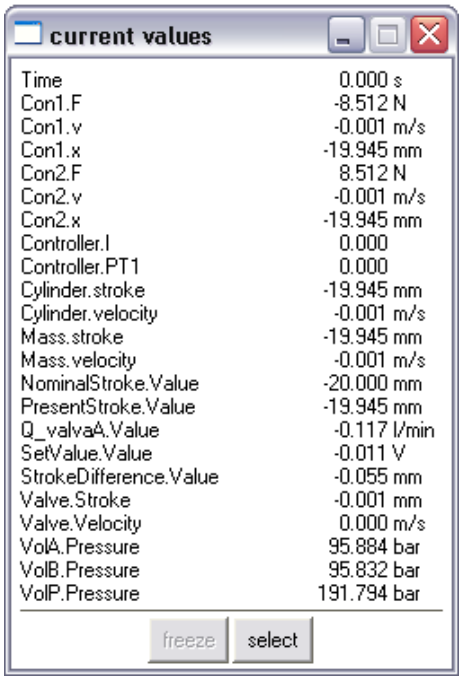
With the key combination **Crtl-V** or **INSERT,** the graphics is inserted into the target applications.

4.7 Current values

The window **current values** in which the simulation data are numerically shown during calculation, is opened by means of **CURRENT VALUES** in the **SIMULATION** menu.

Proceed as follows in order to call the **current values** window and to make a selection of variables:

1. Click on the menu point **CURRENT VALUES** in the **SIMULATION** menu in order to open the **current values** window.



*Display of **current values***

If you want to observe all of the values, no further settings are required here and the following steps can be dispensed with.

2. If you only want to observe some of the values, click on **SELECT** in the **current values** window. The **Selection** dialog is thus opened.

   Select the variables those current values you want to have displayed. Selection of some or of many variables is accelerated by means of the keys **Select none** and **Select all**.

© FLUIDON® GmbH 2008



***Selection** of the sizes displayed*

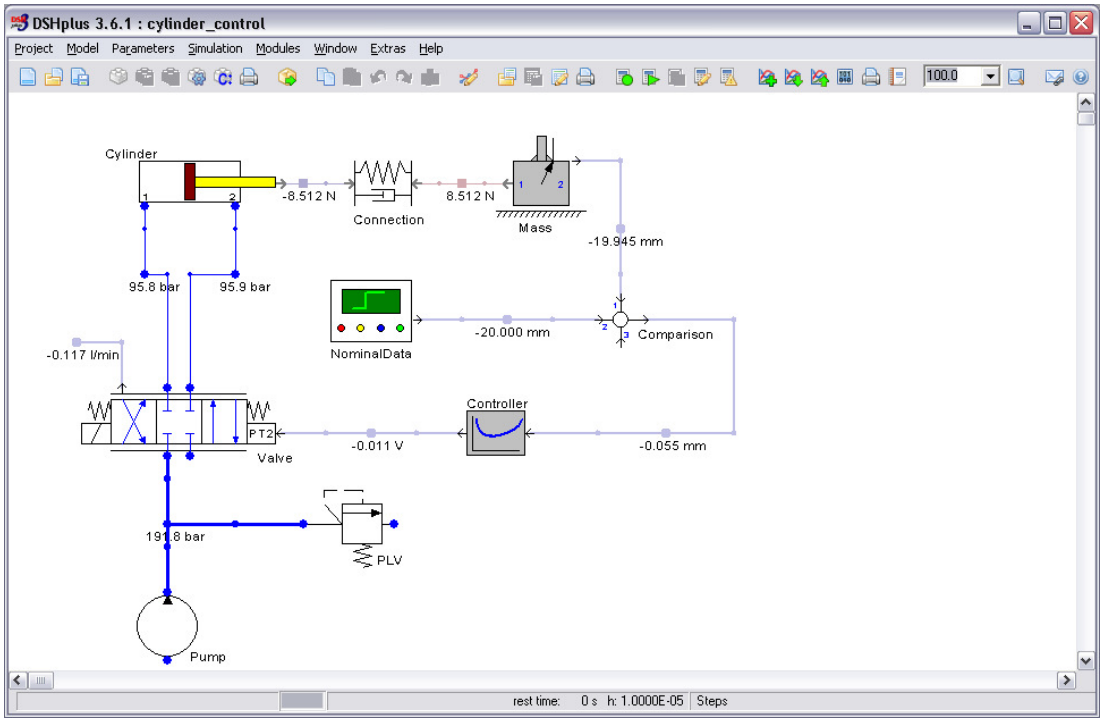
Click on **OK** so that your selection in the **current values** window is used.

3. It is possible to stop up-dating the values during calculation. Click on **FREEZE** in the **current values** window in order to freeze the values. Calculation is, however, not stopped by this action. The key name changes into **CONTINUE**.

   Click on **CONTINUE** in order to display the current values again.

4.8 Animation

With **ANIMATION** it is possible to switch on and off the animation of the signal lines and the additional information displayed at the nodes. The initial settings for the animation can be made on the **Graphic options** and the **Other options** page of the **Options** dialog.



Model with animated value and thickness

Line animation is intended to visualize the signal flow during the simulation run in the following way.

- For fluid power variables the thickness of the line indicates the pressure level of the associated fluid power node.

- For mechanical variables the thickness of the line indicates the force or torque level of the associated mechanical node. For positive values the grey line color is changing slightly into red and for negative values it is changing slightly into blue.

- For signal variables the thickness of the line indicates the value of the associated process-variable node. For positive values the green line color is changing slightly into red and for negative values it is changing slightly into blue.

4.9 Save results

Under the point **SAVE RESULTS** in the **SIMULATION** menu the calculated simulation data can be saved. The **Result save Dialog** is opened. In

 © FLUIDON® GmbH 2008

this dialog the storage place, file name and type of file can be set. The following type are at your disposition:

- DiaPC (*.dat)
- Text (*.dat)
- Excel (*.xls)
- TextSI (*.txt)
- Excel SI (*.xls)
- Matlab (*.mat)
- Text with Blanks (*.txt)
- Text with cont. No. (*.txt)

A pre-selection of the favourite result format can be made in the **Other options** page on the **Options** dialog.

For further information on:	Please see:
Use and management of simulation results	Chapter 4, 4.10 Load results

4.10 Load results

Stored simulation results can be reloaded to the simulation model for further analysis with the post processing tools, for animation during a result presentation or simply as frozen reference for the next simulation run.

The following type are at your disposition:

- DiaPC (*.dat)
- Text (*.dat)

- TextSI (*.txt)

4.11 Create report

By means of the **CREATE REPORT** function a HTML-based report is generated from the presently active simulation results. The report is based on a HTML-template that is located in the standard library folder and represents the following information.

- **Project Title**: All information that is assigned to the **Info** dialog of the **Project** menu is printed into this section.

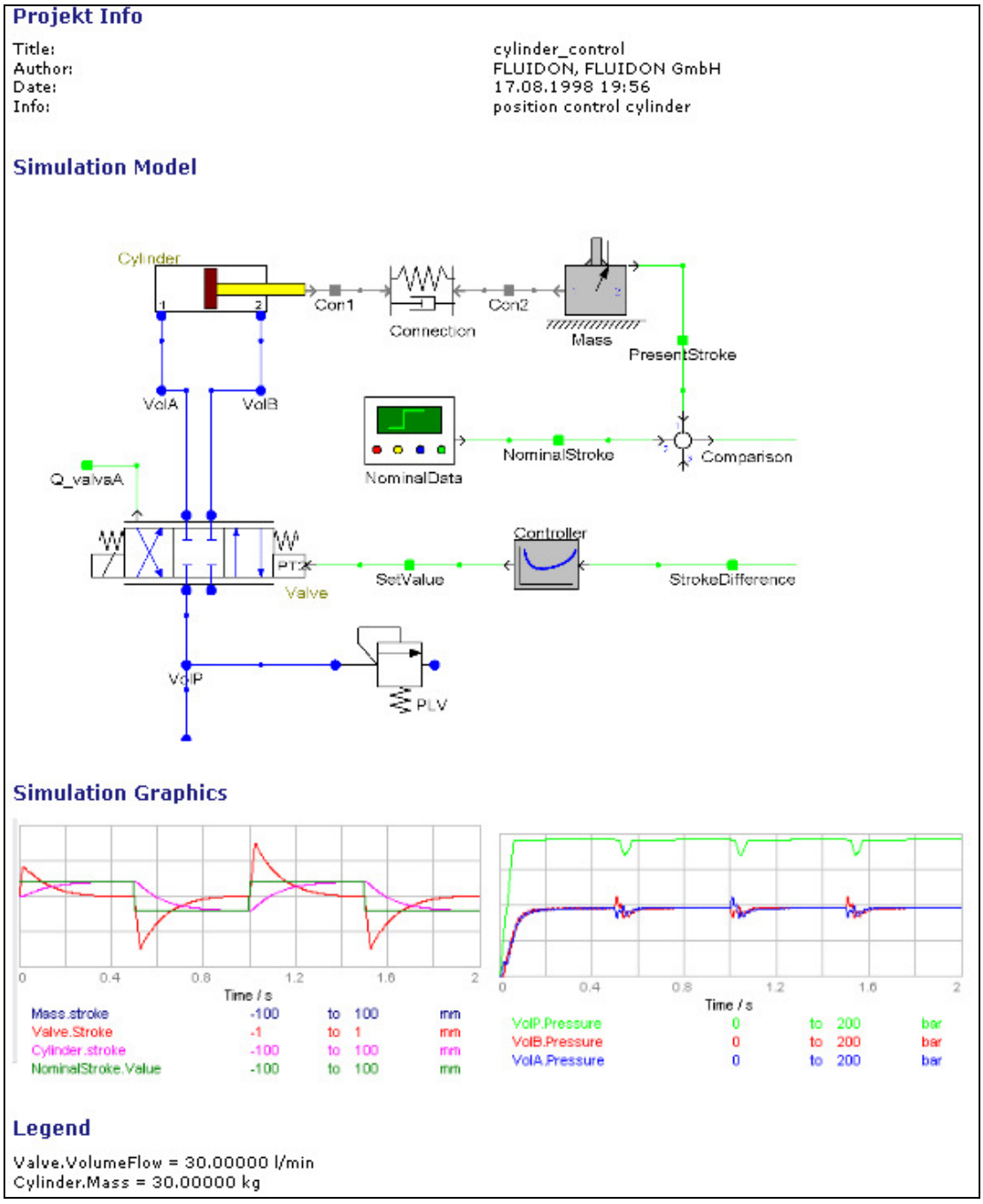
- **Simulation Model**: A screen-shot of the simulation model is shown in this section.

- **Simulation Graphics**: A screen-shot for each open simulation graphic is shown in this section. Please make sure that the windows do not overlap before making the report!

- **Legend**: Parameters names and values of parameters that are indicated as **FAVORITE PARAMETERS** are printed into this section.

After creation the report is directly displayed with the standard browser.

 © FLUIDON® GmbH 2008



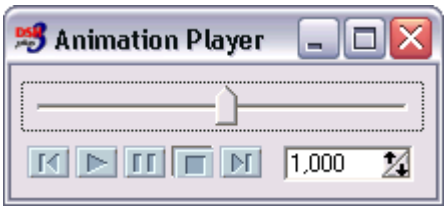
Report of a Simulation

4.12 Create ePLAN report

Similar functionality as **Create report** but the report is not opened. Instead the system tries to open the EPLAN-project linked to the simulation model.

4.13 Play animation

By means of the **PLAY ANIMATION** function the presently active or stored simulation result are represented. The animation is done on real-time basis but with a scaling factor it is possible to adjust a time lapse in order to quicken or slow down the representation.



Dialog Animation Player

A sliding bar is available to move through the animation manually.

4.14 Final values as initials

Should simulation be continued beyond the adjusted period of time or a steady state meant to be used as starting condition, the menu point **FINAL VALUES AS INITIALS** can be selected under **SIMULATION**. This means that the next simulation starts with the final values and not with the starting values of the previous simulation. The time interval of the subsequent simulation is as long as that of the previous one.

Note:

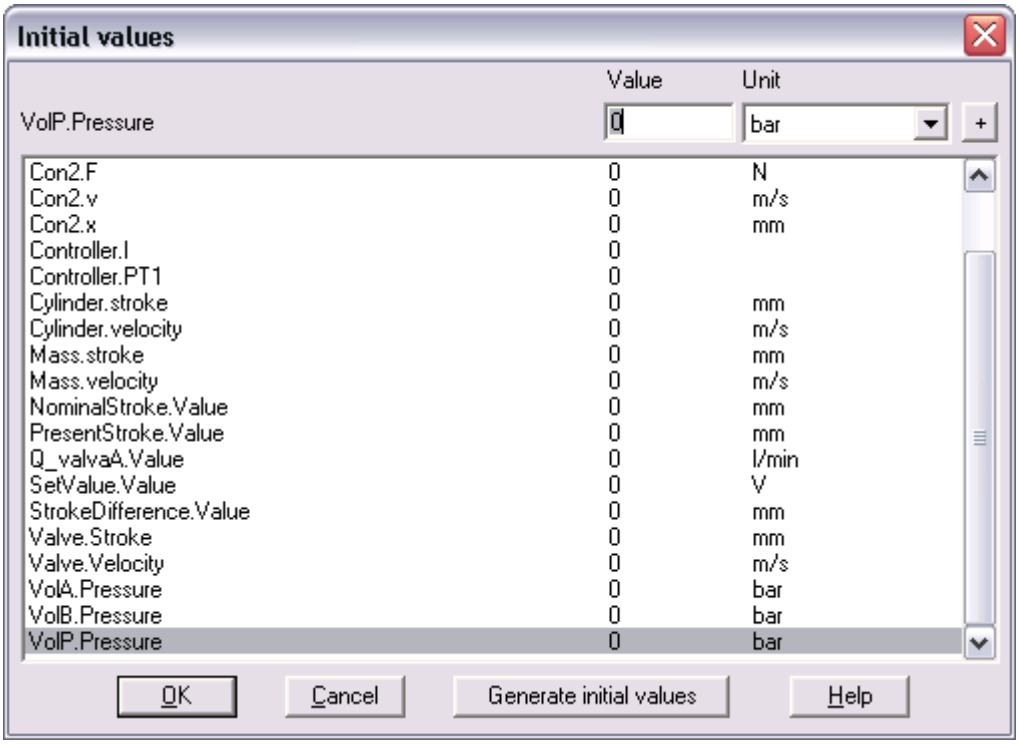
- When the end time of the simulation becomes the starting time, this can influence the time-dependent functions of function generators for example. With some elements which refer to values before the simulation end, the state cannot be stored completely and thus cannot be used as starting state (e.g. idle time).

For further information on:	Please see:
Starting values	Chapter 4, 4.15 Initial conditions Chapter 3, 5.1 Set parameters and starting values of the nodes and components

 © FLUIDON® GmbH 2008

4.15 Initial conditions

The starting values of variables cannot be set in the individual component dialogs only but also under the menu point **INITIAL CONDITIONS** in the **SIMULATION** menu.



*Dialog **Initial values***

For setting the starting values of the variables in the **INITIAL VALUES** menu, proceed as follows:

1. Click on the menu point **INITIAL CONDITIONS** in the **SIMULATION** menu in order to open the **INITIAL VALUES** dialog.

2. Click on a variable in the list. The name of the selected variable is displayed in the display field.

3. Click on the arrow of the drop down list for selecting the desired unit and then on the desired unit.

   If the desired unit is not in the list, click on the "**+**"**-**key. All the units made available by DSH*plus* are now displayed in the drop down list.

4. Put in the initial value in the input field **Value**.

   Only put in the value after having adjusted the unit. Otherwise the value is converted into the new unit.

5. Repeat steps two to four until all variables have the desired unit and the desired value.

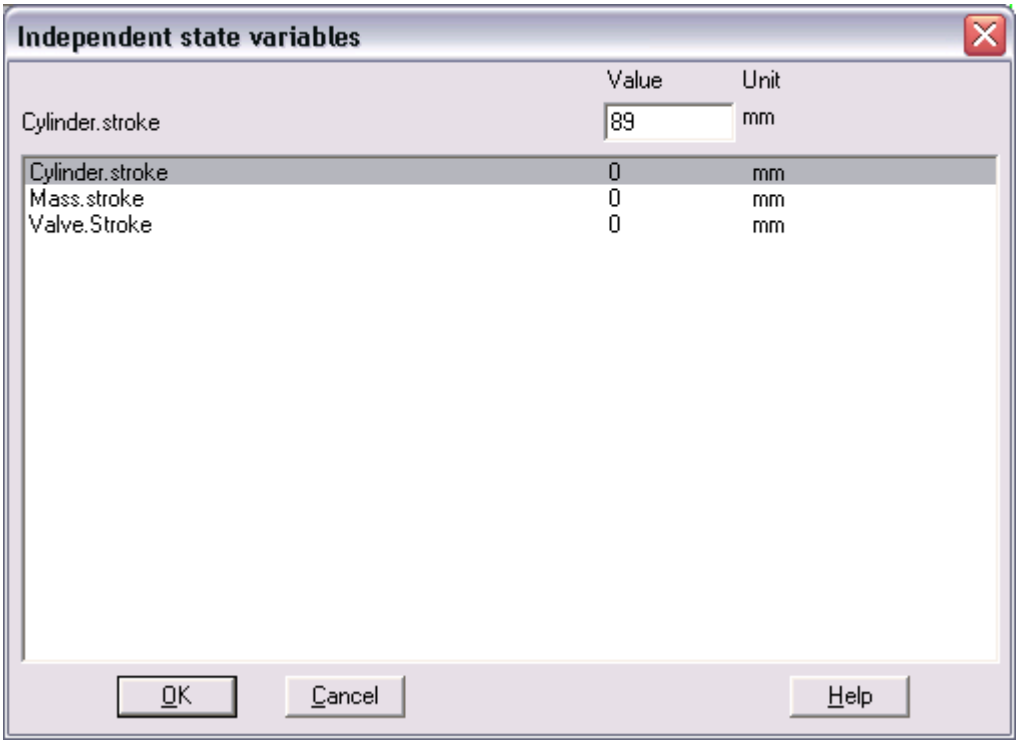
6. Click on OK and close the parameterization dialog.

The initial values are saved together with the parameter set in which they have been assigned.

DSH*plus* also offers the possibility to calculate initial values automatically. In order to do this, proceed as follows:

1. Click on the button **GENERATE START VALUES** in the dialog **Initial values**. In order to generate initial values numerically the values of the independent state variable of the system must be set manually. DSH*plus* automatically recognizes the independent variables of the system. They are displayed in the dialog **Independent state variables.**

 © FLUIDON® GmbH 2008

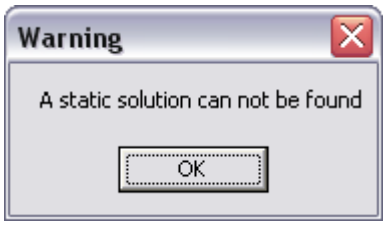


Dialog ***Independent state variables***

2. Put in adequate values for the independent state variables.

3. Click on **OK**.

If DSH*plus* is able to create adequate stating values these are now displayed in the dialog initial values. Otherwise the message



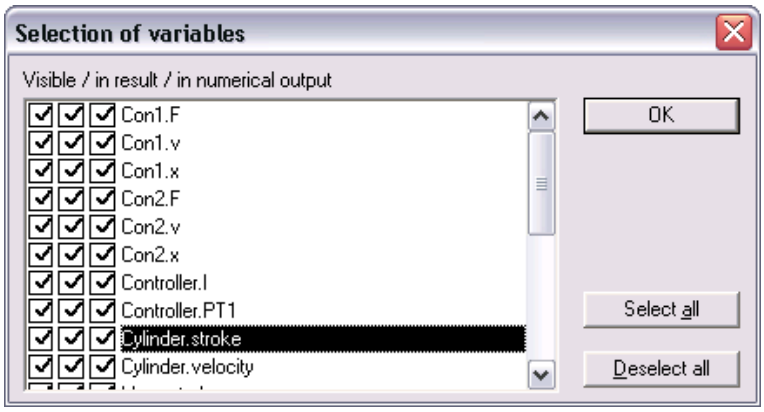
is put out.

Note:

- The variable values, displayed in the simulation graphics, are displayed in the unit which has been allocated to the starting value of the variables.

4.16 Selection of variables

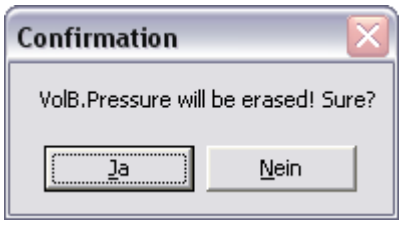
By means of the **SELECTION OF VARIABLES** dialog it is possible to determine whether a variable is available to be displayed in the online graphic, in the current values dialog, or stored into the result file.



*Dialog **Selection of variables***

Especially at large simulation model with a high sampling rate it is recommended to reduce the number of saved variables.

If a variable, that is already used in an online graphic, is deselected by the **SELECTION OF VARIABLES** dialog, the system requires a confirmation in order to preserve the current graphic layout.

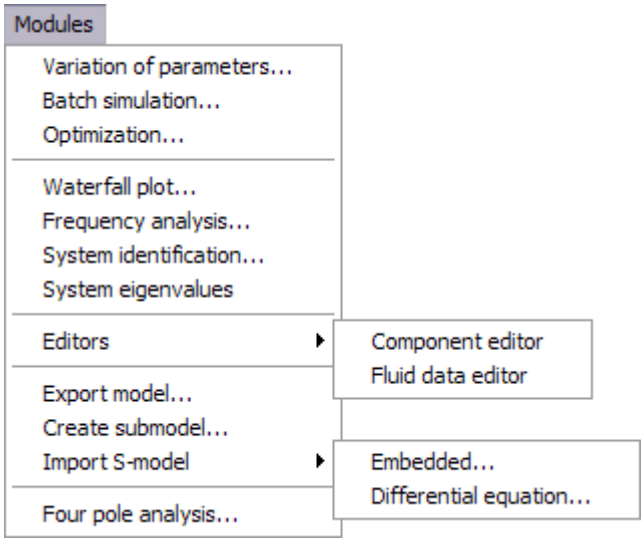


Warning dialog at variable selection

 © FLUIDON® GmbH 2008

5 Modules

The menu **MODULES** contains the DSH*plus*-Virtual-Engineering-Lab modules, a number of tools to analyze simulated and measured signals in the time and frequency domain.



Main menu ***Modules***

In addition to these functionalities this menu also contains the DSH*plus*-STC-model export, the S-Function import and the sub model functionality.

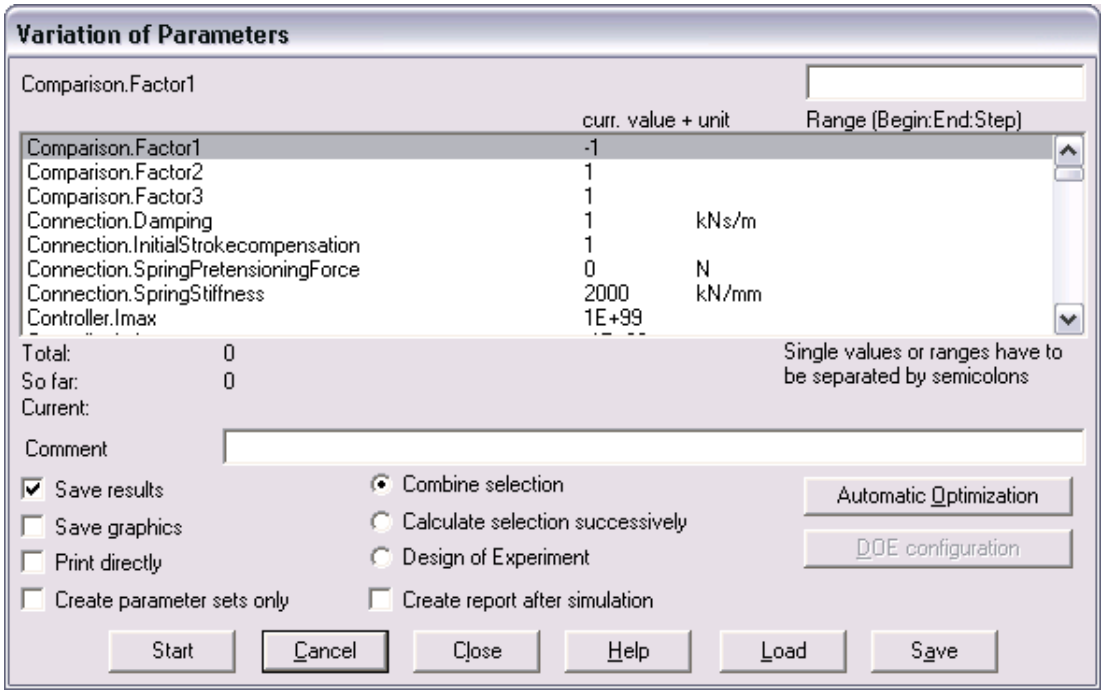
The availability of the listed modules is subject to the end-users licence condition. This online program manual is only intended to provide a brief survey of the module functionality. For further information it is referred to the corresponding module manual.

5.1 Variation of parameters

The module **VARIATION OF PARAMETERS** is one of the core modules of the DSH*plus*-Virtual-Engineering-Lab.

The dialog contains all the systems component parameters in alphabetical order with their current value. For each parameter a variation interval or explicit parameter values can be determined. Several meth-

ods are available to combine the chosen parameters. Even an execution of DoE-matrixes is possible.

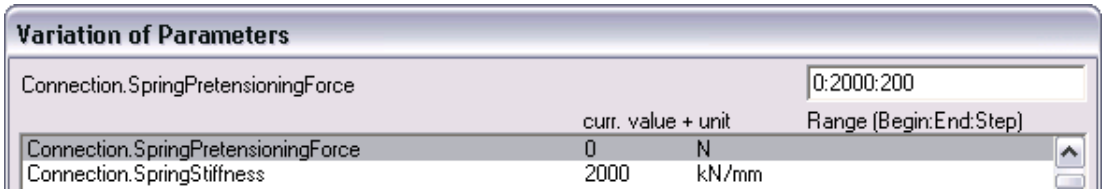


*Dialog **Variation of Parameters***

The module is designed to evaluate the operating range of systems, to test abstract (over dimensioned) demands, and test new designs with changed geometric or physical characteristics (spring stiffness, bore diameter, etc.).

Only a few steps are required, to run a variation.

Mark a parameter in the list and set the range in the input field **Range (Begin:End:Step)** to vary**.** The number of the parameters is not limited, but should but selected wisely for sensible calculations.



Parameter selection and input of the values/ranges

The actual parameter value is shown in the middle column of the main window.

 © FLUIDON® GmbH 2008

The following combination of values and ranges is possible:

- Single values are separated with a semicolon (1;3;5, …). The decimal places are separated with a point (1.5; 2.6;…).

- Value ranges are separated with a double dot (100:250:50). The first value indicates the starting value of the parameter, the middle value indicates the end value, and the third value represents the step size of the variation.

With **Save**, the prepared configuration can be saved into a *.pmo-file, which is saved together with the simulation project. **Load** invokes a file open dialog to reload the settings.

The following settings are available:

- **Save results** stores the result of each simulation run. An abbreviation for each modified parameter will be part of the file name.

- **Save graphic** stores the online graphic after each simulation run in Windows Meta File format. An abbreviation for each modified parameter will be part of the file name.

- **Print directly** generates a print-out of the online graphic windows according to the setting on the **Printing** page of the **Options** dialog.

- **Create parameter sets only** generates parameter sets for all specified modifications. The simulation will not start. It is intended to use the parameter sets in the ***Batch simulation*** mode.

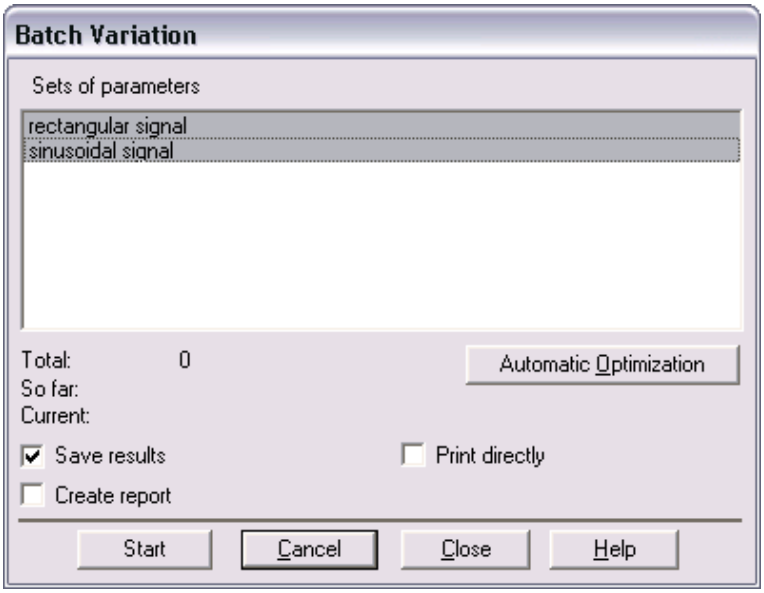
- **Create report after simulation** generates a html-bases report page for each simulation. Model and online-graphic as well as favourite parameters and model information are included into the report. More detailed information is available in the section 4.11 Create report.

The button **Automatic Optimization** loads a configuration file, generated by the **OPTIMIZATION** module, that enables an optimization for each parameter set run by the **VARIATION OF PARAMETERS** module.

5.2 Batch simulation

The **BATCH SIMULATION** is designed to automatically process several pre-defined parameter sets in a row.

To start the batch run the user merely selects the parameter sets and presses the start button. By means of the **BATCH SIMULATION** it is possible to run the simulation overnight and directly start with the analysis the next morning.



*Dialog **Batch Variation***

The following settings are available:

- **Save results** stores the result of each simulation run. An abbreviation for each modified parameter will be part of the file name.

- **Print directly** generates a print-out of the online graphic windows according to the setting on the **Printing** page of the **Options** dialog.

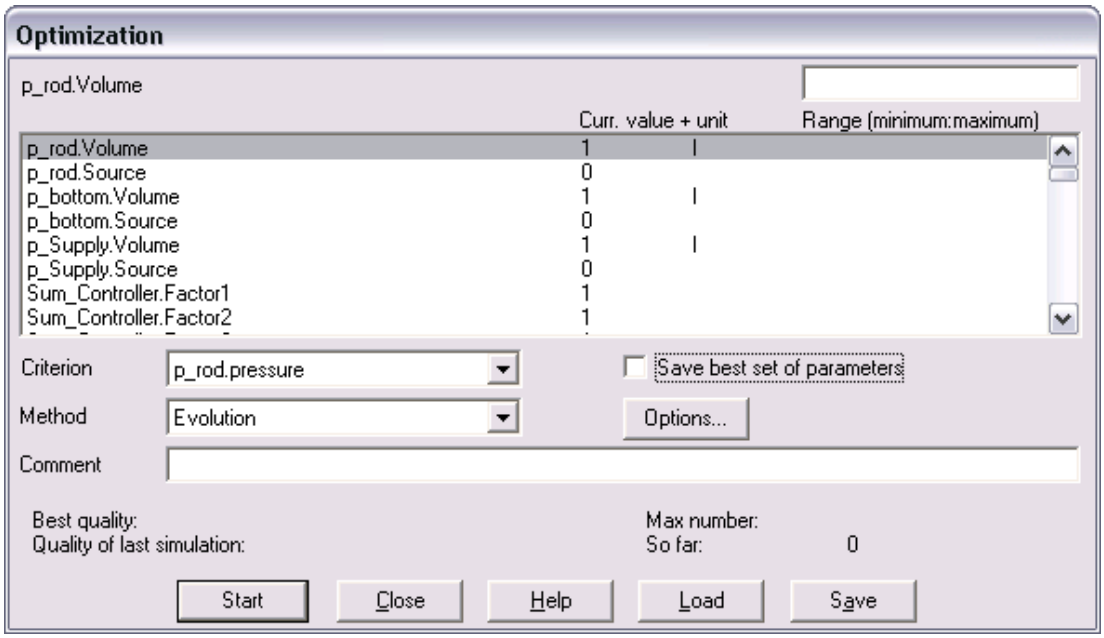
 © FLUIDON® GmbH 2008

- **Create report after simulation** generates a html-bases report page for each simulation. Model and online-graphic as well as favourite parameters and model information are included into the report. More detailed information is available in the section 4.11 Create report.

The button **Automatic Optimization** loads a configuration file, generated by the **OPTIMIZATION** module, that enables an optimization for each parameter set run by the **BATCH SIMULATION**

5.3 Optimization

The module **OPTIMIZATION** is designed to optimize one or more parameters of the simulation model in order to minimize a user defined target function.



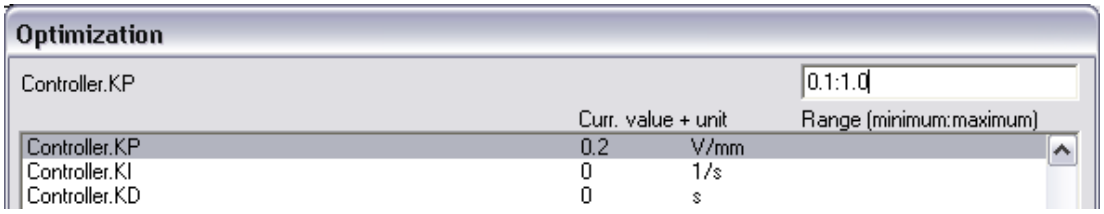
*Dialog **Optimization***

The most important precondition for an optimization is the proper quality criterion in the simulation model. Therefore, the control elements library provides under the sub-section mathematical functions the component **QualityCriterion**. This elements includes four standard integral algorithm to calculate a numerical value, which represents the quality value of the model and that is minimized by the optimization algorithms.

For the use of the component itself, please have a look at the component help.

The **OPTIMIZATION** module offers two different optimization methods. An evolution method and gradient search algorithm (Hooke and Jeeves).

The parameter range, available for the modification, is limited by a minimum and a maximum value. It is possible to optimize one or more parameters, if necessary. To set the range, chose the parameter in the list and define its range in the field **Range** with **minimum** and **maximum** value separated by a colon.



Range setting

The defined configuration can be saved as ***.OMO** file together with the simulation project.

The calculated best parameter values are saved temporarily. It has to be saved as parameter after the optimization.

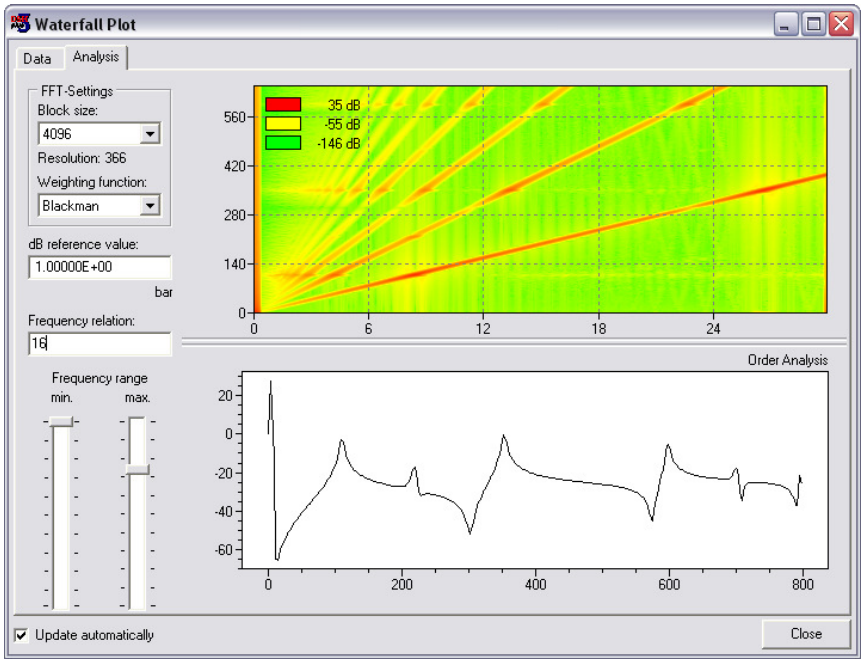
With **Options** you can set the method specific parameters to vary the algorithm precision and number of iterations respectively the number of parents, descendants and the maximum number of generations.

An additional **COMMENT** helps to identify the saved results.

5.4 Waterfall plot

The **WATERFALL PLOT** is intended to support an order-analysis of measured or simulated values. Aim is to visualize the change of eigenfrequencies (natural frequencies) of a system due to the change of a certain input signal (usually time or rotational speed).

 © FLUIDON® GmbH 2008

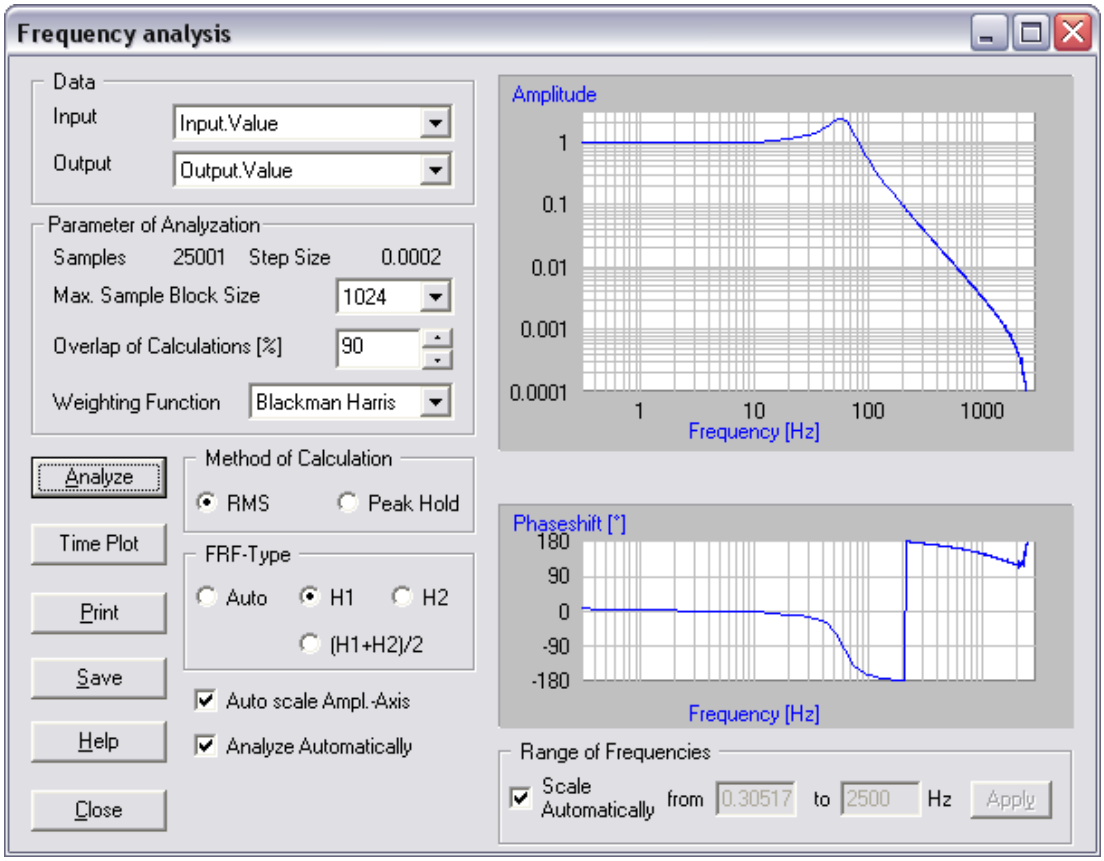


Dialog ***Waterfall Plot***

In the current version there is only a 2D-plot available (analysis page). X-axis is the selected input signal, Y-axis is the frequency, and the color change is symbolizing the amplitude height. In the lower part of the analysis page there is a graphic that is representing a cut through the 2D graphic and that can be used to isolate the amplitude range of a certain system order in respect to the input signal.

5.5 Frequency analysis

The **FREQUENCY ANALYSIS** module computes the frequency response of a system. The resulting transfer function is subsequently displayed in a bode diagram depicting amplitude and phase shift.



Dialog ***Frequency analysis***

Inputs or output may be any state or variable in the model or external data such as measurements of a real system that can be easily imported into **DSH*plus***.

By means of two boundary lines a data interval suitable for calculation can be chosen.

Adjusting the parameter **Max. Sample Block Size** and **Overlap of Calculations** the **FREQUENCY ANALYSIS** is able to perform a sliding analysis of the data.

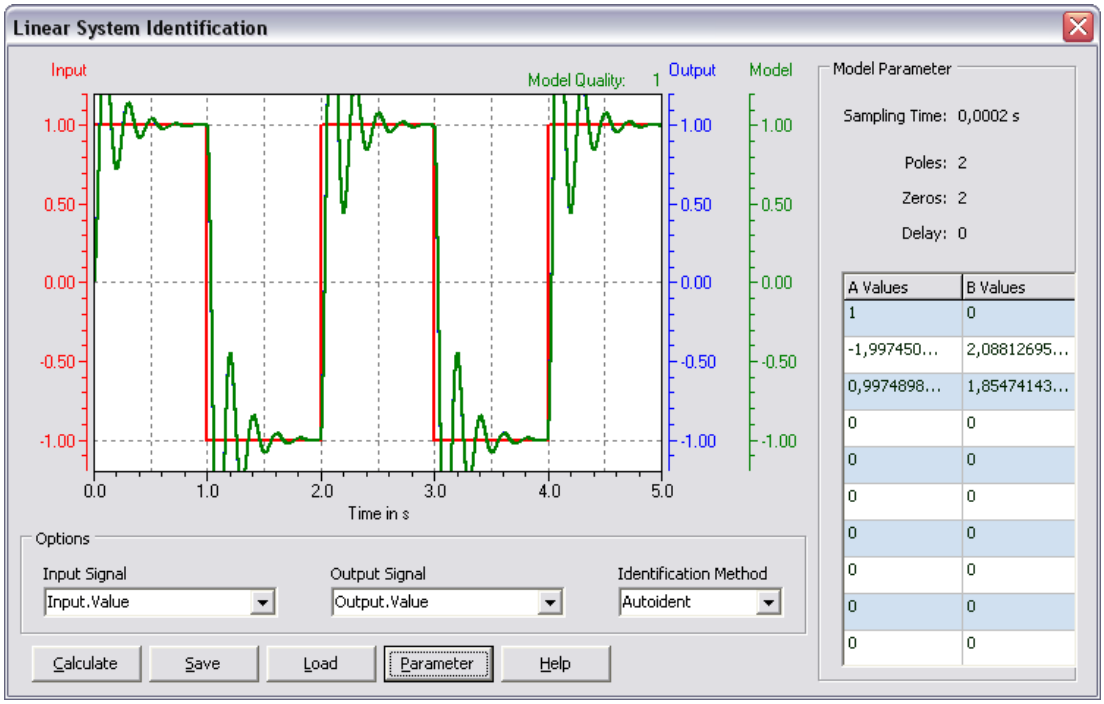
If the data is not periodic a weighting function - Hanning, Blackman Harris - can be assigned to improve the calculation.

Besides the **FREQUENCY ANALYSIS** it is also possible to calculate the frequency spectrum of a variable directly in the **ONLINE GRAPHIC**. Further information is available in section 4.6.2 Adapt simulation graphics.

 © FLUIDON® GmbH 2008

5.6 System identification

The **LINEAR SYSTEM IDENTIFICATION** calculates the time discrete transfer function of a simple linear single input/output system using data vectors containing time samples of the system's input and output values.

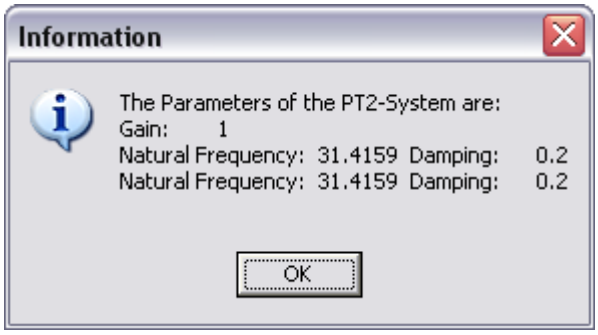


*Dialog **Linear System Identification***

Inputs and outputs may be any state or variable in the model or external data such as measurements from a real system that can be easily imported into DSH*plus* by means of the **ONLINE DATA IMPORT** component.

The default **Autoident** setting tries to identify the model with the lowest numerical order that at the same time fits best to the chosen data. The weighting criteria for this is an quadratic error function.

Subsequent to a calculation of the time discrete transfer function model the natural frequencies and the gain of the identified linear system can be calculated by means of the button **Parameters**.



Dialog ***Identification***

With **Save** the identified model is stored as a configuration file and with **Load** it is reloaded into the system identification dialog.

Using the DSH*plus* Discrete Transfer Function Component the configuration file is included into the DSH*plus* simulation model.

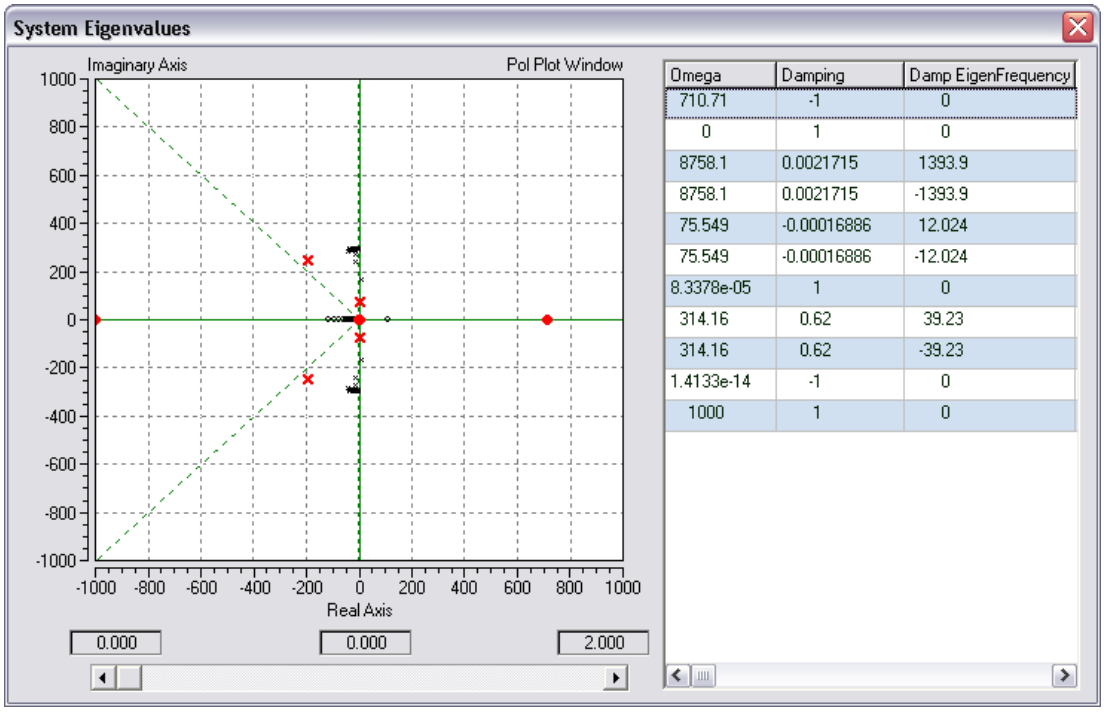
If a linear identification cannot achieve a sufficiently accurate representation of the dynamic behaviour of the system the DSH*plus* Neural Network Module using neural network techniques probably can do!

5.7 System eigenvalues

The **SYSTEM EIGENVALUES** module calculates the system eigenvalues on the basis of the results fn the last simulation run. For a maximum of 1000 stored simulation time step the system of differential equation is re-initialized and numerically linearized to achieve the Jacobian-matrix of the system.

Calculation the eigenvalues of the Jacobian-matrix for each simulation time step and printing them into a diagram displays how the eigenvalues of the system change depending on the current dynamic situation respectively simulation time. This way the analysis can quickly point out where a drift or a variation in a system state such as cylinder or valve position could adversely affect the stability, and often provides answers to where a more detailed analysis is necessary.

 © FLUIDON® GmbH 2008



Dialog ***System Eigenvalues***

The root loci diagram can be zoomed, copied to the Windows clipboard and stored as WMF-file. The calculated root loci can be saved to a TXT-file or XML-file.

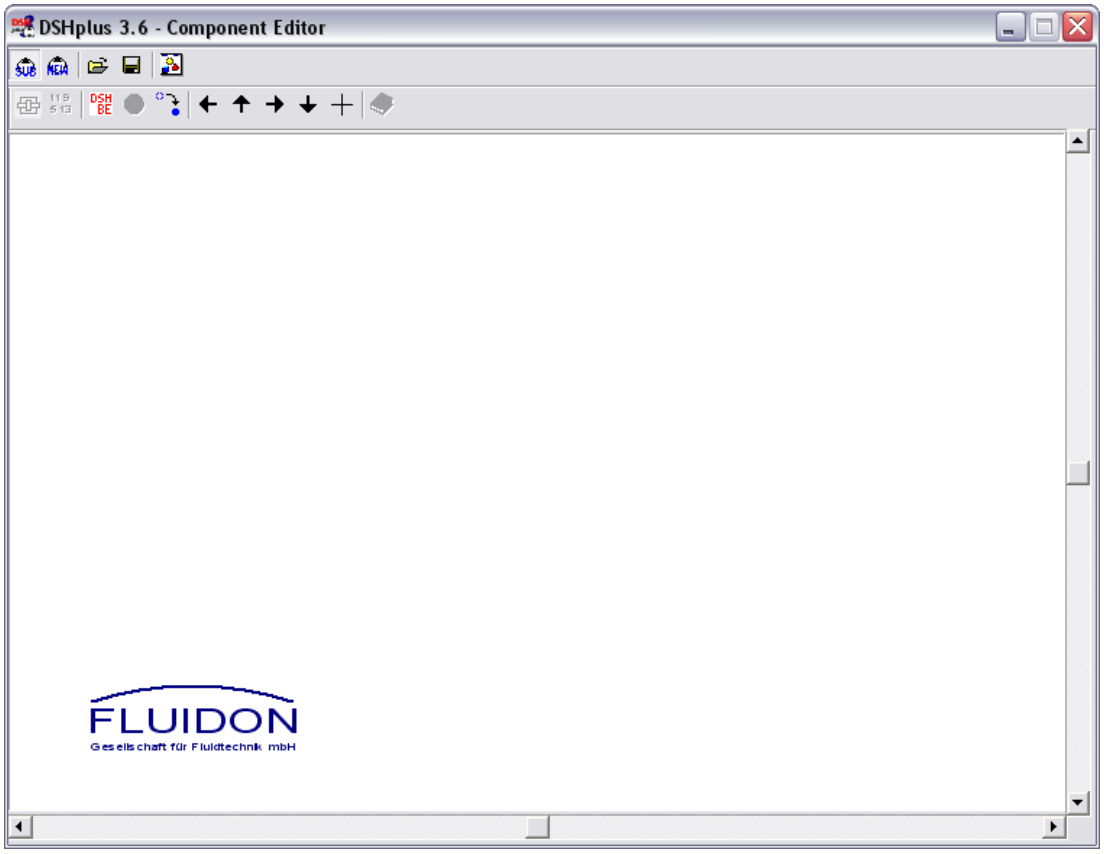
A horizontal scroll bar is available, to navigate through the calculated time steps. The Jacobian matrix of the presently selected time step, indicated by the red dots for the zeros (PT1) and red Xs for conjugated poles (PT2) can be saved into a TXT-file.

5.8 Editors

DSH*plus* comes with a set of editors, that are intended to provide support functionality to specific program modules.

5.8.1 Component Editor

The **COMPONENT EDITOR** is a graphical based development tool for new equation-based DSH*plus* components and a a tool to graphically edit the appearance of sub-module icons.



Dialog ***Component Editor***

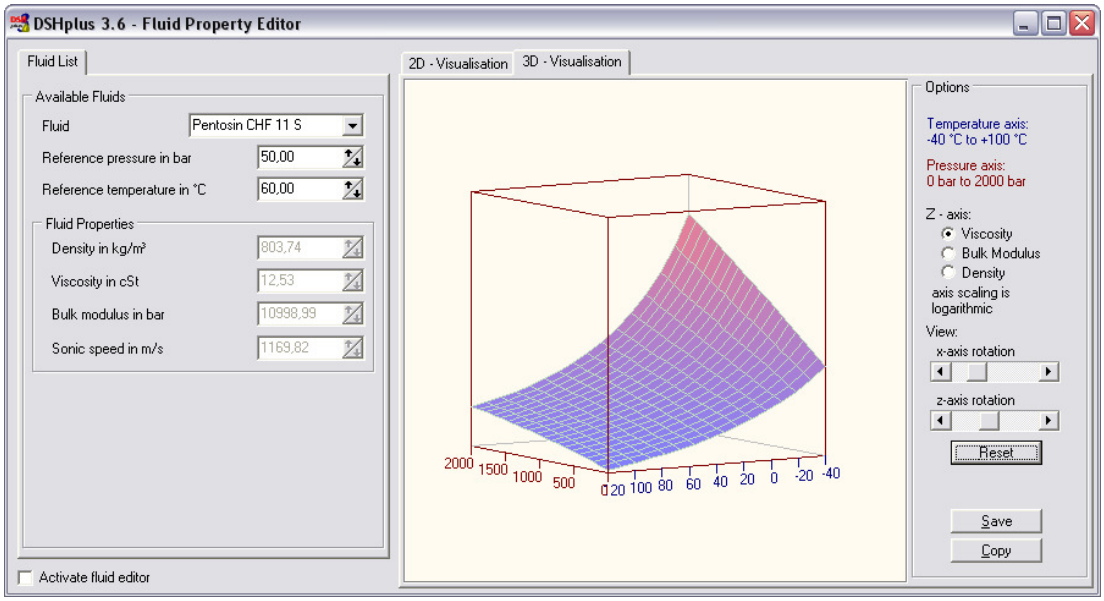
The editor is working in two different modes:

- Sub-module mode: In this mode, the grey automatically generated icon of a DSH*plus* sub-module can be exchanged against a user-specific icon. The new icon has to be a vector graphic file in WMF-file format. It is also possible to rearrange the position and the text alignment of the automatically placed component connections.
- Model mode: In this mode, it is possible to create a new DSH*plus* component, inclusive source code template and default parameter set. The template for the new icon has to be a vector graphic file in WMF-file format. The different types of component connections can freely be placed. For each input and output there will be a corresponding function included into the component source code. Only the user-specific equations have to be added. The syntax is directly checked by the DSH*plus* compiler.

 © FLUIDON® GmbH 2008

5.8.2 Fluid Property Editor

The **FLUID PROPERTY EDITOR** allows the administration of fluids available in the fluid options dialog of DSH*plus*.



Dialog ***Fluid Property Editor***

By means of manufacturer data of viscosity and density the user can add fluids available on the market. The editor compares the temperature-dependent behaviour of the fluid with the available *Witt* fluid types. The chosen fluid type and additional coefficients are stored as a new fluid data set to the DSH*plus* Fluid Data Base File (**.fmo**).

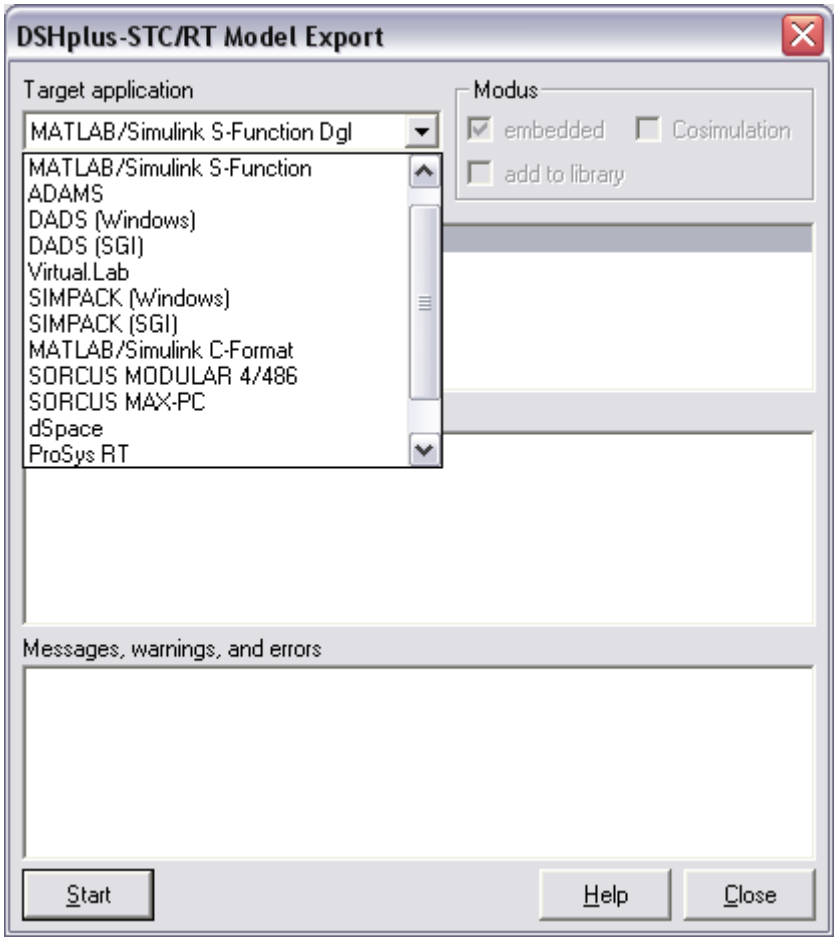
The DSH*plus* Fluid Data Base File contains the definition of all fluids available in the simulation. In the simulation model a fluid is selected by means of the **FLUID OPTIONS** dialog. In the standard simulation the selected temperature is considered as fixed tank temperature.

In order to assign an individual operating temperature to the components, a temperature distribution for the entire simulation model can be defined. Temperature relevant fluid properties will automatically be adjusted to these temperatures. In the thermo-hydraulic simulation the fluid properties are updated each simulation time step considering the pressure and temperature at the component connections.

5.9 Export model

The DSH*plus*-STC/RT model export provides a framework for co-operation between separate tools from independent software vendors.

Thus, it provides the design engineer of hydraulic and pneumatic systems with a tool that meets all requirements of a company or project wide integrated virtual engineering.



*Dialog **DSHplus-STC/RT Model export***

The DSH*plus*-STC/RT manual is providing further information, to the currently available export targets.

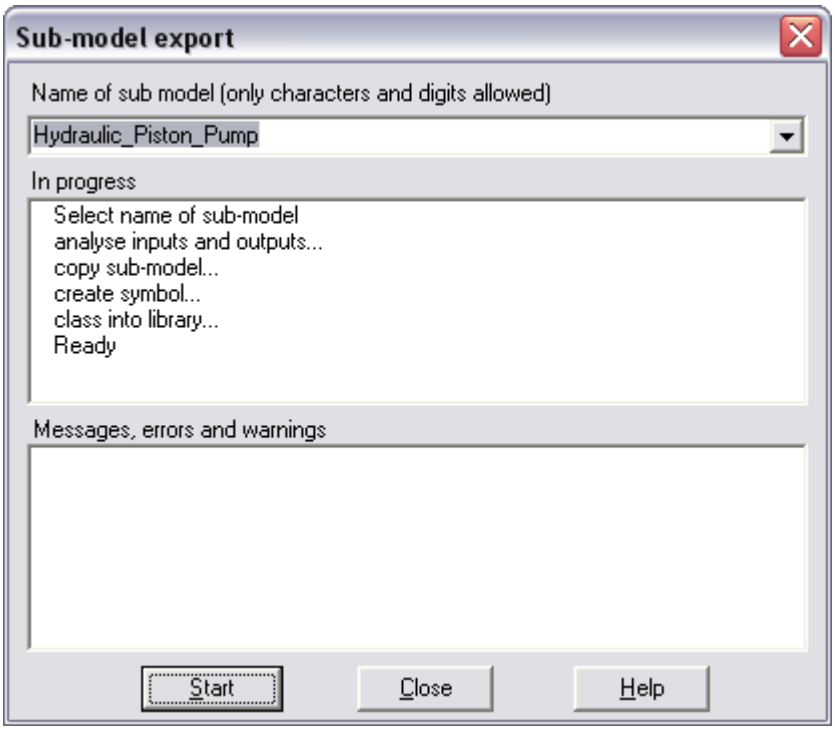
5.10 Create submodel

The Sub-Model functionality is a click on the button solution to graphically model new library elements. Without writing a single line of code,

 © FLUIDON® GmbH 2008

any **DSH***plus* model can be converted to a reusable sub-model. Validated parameter sets build a data base for the new library element.

In each case the design is basing on a combination of existing **DSH***plus* library elements, representing either a basic physical effect or already existing sub-component functionality of the new element. Special sub-model in- and output signal elements define the boundary of the sub-model.



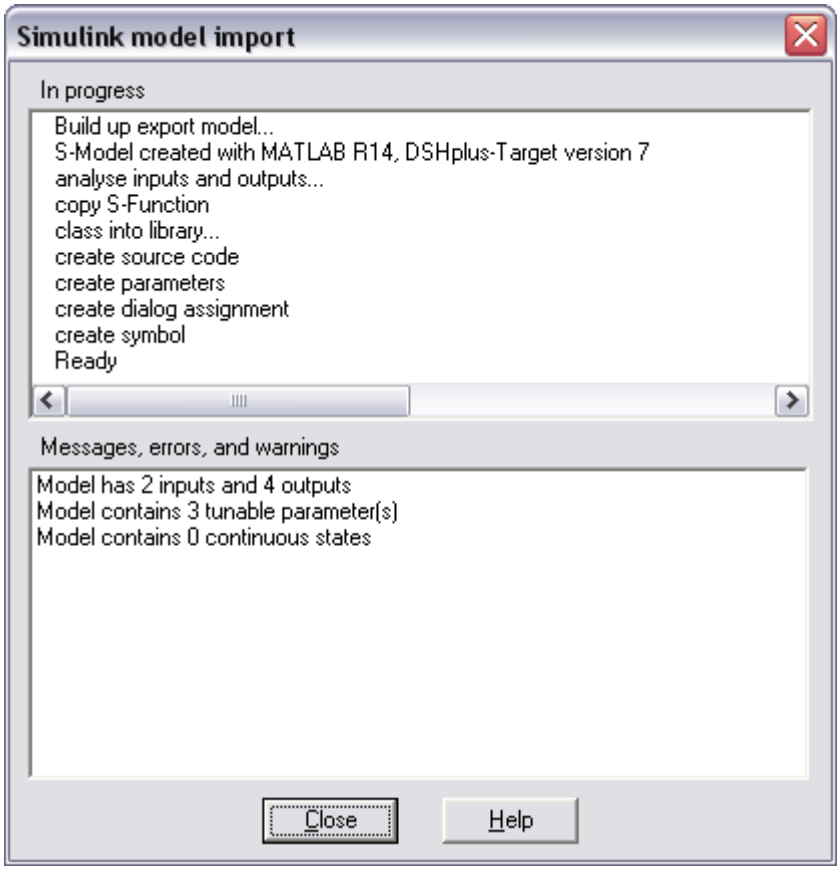
Dialog ***Sub-model export***

The Sub-Model handling in general is similar to that of standard **DSH***plus* library elements. A special dialog, also displaying the original model, handles the parameters of the sub-model. Pre-defined parameter sets can be loaded out of a database and manually modified sets can be stored as new entries.

Since Sub-Models are treated like any other library element they can be given to colleagues, project partners or customer.

5.11 Import S-model

DSH*plus* is able to import Simulink models that have been exported with the Simulink real-time workshop (RTW). To do so, Simulink RTW has to use a special FLUIODN-developed Simulink RTW target.



Dialog ***Simulink model import***

There are two possibilities to import Simulink models.

- Embedded: This way the model will be solved by the fixed step solver that was assigned in Simulink.

- Differential equation: This way the equations of the Simulink model will be included into the DSH*plus* equations and will be solved by the DSH*plus* solver.

In order to change the parameter values of a Simulink model through DSH*plus*, the "Tunable Parameter Functionality" has to be activated in Simulink prior to the export of the model.

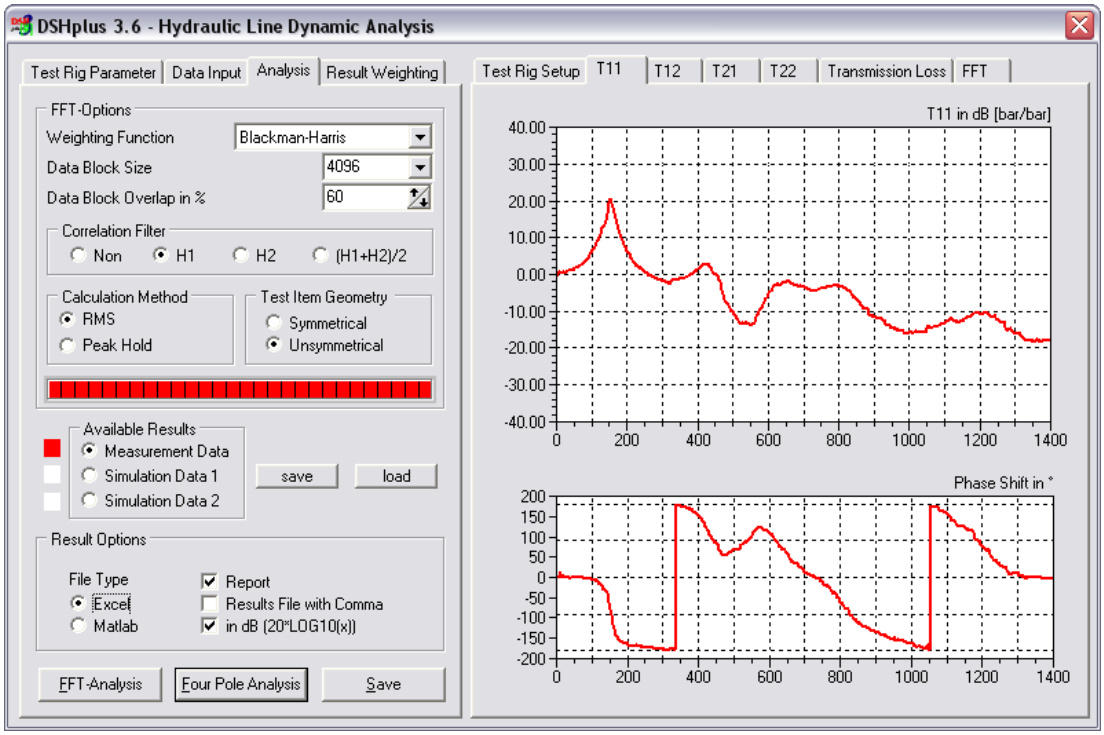
 © FLUIDON® GmbH 2008

5.12 Four pole analysis

One source of acoustic problems in hydraulic systems, the so-called fluid-borne noise, is caused by pressure ripples which are initiated by the pump. Self-propagating throughout the hydraulic pipes, fluid-borne noise leads to audible noise on other places.

FLUIDON developed a test rig for investigating and quantifying the oscillation characteristic of branched hydraulic lines in order to gain a precise calculation/prediction of vibration and resonance characteristics.

The four-pole analysis dialog is a tool that has been especially developed to process these measurement data and to compare the measured line dynamic to the simulated line dynamic.



Dialog ***Hydraulic Line Dynamic Analysis***

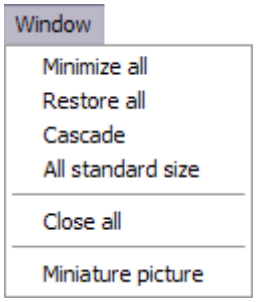
The fields of application for measurement of hydraulic oscillation characteristics is by no means limited to the classical hydraulic system. With respect to automotive pipes the investigation can also include the identification of pressure and temperature dependency of flexible hoses or the generation of line fingerprints for benchmark or quality issues.

Applicable branched pipe systems are pipes in fuel injection systems, the supply pipes of large-scale paint-spray lines or the hydraulic pipes in motor vehicles, such as steering system hoses.

In addition to the line test rig the DSH*plus* Automotive Flexible Hose Library has been developed to support the simulation-based design and optimization process of automotive pipe and hose units. Herewith automotive suppliers and manufacturers possess a new tool to improve the capability and efficiency of their NVH engineering process.

6 Window

The windows for simulation graphics and current values can be managed in common under DSH*plus*.



*Main menu **Window***

Open result windows can be minimized and restored individually by means of the standard Windows functions in the upper right corner of the dialog. Furthermore, DSH*plus* offers the functions **MINIMIZE ALL** and **RESTORE ALL** in the menu **WINDOWS**. By means of these functions all the result windows can be minimized and restored at the same time.

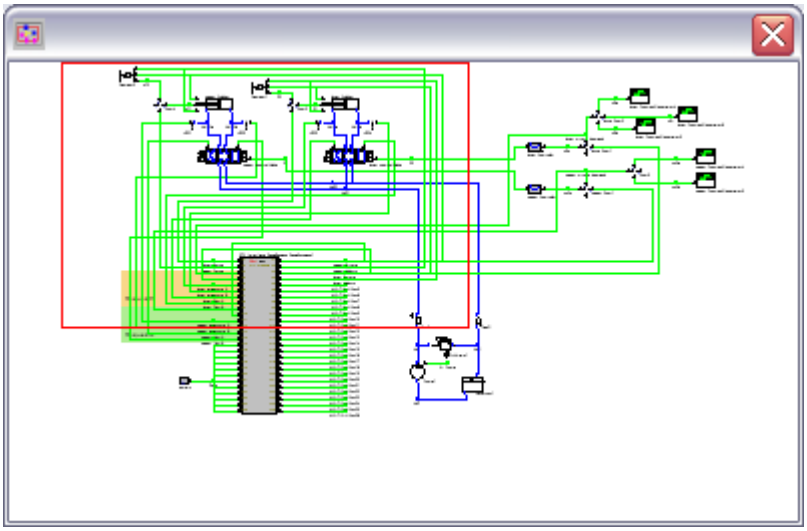
The simulation graphics windows can in addition be adjusted with the functions **CASCADE** and **ALL STANDARD SIZE** in the **WINDOWS** menu.

All the windows are closed by means of **CLOSE ALL** in the **WINDOWS menu**.

 © FLUIDON® GmbH 2008

6.1 Miniature picture

MINIATURE PICTURE opens a new window that shows the entire simulation model. Clicking into the miniature picture it is possible to pan the model view for an easy model overview and faster component location.



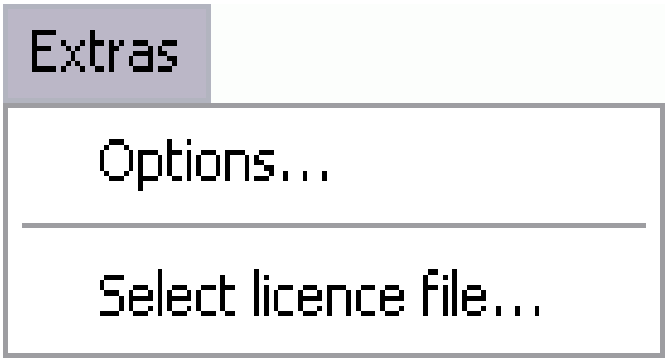
Miniature picture

The current view of the model is indicated by a the red rectangle, whose size is dependent from the zoom factor of the working area.

MINIATURE PICTURE works in the edit mode and also in the design level.

7 Extras

In the menu **EXTRAS,** some functions which are not related to each other from their contents, are allocated. Among others, all analysis- and additional modules are located here. In the module manual they are described in detail.



*Main menu **Extras***

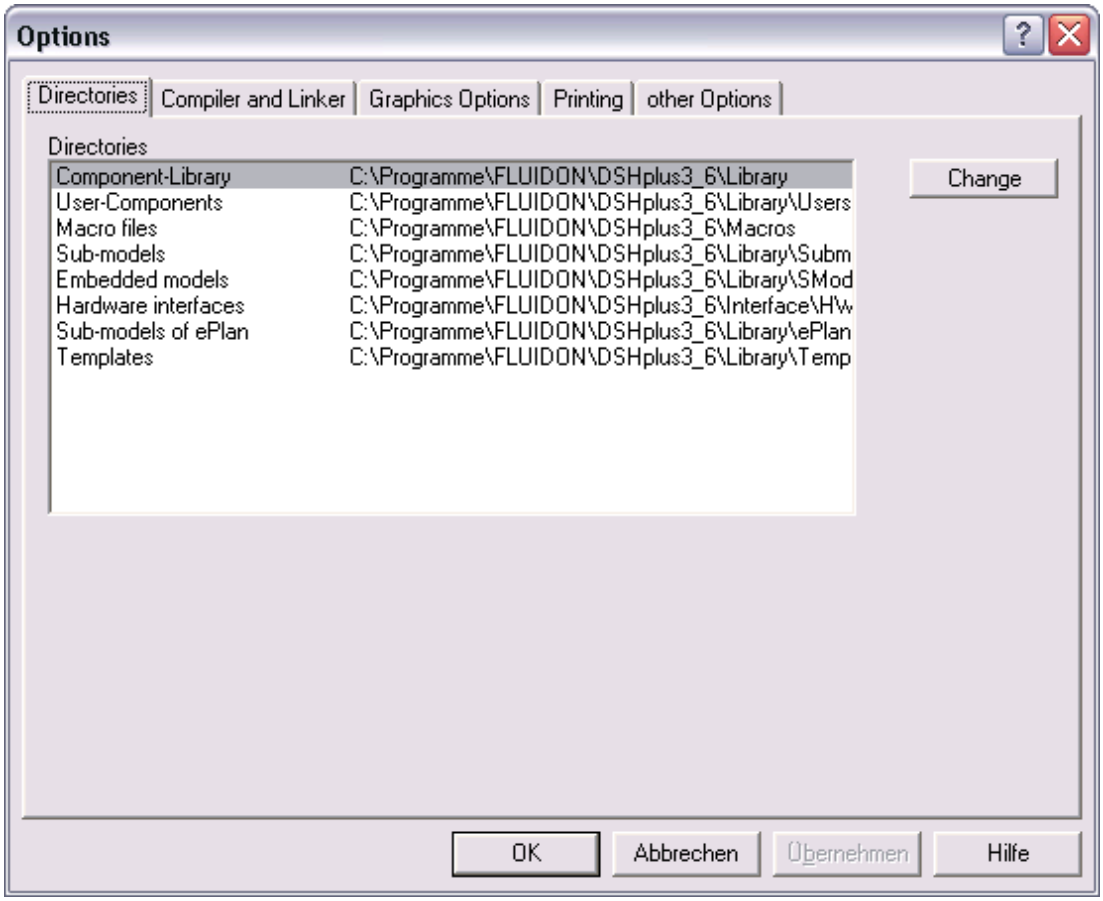
7.1 Options

The menu point **OPTIONS** is located under **EXTRAS**. This menu point opens the dialog **Options**. A number of program settings can be carried out in this dialog.

7.1.1 Directories

Important paths for managing DSH*plus* are adjusted in the category **Directories** in the dialog for **Options**:

 © FLUIDON® GmbH 2008



Settings of paths

- **Component library**: This is the path to the DSH*plus* standard component library.

- **User components**: This is the path to the user specific library and in which the archived component parameter sets are stored.

- **Macro files**: This is the path to the directory in which the macros are stored.

- **Embedded models:** This is the path to the directory in which components, that encapsulate DSH*plus*-RT models, are stored.

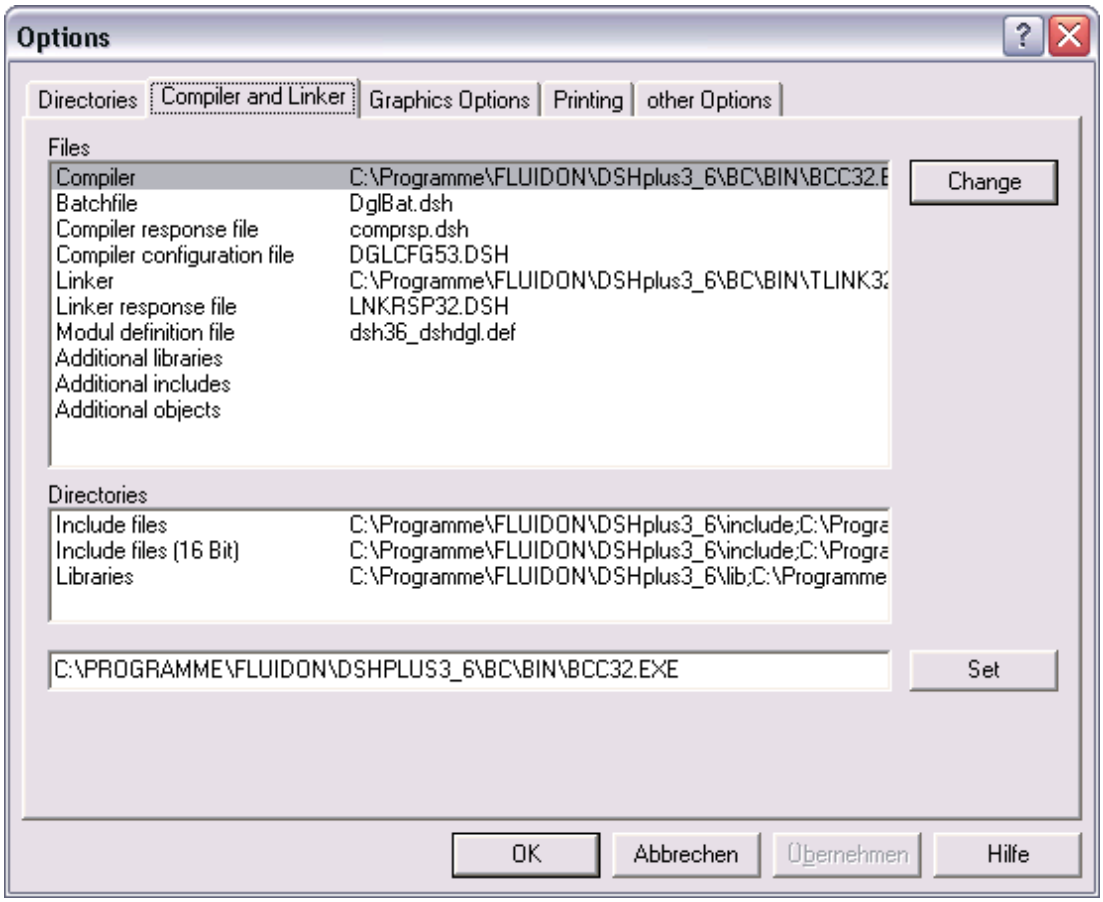
- **Hardware interfaces:** This is the path to the directory in which dialogs, that belong to special DSH*plus*-STC modules, are stored.

- **Sub-models of PLAN:** This is the path to the directory in which EPLAN sub-models are stored.

- **Templates:** This is the path to the directory in which DSH*plus* model templates are stored.

7.1.2 Compiler and Linker

In the category **Compiler and Linker** in the dialog for **Options,** the paths to the compiler and linker used by DSH*plus* are adjusted.



Setting of Compiler and Linker

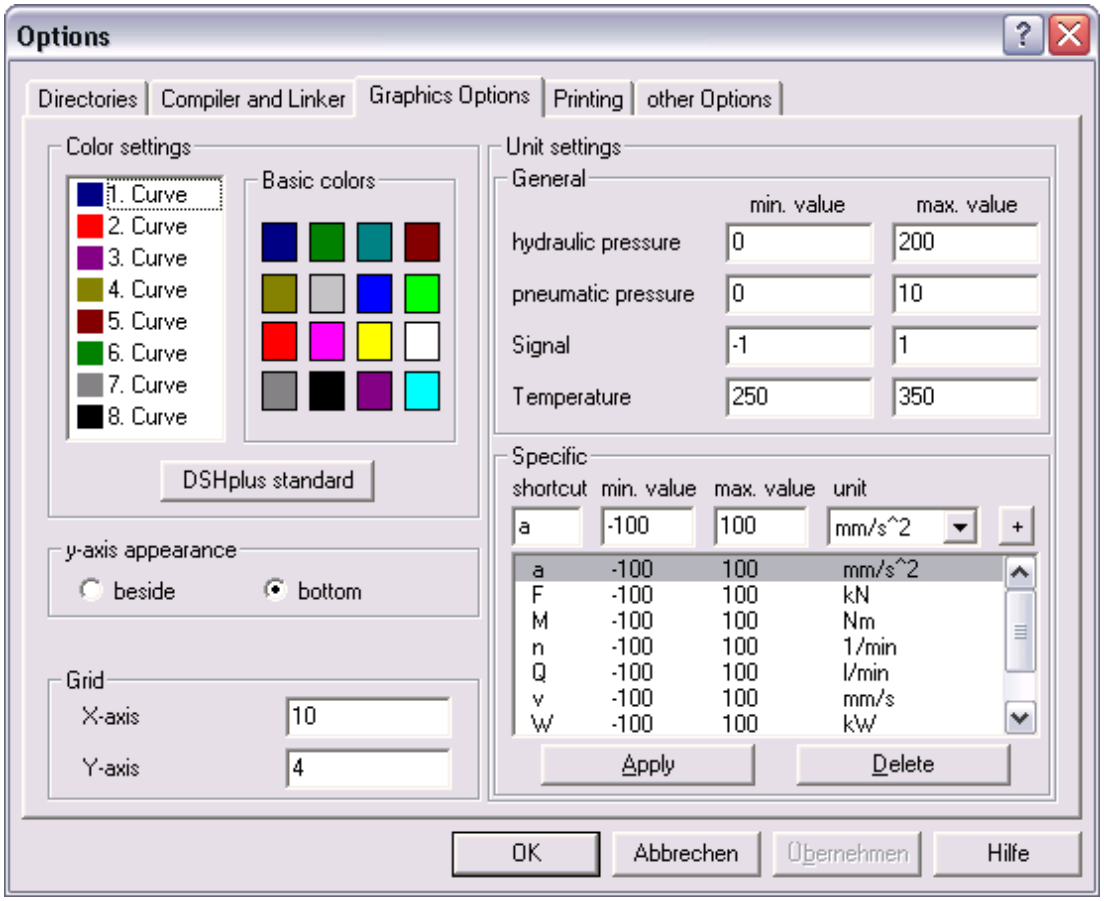
Input field	Explanation
Files	
Compiler	Compiler name with path
Batchfile	Model for the batch file without path (DglBat.dsh)

 © FLUIDON® GmbH 2008

Input field	Explanation
Compiler response file	Model for the response file of the compiler without path (comprsp.dsh)
Compiler configuration file	Model for the configuration file of the compiler without path. dglcfg53d.dsh is used to generate a model with debug information
Linker	Linker name with path
Linker response file	Model for the response file of the linker without path. linkrsp32d.dsh is used to generate a model with debug information
Module definition file	in series dsh36_dshdgl.def without path
Additional libraries	Additional user specific libraries without path
Additional includes	Additional user specific include files without path
Additional objects	Additional user specific object files without path
Directories	
Include files	Path to DSH*plus* and Compiler includes
Include files (16 Bit)	Path to Compiler 16 Bit includes
Libraries	Path to DSH*plus* and Compiler libs

7.1.3 Graphics Options

The category **Graphics Options** in the **Options** dialog determines the standard settings of the simulation graphics.



Settings of Graphics

In the area **Color settings** the default colors which are allocated to the individual variables are determined. The manual settings can be restored at any time to DSH*plus* **standard**.

In the area **Unit settings** the settings for the presentation of nodes in the simulation graphics are made. In the sub-area **General** the minimum and maximum values for hydraulic pressure, pneumatic pressure, signal, and temperature are adjusted. These values are also used for scaling of the connection line thickness during a animated simulation run.

In the sub-area **Specific**, the minimal and the maximal values as well as the units for an automatic parameterization of nodes in the simulation graphics can be defined. The initial character of the node name will be compared to the character printed in the **shortcut** line. Capital and small initial letters are distinguished.

 © FLUIDON® GmbH 2008

The settings in the area **Specific** can be adjusted to the used demand. In order to add a new shortcut a character has to be entered into the field **shortcut**. Subsequently, the desired settings have to be made in the fields for minimum and maximum values and in the drop down list for the units. By clicking on **APPLY NOW** the new shortcut with its settings is included into the list.

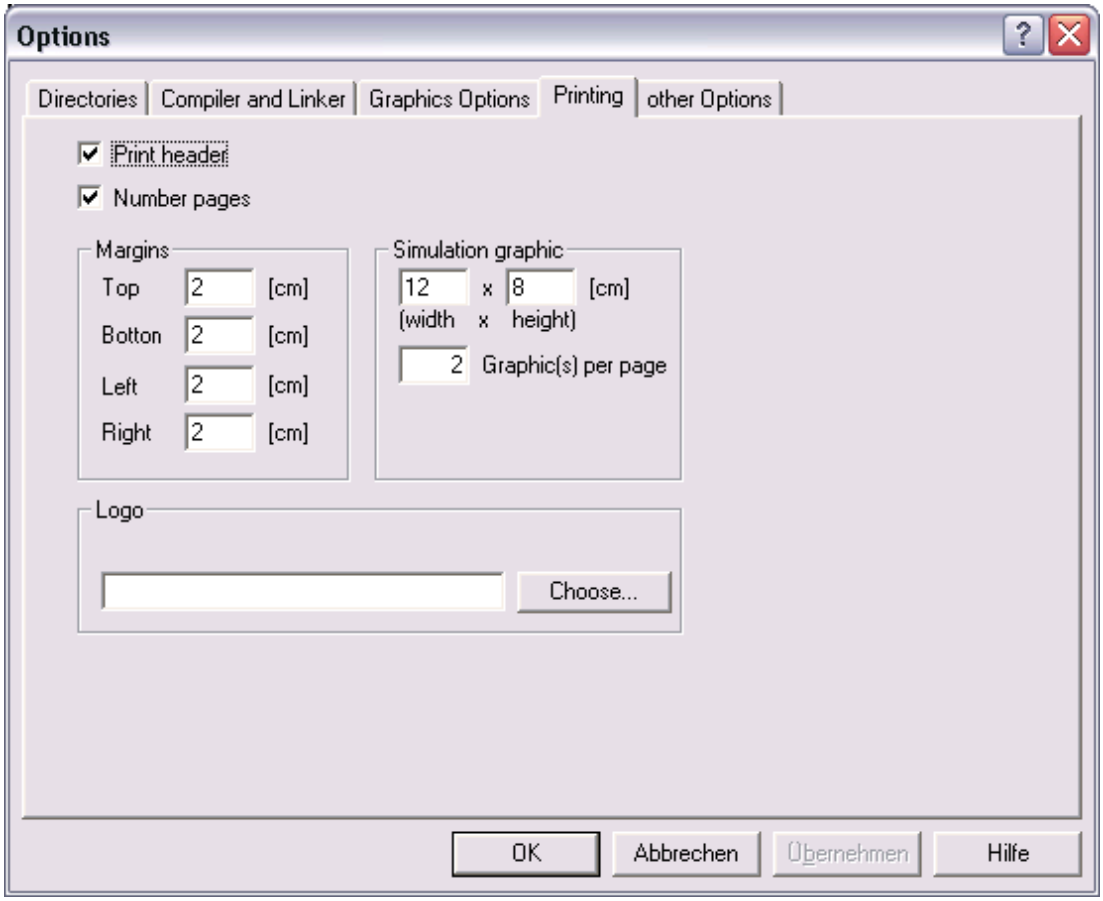
If you want to change the settings for a shortcut, please click on the shortcut. Now you are able to process its settings. With **APPLY NOW** the new settings are included into the list.

If a shortcut including its settings is to be deleted, please select is from the list and delete it by clicking on **DELETE**.

The shortcut list is stored into the CURRENT_USER section of the windows registry.

7.1.4 Printing

The settings for all printing processes under DSH*plus* are adjusted in the category **Printing** in the dialog for **Options**.



Settings of printing

The following setting possibilities are located here:

- A logo can be printed out by means of the option **Logo**. The corresponding graphics file is selected by means of selection. The logo has to be available in the Windows-Metafile-Format (WMF) or Window-Bitmap-Format (BMP). The logo is also used in the DSH*plus*-HTML report.

- the option **Print header** determines whether the project, the author, the remark to the project and the parameter set comment is printed,

- **Number pages** determines whether the pages are numbered,

- the side margins are fixed under **Margins** and

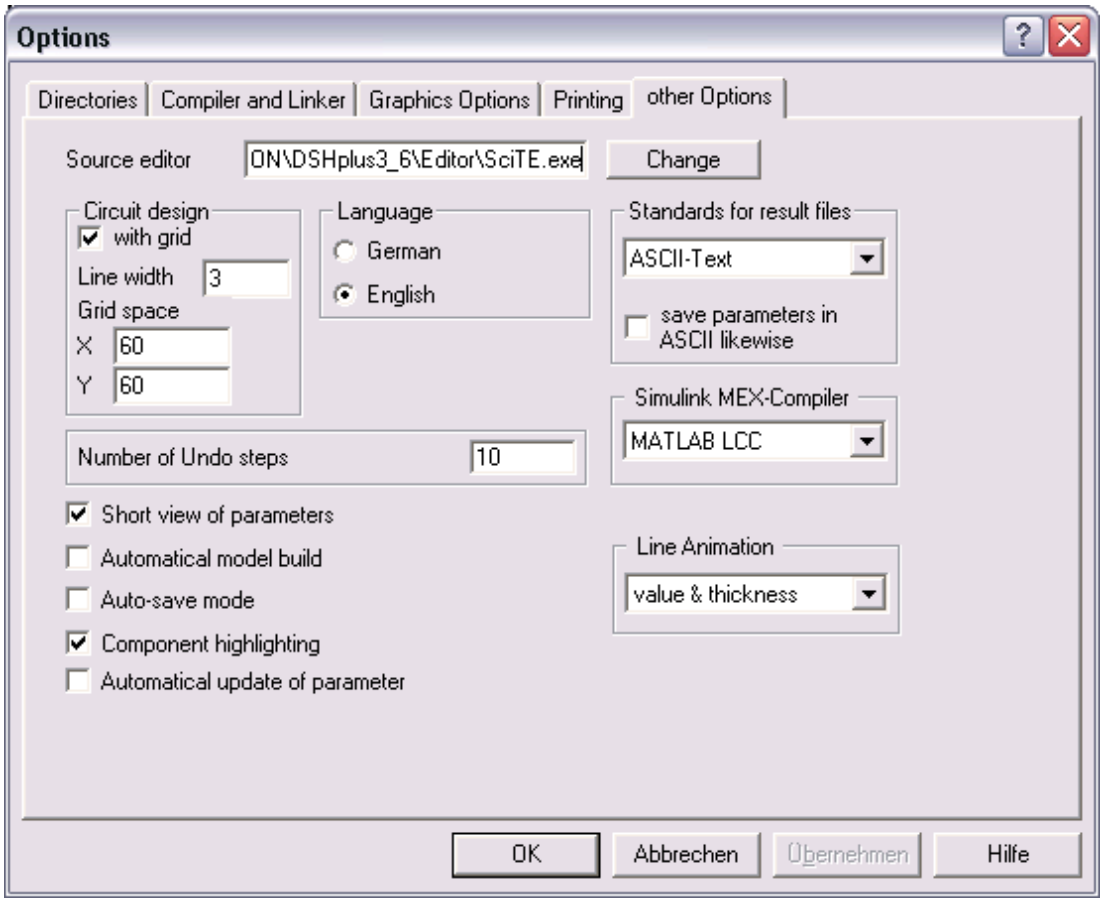
- under **Simulation graphic** is fixed in which size a simulation graphics is printed and how many simulation graphics are to be printed

 © FLUIDON® GmbH 2008

on one page. If the graphics indicated does not fit into one page, the remaining graphics are printed on the next page.

7.1.5 Other options

Under the category **other Options** auxiliary settings are combined in the dialog for **Options.**



Other options settings

The following setting possibilities are located here:

- with **Source editor**, the editor with which the source code of the current simulation model can be opened, is selected. The source code is opened by means of **EDIT SOURCE** in the menu **MODEL**,

- under **Circuit design** is adjusted whether the grid is active or not when the program is in design mode and which grating space the grid has (Standard value 60:60),

- under **Language** the language, English or German, in which DSH*plus* is operated and in which the language in which the simulation models are generated is adjusted,

- with **Standards for result files** is adjusted with which program format the created results are saved,

- with **Number of undo steps** is adjusted how many of the last steps are maintained in the store during the graphic processing of the model,

- under **Simulink MEX-**Compiler it is adjusted with compiler MATLAB is using to compile the DSH*plus*-STC S-Function,

- with **Line Animation** it is adjusted which type of line animation is preferred. Available are: Thickness, Value & name, Value, name & thickness, Value, value & thickness.

- **Short view of parameters** determines whether a yellow tool tip appears if the mouse is positioned over a component or a component connection. Parameter names and values or connection names are displayed.

- **Automatical model build** determines whether the simulation model is generated automatically or manually after the design mode is closed.

- **Auto-save mode** determines whether the entire project is archived automatically every three minutes.

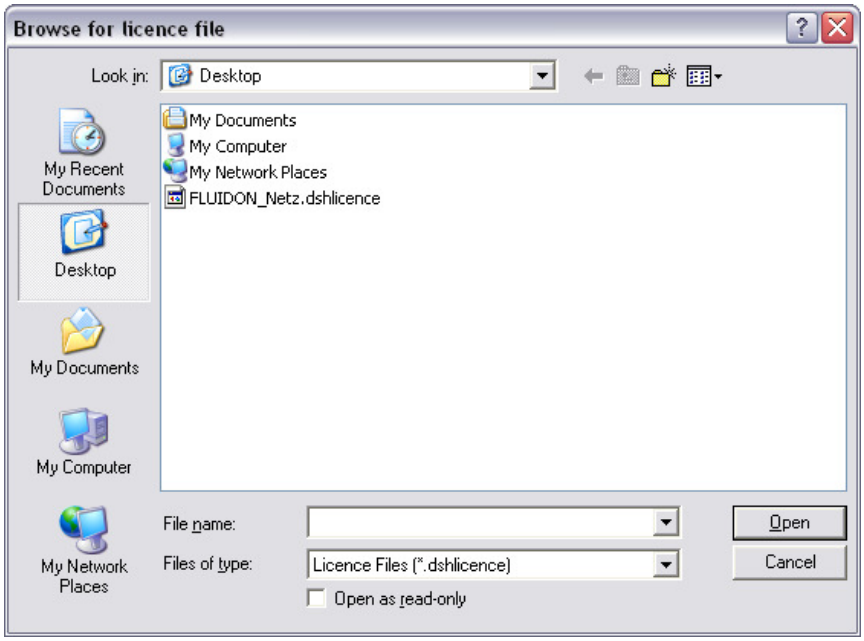
- **Component highlighting** determines whether a component is marked by a red frame if the corresponding variable name is selected in the online graphic setting.

- **Automatical update of parameter** determines whether an external parameter file is read before the simulation start. The parameter settings in such a file overwrite internal parameters. A parameter file template is achieved through the ASCI-export of parameter sets.

 © FLUIDON® GmbH 2008

7.2 Select licence file

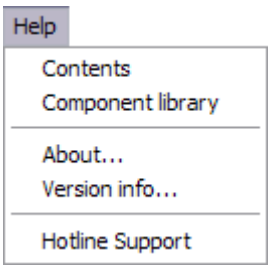
Select licence file opens an open file dialog to search, select and therefore change the licence file.



*Dialog **Browse for licence file***

8 Help

The help documents to for program and library is called by means of the menu **HELP**.



Furthermore, help to a specific dialog can be also obtained by means of the help keys in the individual dialogs.

In the design mode, the data sheet of a component can be called by a right mouse button click onto the component.

8.1 Contents

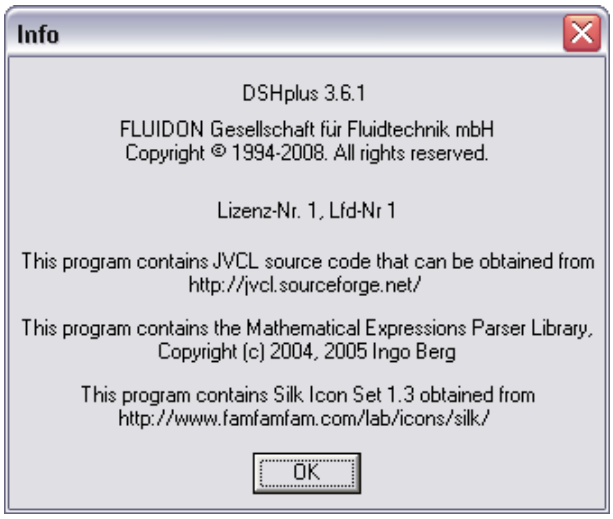
Outline of the operation and all the functions of DSH*plus*.

8.2 Component library

Outline of all the components of the component library.

8.3 About

The **INFO** dialog displays the information on the current DSH*plus* version and the license number.

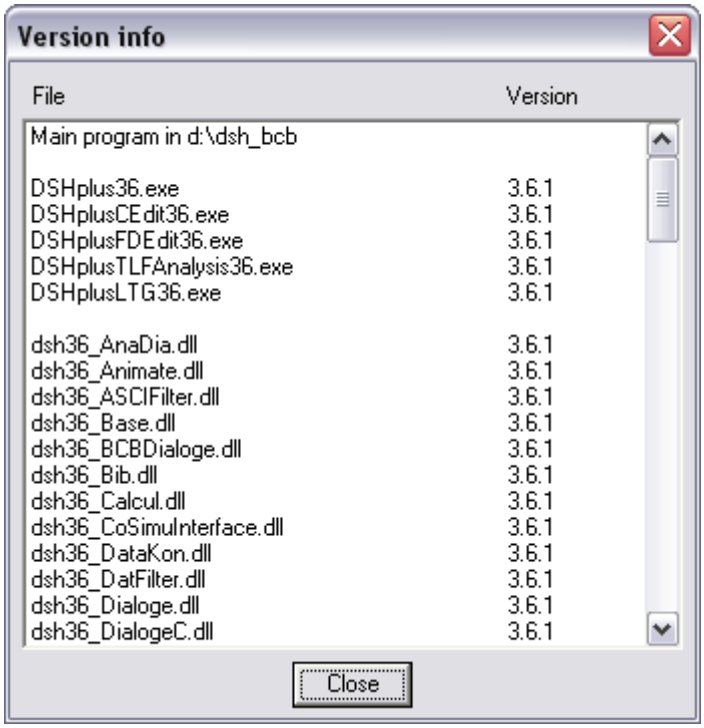


*Dialog **Info***

Moreover the dialog gives credits to open source packages used within DSH*plus*.

8.4 Version info

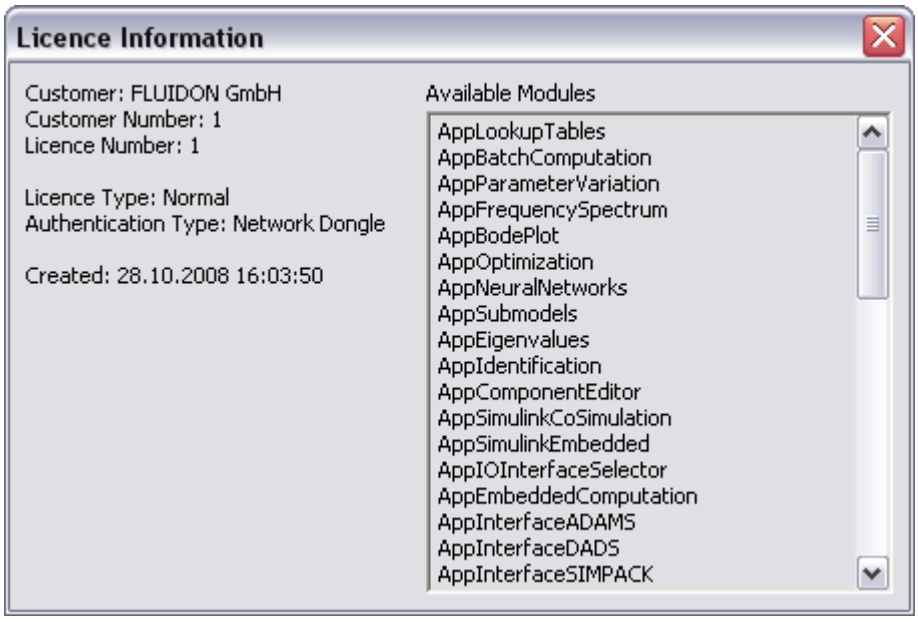
The dialog **VERSION INFO** displays the release numbers of the DSH*plus* modules and libraries.



*Dialog **Version info***

8.5 Licence info

The dialog **LICENCE INFO** displays which DSH*plus* modules and libraries are available in the respective licence.



*Dialog **Licence Info***

8.6 Hotline Support

A click onto the Hotline item of the menu **Help** invokes the users standard email program. The used email address belongs to the FLUIDON customer support. The FLUIIDN support system automatically generates a support ticket notification after receiving the email.

If a DSH*plus* model is open, the model and a compiler result text file is enclosed to the mail.

Please make sure, to send no company confidential information!

9 Context menus

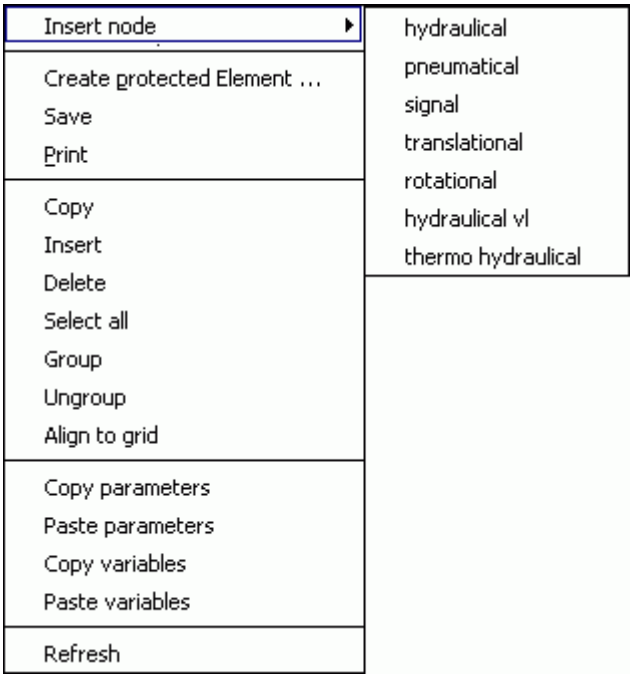
The main menu is complemented by context menus containing functions which are especially available for a particular case of application or respectively a window and which relate exclusively to this application. Context menus are opened by clicking on the object or the area that is to be edited with the right mouse key.

In DSH*plus,* three different context menus are at your disposition:

- main context menu,
- graphics context menu,
- and help context menu.

The main context menu is opened by clicking on the drawing area or on a component on the drawing area by the right mouse-key. It is available both in the design mode and in the simulation mode.

 © FLUIDON® GmbH 2008



Context menu

The individual menu points have the following functions:

- **INSERT NODE** opens an additional menu from which different types of nodes can be selected. A selected node is placed on the drawing area irrespective of a component.

- **CREATE PROTECTED ELEMENT** converts the source code and the component parameters of a selected component into a black box component.

- **SAVE** saves the changes at the model in the design mode.

- **PRINT** prints the current model design, that is visible on the drawing area.

- **COPY** copies marked elements into the clipboard.

- **INSERT** inserts the contents of the clipboard into the model.

- **DELETE** deletes marked objects.

- **SELECT ALL** selects all objects on the drawing area.

- **GROUP** combines all marked components to a group.
- **UNGROUP** cancels the grouping of components.
- **ALIGN TO GRID** aligns all components at the matrix of the grid.
- **COPY PARAMETERS** copies the parameters of the clicked component into the DSH*plus* internal clipboard.
- **PASTE PARAMETERS** transfers the parameters existing in the DSH*plus* internal clipboard onto the clicked component.
- **COPY VARIABLES** copies the variables of the clicked component into the DSH*plus* internal clipboard.
- **PASTE VARIABLES** transfers the variables existing in the DSH*plus* internal clipboard onto the clicked component.
- **REFRESH** removes graphic faults from the drawing area.

The help context menu opens by clicking on a component or a group of components in the component library with the right mouse-key. Via this context menu you are able to call the component related help page.

The graphics context menu is explained is section 4.6.2 Adapt simulation graphics.

 © FLUIDON® GmbH 2008

5 Model structure

The simulation model which is created for each simulation task includes the mathematical description of the technical system which is to be examined, as source code in C++. It is composed of the description of the individual components which form the model as well as the system of differential equations.

In order to show the functioning of the system calculation of DSH*plus* and the structure of the components and models arising from it more clearly, the principles on which the calculation of the systems is based are explained in the following.

First of all, the mathematical principles are described and the implementation into DSH*plus* is explained. In the following sections, the important files for the user are mentioned. The project and library files, among which especially the parameter and unit files, are described and the structures of the source code of a component as well as of a complete model are explained.

This is to give the user an outline of the program structure. It enables him/her to program his/her own components.

1 Mathematical principles

The behaviour of simple technical systems can in general be well-assessed and estimated. With increasing complexity, however, it is becoming more and more difficult to make reliable statements on the behaviour of a system. One needs to create models and to assess the system behaviour by means of simulations. Should the simulations be carried out on a digital computer, it is a first necessity for the model formation to describe the components of the systems by mathematic equations.

The following chapters explain the way in which technical systems are mathematically described and how transfer of the mathematics is effected in DSH*plus*.

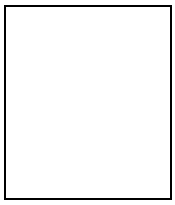
1.1 Mathematical description of technical systems

The dynamic behaviour of a technical system can be described by a differential equation of the nth order or by n differential equations of the first order. Description by n differential equations of the first order is, however, more suitable for simulations on a digital computer as it corresponds to the programming requirements for numeric calculation.

The parameters of the system components and the starting values of the variables lead to an initial value problem with n ordinary differential equations of the first order and the relating initial conditions:



with



and

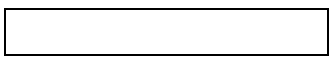


.

By solving this system of differential equations and integration of the results within the simulation time, the variables of the technical system can be calculated for the next time step.

In order to guarantee numeric solubility, the technical systems in DSH*plus* are only to be described with differential equations of the first order. This means that differential equations of a higher order have first to be transferred into a system of differential equations of the first order, if necessary.

A PT2 element for example cannot be described direct with its transfer function



but has to be first transferred into an equivalent system of differential equations with differential equations of the first order:

 © FLUIDON® GmbH 2008

$$x_1 = y$$

$$x_2 = \dot{x}_1$$

$$\dot{x}_2 = k\omega^2 u - 2D\omega x_2 - \omega^2 x_1$$

This equation system forms the basis for implementation into DSH*plus*.

1.2 Calculation method of DSH*plus*

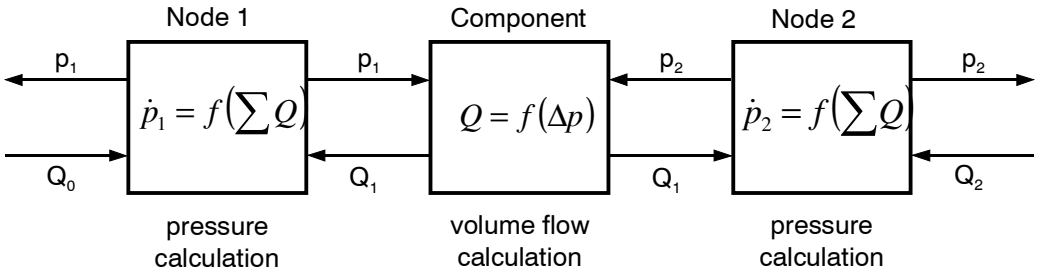
Calculation of the dynamic behaviour of technical systems is effected by forming and solving nonlinear differential equations for the different variables of the system. In the following, an explanation is made of how the differential equations for the variables are created in DSH*plus*.

1.2.1 Calculation of hydraulic variables

In DSH*plus,* the hydraulic components of a system are modelled by means of the components stored in the library. In the first modelling step, the hydraulic pipes between the components are shown as hydraulic nodes. (Modelling of hydraulic pipes by means of specific piping components is explained at a later stage.)

This model structure results from the basic node-oriented calculation method of DSH*plus*: **Pressures** are calculated in the nodes from the balance of the entering and leaving volume flows and transferred to the components. **Volume flows,** however, are calculated in the components from the difference of the pressures applied to the connections.

The following illustration shows the basic calculation method of DSH*plus*:



Calculation method of DSHplus

The starting variable of the nodes is the pressure which is calculated by integration of the pressure build-up equation:

$$\dot{p} = \frac{1}{C_h} \cdot \sum Q = \frac{E_{\ddot{O}l}(p)}{\sum V} \cdot \sum Q$$

The input variables of a node are the entering and leaving volume flows of the components. The volume flows are calculated in the components from flow equations or other mathematical relations.

The node-oriented calculation method of DSH*plus* also leads to the plus/minus sign definition of the volume flows: The volume flows flowing to a node are positive and the volume flows leaving a node are negative. This means that – contrary to the common definition – the volume flows flowing to the components are negative and those leaving the components are positive.

In DSH*plus,* the nodes are treated as ideal capacity and resistance and inertia of masses are not taken into account or added to the components connected. The oil volume included in a node is composed of the volume of the hydraulic pipe and the idle volumes of the components connected. If elements with variable volumes (e.g. cylinders) are connected to a node, the variable volume is automatically added to the volume of the node.

If in a second modelling step, the hydraulic pipes of the system are shown by means of pipe components, resistance and inertia of masses of the pipes can also be taken into account. The basic calculation method of DSH*plus,* however, is not changed as the pipes are also connected to the components by means of the nodes.

 © FLUIDON® GmbH 2008

1.2.2 Calculation of the pneumatic variables

The most important pneumatic values are pressure p and temperature T. A further variable which reflects the relation between mass flow and volume flow is density ρ. If air is considered as ideal gas, however, it is calculated from the pressure and the temperature as per the ideal gas equation.

$$\rho = \frac{p}{T \cdot R}.$$

1.2.2.1 Pressure change

In order to describe the pressure change of a pneumatic system, the 1st axiom of thermodynamics for open systems as well as the ideal gas equation are used. Pressure change is therefore additively composed of three parts: pressure change because of mass flows, heat transfer and volume change.

$$\dot{p} = \dot{p}_{\dot{m}} + \dot{p}_{W\ddot{u}} - \dot{p}_{\dot{V}}.$$

The variable $\dot{p}_{\dot{m}}$ describes the energy exchange which is effected by the mass flows at the components connected.

$$\dot{p}_{\dot{m}} = \frac{\kappa}{V} \cdot R \cdot \left[\underbrace{\sum \left(\dot{m}_{zu} \cdot T_{zu} \right)}_{>0} + \underbrace{\sum \left(\dot{m}_{ab} \cdot T_{ab} \right)}_{<0} \right].$$

Furthermore, the energy exchange with the environment in the form of heat flow has to be taken into account.

$$\dot{p}_{W\ddot{u}} = \frac{\kappa - 1}{V} \cdot \left[\alpha \cdot \left(T_W - T \right) \cdot A \right].$$

This is characterized by the heat transmission coefficient α and the temperature difference between the housing and the airflow within the balanced volume.

The third term describes the work of the volume change at the balanced system

$$\dot{p}_{\dot{V}} = -\frac{\kappa}{V} \cdot p \cdot \dot{V} .$$

1.2.2.2 Temperature change

Calculation of the temperature change is effected by resolution of the ideal gas equation as per the temperature *T*

$$T = \frac{p \cdot V}{m \cdot R}$$

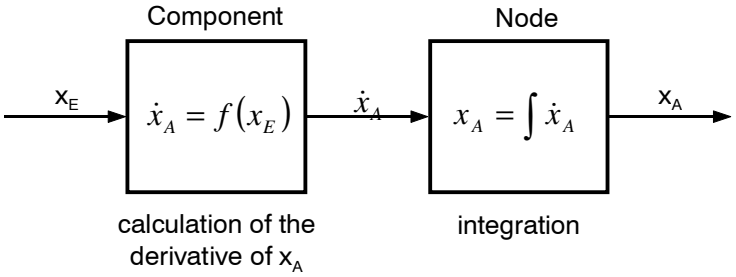
and the following partial differentiation as per all time-dependent variables.

$$\dot{T} = \frac{\dot{p}}{p} \cdot T + \frac{\dot{V}}{V} \cdot T - \dot{m} \cdot \frac{R \cdot T^2}{p \cdot V} .$$

In order to minimize the time needed for calculation, the pressure change $\dot{p}$ is calculated first in the simulation and then the result is used for determining the temperature change $\dot{T}$.

1.2.3 Calculation of the mechanic variables

Calculation of the mechanic variables which are calculated by means of integration, is effected in the same way as the calculation of hydraulic variables: The derivations of the variables which are integrated into the node, are calculated in the components. The illustration shows the basic calculation method of DSH*plus* for mechanic variables.



*Calculation method of DSH*plus *for mechanic variables*

In the components, the equations for accelerations ($\ddot{x}$) as per the Newton axiom are formed at a mass from the equilibrium of forces:

 © FLUIDON® GmbH 2008

$$\ddot{x} = \frac{\sum F}{m}$$

The same is true for angular accelerations ($\dot{\omega}$) with rotary masses which are calculated from the rule of angular momentum:

$$\dot{\omega} = \frac{\sum M}{I}$$

Accelerations or angular accelerations are integrated at the node from which result the speeds (v) and stokes (x) as well as the angular speeds (ω) and the angles (φ):

$$v = \int \ddot{x} dt \quad \text{and} \quad x = \int v dt$$

or

$$\omega = \int \dot{\omega} dt \quad \text{and} \quad \varphi = \int \omega dt \,.$$

1.3 Numeric integration

The system of differential equations shown in chapter 1.1 describes the derivatives of all the variables in every operating point. In same instances, it can be explicitly solved. For calculation of the initial value problem on a digital computer, however, a numeric solution is required.

To do this, the integration interval $[t_0,t_n]$ is first reduced into a grid

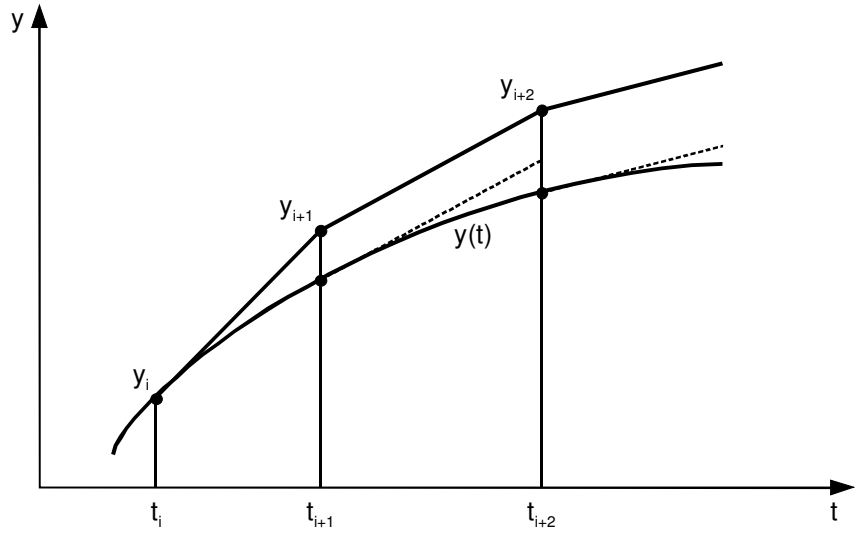
$$t_0 < t_1 < t_2 < \ldots < t_n$$

with local step size $h_i = t_{i+1} - t_i$ for $i = 0 .. n-1$.

Approximate values $\underline{y}(t_i)$ at the discrete grid points are obtained from the relation

$$\underline{y}(t_{i+1}) = \underline{y}(t_i) + \int_{t_i}^{t_{i+1}} f(t, \underline{y}(t)) dx \quad \text{with } i = 0 .. n-1,$$

which can be solved by numeric methods.



Numerical integration

An error (discretisation error) results from the approximate numerical integration which occurs with every integration step. In order to minimize this error, discretisation of the integration interval should be as fine as possible. Discretisation has its limits, however, as on the one hand any fine subdivision of the integration interval leads to a large number of calculation steps and thus to a highly increasing calculation time, on the other hand the number of numeric rounding errors increases.

In DSH*plus,* both one-step methods and multiple step methods are used for solving system of differential equations. “One-step methods” means that the approximate values $\underline{y}(t_i)$ at the time t_i can be completely calculated from the previous result $\underline{y}(t_{i-1})$ at the time t_{i-1} and the interval $h_i = t_{i+1} - t_i$. Multiple step methods, however, also use much older results, t_{i-2}, t_{i-31}, ….

Explicit as well as implicit one-step methods are implemented. Explicit means that the new value is calculated exclusively from old values. The implicit solution, however, includes the new value into the calculation, thus iteration is necessary.

If the number of discretisation errors which occur do not increase when the number of increments increase, an integration procedure is numeri-

 © FLUIDON® GmbH 2008

cally stable. The stability field of the explicit methods is larger than that of the implicit methods. This means that implicit methods can lead to inexact results when the simulation step size is too large without getting instable at any case while explicit methods rather tend to instability.

If the inherent values of a system of differential equations are extremely different, mathematical stability is high. system of differential equations with which hydraulic systems are described are in general very stable because the individual systems considerably differ in their dynamic behavior. This means that the occurring volumes, masses or moments of inertia are extremely different. The mathematical stability can also extremely change in the course of the simulation. In such cases, it is useful to carry out the calculations with the variable step size *h*.

An automatic step size control is provided in DSH*plus* in order to achieve optimum results with systems with a large number of nonlinearity's and high mathematical stability. With every calculation step, it is checked whether the preset error tolerance is met and the step size is adapted accordingly. An step size control does not necessarily mean a reduction of the total calculation time.

The following variable step size integration methods are available in DSH*plus*:

- ODE23 (rk2(3) RK) parameter ID.-No. = 200
- ODE45 (rk5(4)6m PD 45) parameter ID.-No. = 201
- Dormand-Prince 45 parameter ID.-No. = 202
- Euler implicit parameter ID.-No. = 203
- Rosenbrock implicit parameter ID.-No. = 204
- Radau implicit parameter ID.-No. = 205

In the procedures with fixed step size the step size is maintained constant throughout the simulation. The following methods are at your disposition:

- ODE1 (Euler-Cauchy) parameter ID.-No. = 100
- ODE2 (Heun) parameter ID.-No. = 101
- ODE3 (RK I) parameter ID.-No. = 102
- ODE4 (Runge-Kutta) parameter ID.-No. = 103
- ODE5 (Prince-Dormand) parameter ID.-No. = 104
- ODE4 (Runge-Kutta RCH) parameter ID.-No. = 105
- ODE5 (Prince-Dormand RCH) parameter ID.-No. = 106

Subsequent are some general information about the setup of some of the integration algorithms.

- *Euler-Cauchy*

  Euler-Cauchy calculates the integration as per Euler's point slope method of the first order. The explicit method calculates with a fixed simulation step size. The equation system is only calculated once for every integration step. This simple method is suited to blind out numeric effects in the solution which are possibly caused by the integration procedure. Apart from that, the method is used rather seldom.

- *Heun*

  With Heun the differential equation system is called twice. This method is more precise than the Euler-Cauchy method.

- *Runge-Kutta I*

  The Runge-Kutta-I-procedure is designed according to the principle of the Runge-Kutta method and is of the third order. The differential equation system is called three times per step.

 © FLUIDON® GmbH 2008

- *Runge-Kutta*

  With Runge-Kutta the integration is calculated as per the classic Runge-Kutta method of the fourth order with a fixed simulation step size. The explicit method for every integration step is called four times all together. Because of the higher quote of error the step size can be set 10 times higher than with the Euler method given the same error.

- *Prince-Dormand*

  Prince-Dormand is also a method of the type Runge-Kutta. Contrary to the classic Runge-Kutta-method, however, it is of fifth order and thus reaches a higher quote of error. Whereas the accuracy increases the speed decreases due to the greater number of evaluations. Nevertheless, this method is a good compromise between accuracy and speed.

- *Iteration*

  The iteration serves for the static solution of the differential equation system (determination of initial value). The system is solved in every point in such a way that it is in balance. The variables which are dependent only on another variable (e.g. strokes) are calculated from one time to another with an Euler step and determine the marginal conditions for a new solution.

- *Runge-Kutta 2/3*

  Runge-Kutta 2/3 is a method of 2./3. order. The system is solved respectively with a method of second and third order. A measure for the step size adjustment is obtained from the tolerance of the two results. Because the solution is calculated with so-called embedding formula it is not necessary to calculate the equation system five times altogether but only four times. One evaluation point is used by both methods.

- *Runge-Kutta 4/5*

  The variable Runge-Kutta is the step size controlled version of the 4th and 5th order. Like Runge-Kutta 2/3 this method uses embedding formula in order to reduce the total of nine points to six evaluations.

- *Variable Prince-Dormand*

  The variable Prince-Dormand is the step size controlled version of the Prince-Dormand-method of 4th and 5th order. Like Runge-Kutta 2/3 this method uses embedding formula in order to reduce the total of nine points to six evaluations.

- *Variable implicit Euler, Rosenbrock, and Radau*

  This method is based on the implicit Runge-Kutta. Apart from the step size also the order of the method is adjusted.

More information about integration methods is offered in various numeric books.

2 Description of the most important files

The files of the projects and the library are described in the following two sections. In general, all project files are automatically created by DSH*plus*, but an outline of the files in which the project data are saved is used for better understanding the system. Thorough knowledge of the library files is needed if user-defined components are to be programmed.

2.1 Files of the projects

All the files in which project data are saved are managed internally by DSH*plus*. The files are archived into one single file with the extension ".dsz". At each save operation the previous ".dsz"-file is converted into a ".bak"-file

 © FLUIDON® GmbH 2008

Internally the DSH*plus* project consist of several files that are briefly explained in the following section.

The names of the files are in general *dshpp__1.*, pp* designating the project number and „*“ a file specific extension (*dsh, cpp, dll, grf, sgr*, *str* or *ken*). (If the project number is only one-digit, the first digit is replaced by an underline: *dsh_p__1.**)

Files in which parameter or result sets are saved, are designated by *dshppnnn.**, *n* designating the number of the parameter set or the number of the result and „*“ designating a file specific extension (*par* or *erg*). (If the number of the parameter set or the number of the result are only one-digit or double-digit, the first digits are replaced by underlines: *dsh_p_nn.**)

In general, the projects are only processed via editors so that it is not necessary for the user to have any thorough knowledge of the structure of the file management and the structure of the files.

Only in specific applications such as a new programming of the components for example, the user needs further knowledge on data management of DSH*plus*. Thus, the contents of the individual files is described in the following sections.

- projects.inf

  The file *projects.inf* is a text file containing information about the current directory. This information is displayed in the project dialogs and can be edited there.

- projects.dsh

  The file *projects.dsh* is the management file for all the projects which have been stored in a directory. It contains a list of all the projects with their project number. Furthermore, the title, additional remarks to the project, the author, the date and the time of the creation of the project and the date and the time of the last change are saved in the file *projects.dsh*.

DSH*plus* creates a *projects.dsh* file for every directory in which the projects are stored.

- dshpp__1.dsh

  The file *dshpp__1.dsh* is a management file for all the files of a project. It contains a list of all the files which belong to a project.

  Furthermore, the connections between the parameter and result files are saved in the file *dshpp__1.dsh* (also see chapter 4, 4.6 Save result).

  DSH*plus* creates a *dshpp__1.dsh* file for every project.

- dshpp__1.cpp

  The file *dshpp__1.cpp* contains the source code of the entire model. It is created during model formation from the source code of the components and the connections of the circuit diagram. A direct intervention into the source code of a model can be useful under certain circumstances, a more detailed description of its structure can be seen in chapter 6.

- dshpp__1.dll

  The file *dshpp__1.dll* is a dynamic link library of a project. It is created from the *cpp* file. The user is not able to make changes in this file.

- dshpp__1.grf

  The file *dshpp__1.grf* contains the graphic description of the model, i.e. in this file the arrangement of the components and their connections are saved. It is automatically created from the circuit diagram of the model when the model input is terminated.

- dshpp__1.sgr

 © FLUIDON® GmbH 2008

In file *dshpp__1.sgr,* all the information on the configuration of the on-line graphics and the result graphics are saved. It is always updated when a project is terminated so that the configurations are again available the next time the project is opened.

- dshpp__1.str

  All the information on the connections of the components are saved in the structure file *dshpp__1.str*. It is created from the graphical connections of the components after the model has been put in.

- dshppnnn.par

  The files *dshppnnn.par* contain the parameter sets which have been saved for the model with the project number *pp*, *nnn* representing the number of the parameter set. The number of parameter files in one project correspond to the number of result sets and parameter sets saved.

- dshppnnn.erg

  The files *dshppnnn.erg* contain the simulation results, *nnn* representing the number of the result set. The number of result files in one project correspond to the number of results saved.

- dshpp__1.ken

  If look-up tables are used in one project, they are saved in the file *dshpp__1.ken*. The look-up tables file is always created, even if no look-up tables are used.

2.2 Files of the library

The component models in DSH*plus* are composed of three parts

- Mathematical description,
- Standard parameter set and
- Component symbol

and are allocated to different assemblies and subassemblies. These files are stored in the library files. The following table contains a list of the library files with their contents.

File	Contents
dsh36_StdLib.daz	Mathematical descriptions of the DSH*plus* specific components (cannot edited)
dsh36_StdLib.par	Standard parameter sets of DSH*plus* specific components. ASCI-text file that should not be edited by the user.
dsh36_StdLib.ein	List of the standard units. ASCI-text file that should not be edited by the user.
dsh36_StdLib.vg	Symbols of the DSH*plus* specific components (cannot be edited)
dsh36_StdLib.grb	Sub-division of the DSH*plus* specific components in assemblies and subassemblies (cannot be edited)
dsh36_StdLib.html	Template file for HTML-based reports. Can be edited but will be overwritten by a library update.
dsh36_StdLib.fmo	Data sets of the fluids
dsh36_StdLib.dia	A file, that includes links to special parameter dialogs.
AnyName.dat	Mathematical descriptions of the user-defined components
AnyName.par	Standard parameter sets of the user-defined components
AnyName.ein	List of the user-defined units
AnyName.vg	Symbols of the user-defined components (cannot be edited)
AnyName.gru	Allocation of the user-defined components to assemblies and subassemblies
AnyName.dia	A user file, that includes links to special user specific parameter dialogs.

 © FLUIDON® GmbH 2008

File	Contents
User.ken	Central look-up tables library

The structure of the important files for the user is described in the following chapters.

3 Structure of the source code of a component

The mathematical description of the components forms the basis of modelling. The equations describing the dynamic behaviour of the components are stored as source code. In the following chapters, the structural design of the component descriptions is explained.

Every component description is organized as per the same formal structure which is used with every element. It contains the following points:

- Definition of category
- Definition of parameters
- Definition of variables
- Definition of auxiliary parameters
- Definition of auxiliary functions
- Definition of exit functions
- Definition of functions
- End designation of component description

The following sections give a more detailed description of the individual points.

Note:

- Both the compiler and DSH*plus* take capital and small initial letters into consideration. Thus it is absolutely necessary to adhere to the given notation (this also true for capital letters in a word.

3.1 Definition of category

Every component is created as category. The name of the category corresponds to the name of the component and is placed at the beginning of every component description behind the keyword *class*. A comment which can be used for information on the component can follow. The definition of the category ends with *public:*.

Example:

```
class Servo22HydraulicCharacteristics {
/*-----------------------------------------------------------*/
/*   2/2 directional control valve  (Stroke>Strokemin P->A)*/
/*   Field of characteristics for the aperture              */
/*   cross-section of the metering edge hydraulic spool     */
/*   control                                                */
/*-----------------------------------------------------------*/
public:
```

Note:

- The key word class, the class name, and the brace ***{*** have to stand in one line.

3.2 Definition of parameters

The definition of the category is followed by the definition of the parameters with their file types. After the keyword *// Parameters*, all the parameters characterizing the component are listed.

Please see that the parameters are only defined here. Assignment of the parameters with numerical values is only effected when initializing the model.

Example:

```
// Parameter
  LDOUBLE Strokemax;
  LDOUBLE Strokemin;
  LDOUBLE PistonMass;
  LDOUBLE PistonDiameter;
```

 © FLUIDON® GmbH 2008

```
LDOUBLE BreakawayFrictionForce;
LDOUBLE MixedFrictionForce;
LDOUBLE SpeedMixedFriction;
LDOUBLE ViscousDamping;
LDOUBLE SpringStiffness;
LDOUBLE SpringPretensioning;
LDOUBLE AlphaDPA;
LDOUBLE Epsilon;
LDOUBLE Rho;
LDOUBLE OverlapStrokePA;
int     CharacteristicEdgePA;
```

Explanations to the example:

In general, only *LDOUBLE, int* and *PTCharacteristic* are used as variable types in DSH*plus*. Component parameters are defined as *LDOUBLE*, integral numbers (serial numbers, numbering, plus/minus signs, etc.) as *int* and look-up tables as *PTCharacteristic*. *LDOUBLE* and *PTCharacteristic* are simplified spellings which have been determined in DSH*plus*.

Note:

- The line comment // of C++ is used as marking of keywords in the DSH*plus* models. In order to avoid any errors when creating a model, it is not to be used for program comments. They have to be marked line by line with */* ... */* .

3.3 Definition of auxiliary parameters

Additional values which are only to be used internal to the categories, are defined under *// Auxiliary Parameters*. Auxiliary parameters can be allocated with the results of constant terms which result from the connection of component parameters and which are often used in the differential equations. Calculation of the auxiliary parameters is made in the auxiliary function which is called once at the beginning of a simulation run after assignment of the component parameters. This procedure increases clarity of the models and minimizes the calculation time.

Calculation of the auxiliary parameters is not to contain any variable! If an auxiliary value is needed the calculation of which also contains vari-

ables, another method has to be chosen which is described in chapter 6, 2.1.

Example:

```
// AuxiliaryParameter
  char*      ElementName;
  char*      ParameterName;
  PTCharacteristic CharPA;
  LDOUBLE    KPA;
  LDOUBLE    Q;
```

```
  Int        Calculated;
  LDOUBLE    Area;
  LDOUBLE    StrokeOffset;
  LDOUBLE    Acceleration;
  LDOUBLE    FOutside;
  LDOUBLE    FlowForceCoefficient;
```

Explanations to the example:

With an hydraulically controlled valve, the cross-sectional area of the valve spool is needed for calculation of the spool acceleration. As it is not directly given as parameter, the auxiliary parameter *Area* is defined here which is determined from the existing values in the auxiliary function.

Note:

- It is imperative that the auxiliary parameters *char* ElementName* and *char* ParameterName* are included as the two first parameters.

3.4 Definition of variables

Under *// Variables*, definition of the input values and variables as well as the hydraulic, pneumatic and mechanic connections of the component are made.

Chapter 6, 1.3 gives an exact description of the variables which have to be listed in the variable list.

 © FLUIDON® GmbH 2008

Example:

```
// Variablen
  LDOUBLE *pP;
  LDOUBLE *pA;
  LDOUBLE *pControl1;
  LDOUBLE *pControl2;
  LDOUBLE *Stroke;
  LDOUBLE *Velocity;
```

Note to the example:

- Variables are in general defined as pointer variables.

3.5 Definition of auxiliary function

The auxiliary function is called at the beginning of a simulation run. Under *// Auxiliary Function,* the auxiliary parameters which have to be calculated only once from the component parameters when starting the simulation, are determined. If dynamically assigned memory (e.g. for look-up tables) is needed for the component, the procedure to allocate it is called here.

Calculation of the auxiliary parameters must not contain any variable! If the calculation of an auxiliary value contains also variables, another method has to be chosen which is described in chapter 1.3.

Example:

```
// AuxiliaryFunction
  void AuxiliaryFunction (SIMSTRUCT* S)
  {
    if (Rho)
       KPA = AlphaDPA * SQRT (2 / Rho);
    else
    {
       KPA = 0;
       (*MessageBox) (Error, ElementName, "KPA <= 0");
    }

    Calculated = 0;

    CharPA = new TCharacteristic (CharacteristicEdgePA,
                                  S->ModellPfad);
  }
```

Notes to the example:

Under *// AuxiliaryFunction,* a plausibility test of critical parameters is carried out. If for example the wrong input of a parameter causes a division through zero, this is retained here. The user is shown a message in a message box.

3.6 Definition of exit function

The exit function is called at the end of a simulation run. Under *// ExitFunction*, dynamically assigned memory is released here. If dynamic variables have been created in the auxiliary function as it is the case with establishing an auxiliary category for look-up tables for example, they have to be released after the simulation run has been terminated. For this purpose, the function *Exitfunction* is provided.

Examples:

```
// ExitFunction
  void ExitFunction(SIMSTRUCT* S)
  {
     free (InputOld);
  }
```

```
// ExitFunction
  void ExitFunction(SIMSTRUCT* S)
  {
     delete (CharPA);
  }
```

```
// ExitFunction
  void ExitFunction(SIMSTRUCT* S)
  {
    FreeRing2 (WritePtr);
  }
```

 © FLUIDON® GmbH 2008

Notes to the examples:

Dynamic memory which has been previously allocated with the functions *malloc*, *new*, *CreateRing1* or *CreateRing2,* is again released with the functions *free*, *delete*, *FreeRing1* or *FreeRing2*. The memory is to be released again in order to avoid any unnecessary memory loss.

3.7 Definition of functions

The most important element of the mathematical component description is to be found under *// Functions*. The calculation specifications for the model are defined here.

The functions start with a line comment which contains the designation of the connection (e.g. *// pP*) or the variable (e.g. *// Stroke*). The designation of the connection of the component has to correspond for its part to the designation of the symbol as otherwise the program is not able to allocate a function to this connection.

Then, the function is defined with *LDOUBLE Function name.* Description of the secondary and main calculations the result of which is returned to the system of differential equations with *return,* is placed in braces.

In principle, all the function values which are returned to the DGL system with *return*, are integrated. The output value is thus integrated from the time derivation which is calculated in the function. An exception are functions in the line comment of which the keyword *algebra* is used (e.g. *// Volume flow algebra*). They are not integrated but used as algebraic values.

During simulation, the functions are not called one after the other so that their sequence is not important for the function definition section. The result of this is, however, that with some components operational parts have to be programmed for every variable.

In total, difference is made between four different functional types and different additional functions:

- Functions allocated to hydraulic pressure connections

  First of all, the functions allocated to a pressure connection have to be mentioned. They supply the volume flow entering or leaving the connection. For pressure variables, not the entire differential equation is formed but only the volume flow leaving the component as the differential equation is allocated to the connected node. The nodes sum up the volume flows of the connections linked to each other and use them for calculating the pressure build-up.

 © FLUIDON® GmbH 2008

Example:

```
// pP
  LDOUBLE VolFlowP(SIMSTRUCT* S)
  {
     Calculated = 1;
     return (- (Q = (*CharPA) (*Stroke - OverlapStrokePA)
                    * KPA * psqrt (*pP - *pA)));
  }
```

- Functions allocated to pneumatic pressure connections

Formation of the differential equations for the description of the pressure build-up and the temperature change is effected in the node as with the hydraulic values. The equation parts necessary are defined in the components (see chapter 5, 1.2.2). These are terms for dV/dt, dm/dt, Tdm/dt and dQ/dt. If individual equation parts are not applicable to the component (as they are zero for example), they can be omitted.

Example:

```
// PN1 dV/dt
  LDOUBLE PN1VolumeChange(SIMSTRUCT* S)
  {
      if (ShockAbsorber1Aktiv(S))
         return ( *velocity * DampingArea1);
      else
         return ( *velocity * PistonArea1);
  }

// PN1 dm/dt
  LDOUBLE MassFlow1(SIMSTRUCT* S)
  {
     ...
     return (AuxiliaryMassFlowDamping1);
  }

// PN1 Tdm/dt
  LDOUBLE MassFlow1_multiplied_temperature(SIMSTRUCT* S)
  {
      ...
      return (MassFlow1(S) * TMassFlowDamping1);
  }
```

```
// PN1 dQ/dt
  LDOUBLE HeatFlow1(SIMSTRUCT* S)
  {
     ...
     return (HeatFlow (Alpha, WallArea,
                       WallTemperature, *T1));
  }
```

- Functions for calculation of component internal variables

  Differential equations for component internal values represent a second functional type. The variable allocated to this function has to be mentioned in the variable list at the beginning of the component description. It has, however, not in any case to be put out as component output.

  These functions are used for example with volumes inside the component which do not have any connection to other components or with calculations of stoke and speeds.

  Example:

```
// Velocity
  LDOUBLE VelocityDeriv (SIMSTRUCT* S)
{
  Acceleration = (  (*pControl1 - *pControl2) * Area
                  - *Velocity * ViscousDamping
                  + SpringForce
                  - FlowForceCoefficient * (aw1 + aw2))
                  / PistonMass;

    if (((*Stroke >= Strokemax) && (Acceleration >= 0.))
     || ((*Stroke <= Strokemin) && (Acceleration <= 0.)))
       return (0.);
    return (Acceleration);
  }
```

- Integration functions allocated to signal outputs

  Another type are the functions calculated by means of integration and allocated to a signal output, that means that the variable is not mentioned under //*Variable.* As integration is effected in the connected node, the functions of this type are not calculated unless this

 © FLUIDON® GmbH 2008

connection is used. If the output is not connected, the function is not calculated.

Example:

```
class Integrator {
...

// Functions
// Output
   LDOUBLE Out(SIMSTRUCT* S)
   {
     return (*Input * Ki);
   }
};
// End
```

If the variable is to be calculated in any case and is to be put out to a connection without disappearing as internally mentioned variable an additional variable is introduced. For reasons of simplicity, both have the same name but are differentiated by capital or small initial letters. The function allocated to the output is of the type algebra and its return value is the variable.

Example:

```
class Integrierer {

...

// Variablen
   LDOUBLE *exit;
...

// Funktionen
// exit
   LDOUBLE outPoint(SIMSTRUCT* S)
   {
      return (*input * Ki);
   }

// output algebra
   LDOUBLE Out(SIMSTRUCT* S)
   {

     return (*output);
   }
};
// End
```

- Algebraic functions

  A last type are the functions which are not used for calculation of a differential equation but for an algebraic function. These functions are marked by means of the keyword *algebra* behind the allocated variable name comment. They are allocated to the mechanical connections and signal outputs which are not calculated by means of integration.

  Example:

```
// Volumeflow algebra
  LDOUBLE VolFlow(SIMSTRUCT* S)
  {
     if (Calculated)
        return (Q);
     else
        return (Q = (*CharPA) (*Stroke - OverlapStrokePA)
                    * KPA * psqrt (*pP - *pA));
  }
```

- Additional functions

  Additional functions are functions which are frequently required in the component but which have to be calculated only once. They are in general directly placed behind *// Functions* (The normal standards of C++ apply). Additional functions have to be placed in front of the position in the source code at which they are called.

  Example:

```
// Functions
  LDOUBLE LaminarFriction (SIMSTRUCT* S, LDOUBLE
*VolumeFlow)
  {
     Re = FABS (ConductanceRe * *VolumeFlow);

     if (Re < 2300.)
        return (*VolumeFlow * Conductance * 64);
     if (Re < 2400.)
        return (  *VolumeFlow * Conductance
                * (-994 + 0.46 * Re));
     if (Re < 7100.)
        return (*VolumeFlow * Conductance
                * (39.53191 + 0.0293617 * Re));
```

 © FLUIDON® GmbH 2008

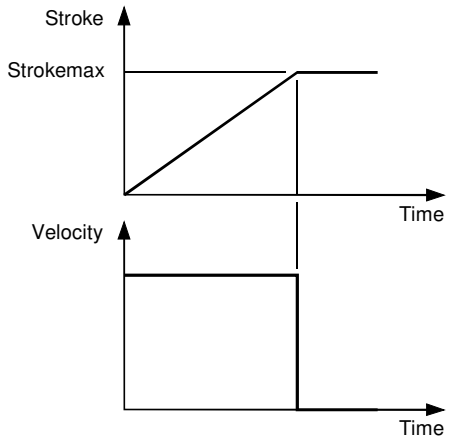
```
      if (Re < 21000.)
         return (  *VolumeFlow * Conductance
                  * (88.63309 + 0.022446 * Re));
      if (Re < 65000.)
         return (  *VolumeFlow * Conductance
                  * (206.81818 + 0.01681818 * Re));
      return (*VolumeFlow * Conductance * 0.02 * Re);
   }

// p1
   ...
```

- Limit function

  The limit function is an additional function to the differential equation of a variable with which the effects can be taken into account when the variable exceeds its limit value.

  If for example the piston of a directional valve gets to its limit stop (*Stroke* = *Strokemax*), the stroke is limited to its limit value *Strokemax*. Achieving this limit stop also has its effects on the *speed* of the piston which has to be set to zero although the limit value has not been exceeded (see illustration).



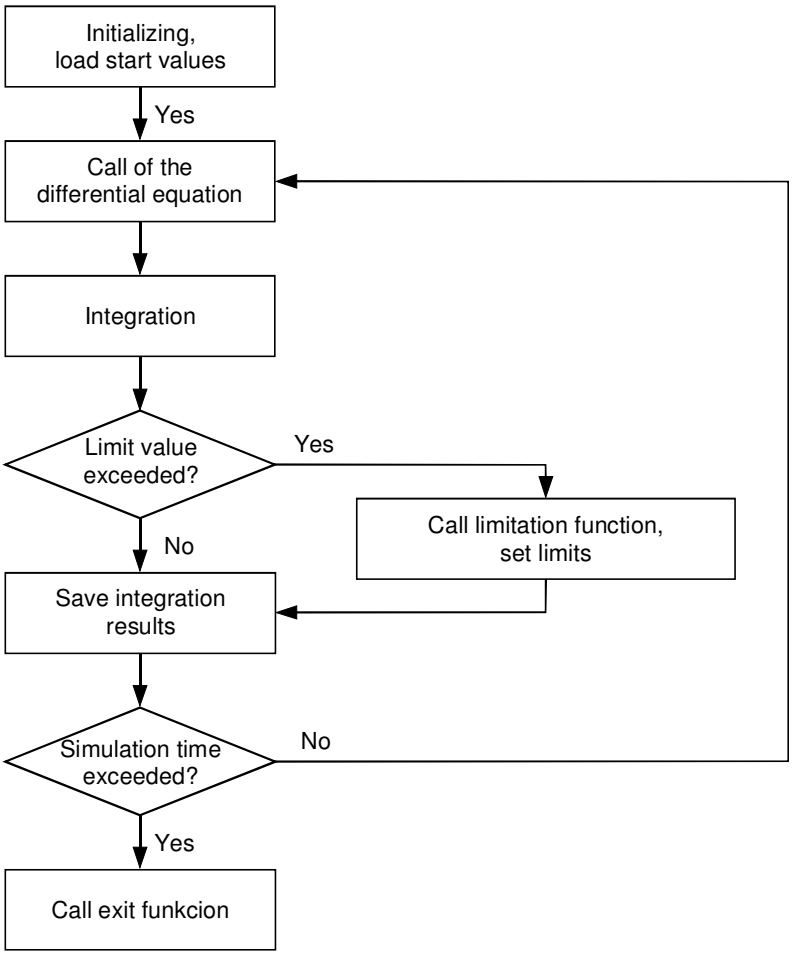
Stroke and velocity when a limit stop is achieved

  If achieving the limit value of one variable affects another variable, a second function to this variable is defined which describes its behaviour when the limit value is exceeded. A limit function is called

with the variable name to which the ending *Limit()* is added. The result of the limit function returned with *return* is not integrated.

If a variable exceeds a limit value, it is checked whether a limit function exists and can be called, if necessary.

The following illustration shows the integration of limit functions into the simulation process.



Simulation process in DSHplus

Example:

```
// velocity
  LDOUBLE velocityLimit(SIMSTRUCT* S)
  {
    if (((*stroke >= strokemax) & (*velocity >= 0.)) ||
       ((*stroke <= strokemin) & (*velocity <= 0.)))
       return (0.);
    else
       return (*velocity);
```


© FLUIDON GmbH 2008

```
    }
```

- Volume correction function

  The volume correction function is an additional function to hydraulic connections. If the volume belonging to a pressure variable is variable (e.g. with a cylinder), adjustment of the volume is required in the node with which the pressure connection is linked. Similar to the limit function, a second function to this variable is defined which corrects the volume of the node. This function is called by the variable name to which the ending *VolumeCorrection* is added. The result of the volume correction function returned by *return* is not integrated.

  If the status equation of a variable is formed, it is checked whether a correction function to this variable exists and can be called, if necessary.

  Example:

```
class Cylinder
...
// Functions
// p1
LDOUBLE Volumflow1(SIMSTRUCT* S)
{
...
}
// p1
  LDOUBLE p1VolumeCorrection(SIMSTRUCT* S)

    {

      return ((*stroke - strokemin) * Area1);
    }
```

3.8 End of component description

The keyword *// End* marks the end of the component description:

```
// End
```

4 Standard parameters and standard starting values

All component parameters and variables have to be pre-allocated with values prior to the simulation start. In the *StdLib3.par* file or in the *User3.par* file, standard values for every component parameter and standard starting values for every variable are stored with which the parameters and variables of the components can be pre-allocated after creating the model.

Furthermore, the units which are to be allocated in series to the respective parameters and variables as well as the related conversion factors to the SI-unit are determined in the **.par* files.

The lists of the parameters and variables of the components in the **.par* files is made up as follows:

```
element Component Name
// Parameter
  Parameter  Standard value  Conversion factor  Standard unit
  ...
// Variables
  Variable   Standard value  Conversion factor  Standard unit
  ...
// End
```

Notes:

The conversion factor is calculated from

$$Conversion\ \ factor = \frac{Standard\ \ unit}{SI\ \ unit}.$$

Example:

```
element ServoValve22P
// Parameter
  VolumeFlow         30.      1.66666667E-5    l/min
  PressureDifference 35.      1.E5             bar
  Strokemax          1.       1.E-3            mm
  Strokemin          0.       1.E-3            mm
  Overlap            0.       0.01             %
  InputMax           1.       1.
```

 © FLUIDON® GmbH 2008

```
// Variables
  pP                0.        1.E5              bar
  pA                0.        1.E5              bar
  Input             0.        1.
// End
```

5 Unit file

In DSH*plus,* all the parameters and variables can be put in and put out in any possible unit. The units available in series in DSH*plus* are arranged in subassemblies in the SI unit together with their conversion factors and stored in the *dshplus.ein* or *dsheplus.ein* file (excerpt):

```
/* File dsheplus.ein */

//Group                 Unit                Factor
Length                  m                   1
Length                  dm                  0.1
Length                  cm                  0.01
Length                  mm                  0.001
Length                  microm              0.000001
Length                  in                  0.0254

Velocity                m/s                 1
Velocity                mm/s                0.001
Velocity                in/s                0.0254

Acceleration            m/s^2               1
Acceleration            mm/s^2              0.001
```

Note:

- The conversion factor into the SI unit is calculated from

$$Conversion\ \ factor = \frac{Standard\ \ unit}{SI\ \ unit}.$$

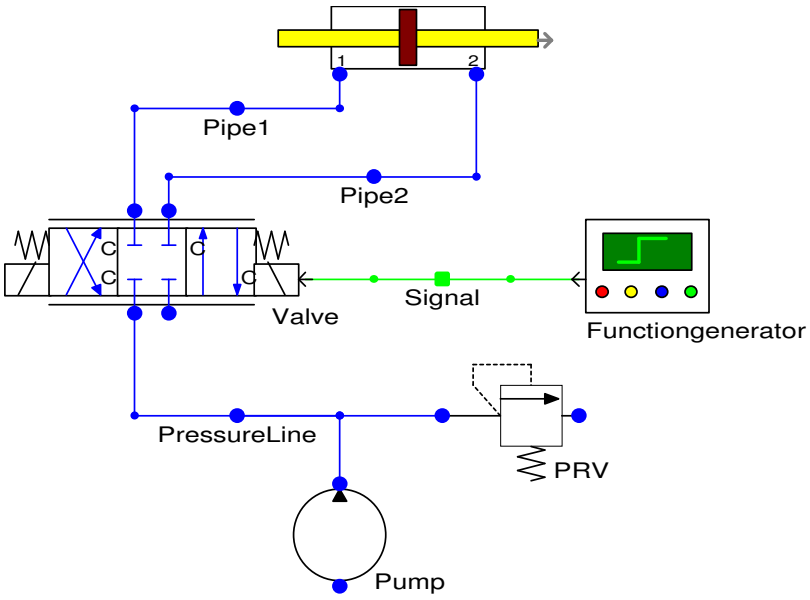
6 Structure of the source code of a simulation model

DSH*plus* offers the user the possibility to see an existing model on the level of the source code and to intervene there. This enables adaptation of the model variants pre-set by the component library to the re-

quirements of the respective user or can be used for better understanding the system by trying to understand the model mathematics. In contrast to the extension of the component library, intervention into the source code of a model requires further knowledge on the model structure as the user now has to process functions which run in the background when the model is automatically created by means of DSH*plus.*

Input of simulation models in DSH*plus* is effected in the input mode which is activated by the menu point **NEW...** in the **PROJECT** menu or by the menu point **PROCESSING** in the **MODEL** menu. In the input mode, the component library is open which enables the user to create new simulation models in the form of a circuit diagram. This circuit diagram forms the basis of the further automatic creation of a model.

The model source code established by DSH*plus* is subdivided into two large zones which on their part are also subdivided into smaller functional units. The first zone includes the descriptions of the individual components used in the way they are stored in the library (see chapter 6, 3). The second zone includes all the functions created for the calculation of the entire simulation model. These are in particular the component definition, initialization, the exit function, the limit values, the differential equation function, the limits and the data function.



Circuit diagram of the example model

 © FLUIDON® GmbH 2008

The individual sections of the model source code are described in detail in the following by means of the sample model shown on the right.

A cylinder is actuated by means of a 4/3 way proportional valve controlled by a function generator. The pressure supply which is secured by a pressure relief valve, is effected by a pump.

The following sections describe the source code which is generated from the graphic structure of the model on the drawing area.

6.1 Header of the model source code

The most important information to the model and the Include files can be seen from the header of the source code of a model.

Example:

```
// Title:       Example model
// Author:      FLUIDON GmbH
// Info:        Example model for the user manual
// Datum:       26.07.1999  10:32

#include "dgltool.h"
```

Explanations to the example:

The include file *dgltool.h* includes the function prototype declarations for the library functions. If required, more include files are added.

6.2 Component descriptions in the model source code

The section *Component description* includes the descriptions of all the components used in the simulation model. They correspond to the source code of the components stored in the component library.

Example:

```
/*-----------------------------------------------------*/
/*  Bauteilbeschreibungen / component description      */
/*-----------------------------------------------------*/

class ServoValve43PCharacteristics {
/*-------------------------------------------------------*/
/* 4/3 directional control valve (Stroke>0 P->A B->T,    */
/* Stroke<0 P->B A->T)                                   */
/* Spool stroke proportional to input signal             */
/* Field of characteristics the aperture cross-section of */
/* each metering edge                                    */
/* (pP, pT, pA, pB, Stroke, Input)                       */
/*-------------------------------------------------------*/
public:

// Parameter
...
};
// End
```

Explanations to the example:

Every component is only listed once, even if it is used several times in the model. For every component, an instance of the category is created.

6.3 Component definition

This part of the program allocates the corresponding component descriptions to the names allocated by the user (Defining an instance of the class).

Example:

```
/*-----------------------------------------------------*/
/*  Bauteildefinitionen / component definition         */
/*-----------------------------------------------------*/
HydroNode                                   PressureLine;
SignalNode                                  Signal;
Pump                                        Pump;
FunctionGenerator                           Functiongenerator;
CylinderWithFriction                        Cylinder;
ServoValve43PCharacteristics                Valve;
LDOUBLE *null;
LDOUBLE *TNorm;
LDOUBLE *EinBarAbs;
```

 © FLUIDON® GmbH 2008

Explanations to the example:

The component designation is written on the right and the name allocated by the user on the left.

6.4 Initialization

In the *Initialization* section, every component is allocated its specific component parameters. All the parameters used in the model are listed and the related value is called from the parameter file.

Furthermore, the auxiliary functions are called here as far as auxiliary functions have been defined in the component descriptions of the elements. In this way, the values which have only to the updated once per simulation run, are calculated in advance.

Example:

```
/*-----------------------------------------------------*/
/*  Initialisierung / Initialization                   */
/*-----------------------------------------------------*/

void Initialisierung (DATA *dat, SIMSTRUCT *S)
{
  if (!null)
  {
     null  = (LDOUBLE *) malloc (sizeof (LDOUBLE));
     *null = 0;
  }

  ...

   Valve.OverlapBT   = LoadParameter(dat, "Valve.OverlapBT");
   Valve.InputMax    = LoadParameter(dat, "Valve.InputMax");

  ...

   Pipe2.AuxiliaryFunction(S);
   Pipe1.AuxiliaryFunction(S);
   PRV.AuxiliaryFunction(S);
   Pump.AuxiliaryFunction(S);
   Functiongenerator.AuxiliaryFunction (S);
   Cylinder.AuxiliaryFunction(S);
   Valve.AuxiliaryFunction (S);
}
```

Explanations to the example:

The syntax for the designation of the parameters is „*Component-Name.Parameter*“.

After the parameters have been loaded, the auxiliary functions are called. The auxiliary functions are designated by „*ComponentI-Name.AuxiliaryFunction()*“.

6.5 Exit function

If dynamic variables are created in a model, they have to be released again in the exit function. They are called after termination of the simulation run.

The exit function is designated by „*ComponentName.ExitFunction()*“.

Example:

```
/*-----------------------------------------------------*/
/*  ExitFunktion / Exit function                       */
/*-----------------------------------------------------*/

void ExitFunktion (SIMSTRUCT *S)
{
  if (null)
  {
     free (null);
     null = NULL;
  }

  ...

   Functiongenerator.ExitFunction(S);
   Valve.ExitFunction(S);
}
```

6.6 Limit values

The minimum and maximum values of all variables are listed in the limit values function. If an upper limit or lower limit of a variable exists, these values are listed here. Otherwise, *-1 bar* is indicated as lower limit for pressure variables and *+/- 1.E99 (+/- 1.E99* corresponds to infinite) for other variables.

 © FLUIDON® GmbH 2008

After every integration step, it is checked whether the limit values are exceeded. If a limit value is exceeded, the exceeded value is limited and the function *Limitations* is called.

Example:

```
/*-------------------------------------------------*/
/*  Grenzwerte / Limits                            */
/*-------------------------------------------------*/

void Limitations (GRENZWERT *yGrenz, )
{
   yGrenz[0].Min = -1E5;
   yGrenz[0].Max = 1.E99;
   yGrenz[1].Min = -1E5;
   yGrenz[1].Max = 1.E99;
   yGrenz[2].Min = -1E5;
   yGrenz[2].Max = 1.E99;
   yGrenz[3].Min = -1.E99;
   yGrenz[3].Max =  1.E99;
   yGrenz[4].Min = Cylinder.strokemin;
   yGrenz[4].Max = Cylinder.strokemax;
}
```

6.7 Reset update flag

This is an auxiliary function that will be used together with future program functionality.

Example:

```
void ResetUpdateFlag (SIMSTRUCT *S)
{

}
```

6.8 Differential equations

The function *differential equations* is the central element of the simulation model and also includes, besides the actual differential equations, the algebraic equations. First of all, the current status values are allocated to the variables. This includes also external inputs to the simulation model, that are taken from the simulation input vector (&SimIn[i]).

Then, all the algebraic values are updated and the differential equations are called afterwards. Finally, the derivations are transferred to the derivation vector. The current time, the status values and the derivation vector form the connection elements to the integrator.

In the differential equation part, allocation of the values is effected to the variables from the central integration program. *t* represents the current integration time, **y* is the pointer of the variables and **f* includes the pointer to the derivation values of the variables.

Example:

```
/*-----------------------------------------------------*/
/*  Differentialgleichungen / Differential equations */
/*-----------------------------------------------------*/

void Dgl (LDOUBLE t, LDOUBLE *y, LDOUBLE f, LDOUBLE h,
          LOUDLBE *SimIn, SIMSTRUCT *S)
{
// Rücksetzen des Updateflags / reset of updateflag
   ResetUpdateFlag(S);

// Initialisierung der Knoten / Initialisation of nodes
   PressureLine.Pressure          = y;            //0
   Pipe2.Pressure                 = ++y;          //1
   Pipe1.Pressure                 = ++y;          //2
// Initialisierung integrierte Variablen
// Initialisation of integral variables
// Zuweisungen Knoten-Element / assignment node-element
   PRV.pT                         = null;
   PRV.pP                         = PressureLine.Pressure;
   Pump.pP                        = PressureLine.Pressure;
   Pump.pT                        = null;
   Functiongenerator.Input        = null;
   Cylinder.p1                    = Pipe1.Pressure;
   Cylinder.p2                    = Pipe2.Pressure;
   Cylinder.velocity              = ++y;              //3
   Cylinder.stroke                = ++y;              //4
   Cylinder.Load                  = null;
   Cylinder.MechF                 = null;
   Valve.pP                       = PressureLine.Pressure;
   Valve.pA                       = Pipe1.Pressure;
   Valve.pB                       = Pipe2.Pressure;
   Valve.pT                       = null;
   SignalEingang.Input1           = &SimIn[0];
   SignalEingang.Input2           = &SimIn[1];
   SignalEingang.Input3           = &SimIn[2];

// Initialisierung algebraische Knoten
// Initialisation of algebraic nodes
   Signal.Value                   = ++y;              //5

// Initialisierung algebraische Variablen
// Initialisation of algebraic variables
   Valve.Input                    = Signal.Value;

// Ausgabe algebraische Berechnungen
```

 © FLUIDON® GmbH 2008

```
// Output of algebraic calculations
   *Signal.Value              = Functiongenerator.Output (S);

// Ausgabe fluidische Gleichungen / Output of fluid equations
   PressureLine.PressureDeriv = PressureLine.EOil(S)
               / PressureLine.Volume
               *( PRV.VolumeFlowP (S)
                + Pump.VolumeFlowP(S)
                + Valve.VolumeFlowP (S));

   Pipe2.CorrectionVolume = Cylinder.p2VolumeCorrection(S);
   Pipe2.PressureDeriv = Pipe2.EOil(S) /
                                        Pipe2.ActualVolume(S)
               *( Cylinder.VolumeFlow2(S)
                + Valve.VolumeFlowB (S));

   Pipe1.CorrectionVolume = Cylinder.p1VolumeCorrection(S);
   Pipe1.PressureDeriv = Pipe1.EOil(S) /
                                        Pipe1.ActualVolume(S)
               *( Valve.VolumeFlowA ()
                + Cylinder.VolumeFlow1(S));
// Ausgabe der sonstigen Integrationsgleichungen
// Output of other integral equations

// Ausgabe der Ableitungen / Output of derivations
   f[0] = PressureLine.Pdot(S);
   f[1] = Pipe2.Pdot(S);
   f[2] = Pipe1.Pdot(S);
   f[3] = Cylinder.velocityDeriv(S);
   f[4] = Cylinder.strokeDeriv(S);
}
```

Explanations to the example:

The allocation list of the variables is started with the volume node which is followed by the variables of other components and the signal nodes. Every variable of a component is either allocated a pointer to a variable of a node or a function number for the derivation vector. The pointers are constantly incremented. For orientation, the serial number is shown as comment in the same line. If a variable is not connected, it is allocated the value *0* (e.g. *Valve.pT = zero*).

Then, the values of the pointers are updated, the derivations of the pressures in the nodes are calculated and the derivations are allocated to the return vector which is integrated.

6.9 Output function

Ín the *Output-function* section, the algebraic value are updated, before the values to the DSH*plus* simulation output vector (SimOut[i]) will be assigned.

Example:

```
/*-------------------------------------------------*/
/*  Ausgabefunktion / Output function              */
/*-------------------------------------------------*/

void Output (LDOUBLE t, LDOUBLE *y, LDOUBLE h, LDOUBLE
*SimIn, LDOUBLE *SimOut, SIMSTRUCT *S)

{
// Rücksetzen des Updateflags / reset of updateflag

   ResetUpdateFlag(S);

// Initialisierung der Knoten / Initialisation of nodes
   Volume9.Pressure                  = y;        // 0
   Volume8.Pressure                  = ++y;      // 1
   Volume7.Pressure                  = ++y;      // 2

// Initialisierung integrierte Variablen / Initialisation of
integral variables
// Zuweisungen Knoten-Element / assignment node-element
   Constant1.Input                   = null;
   Hydro2signal4.pA                  = Volume4.Pressure;
   Hydro2signal4.Q                   = null;

// Initialisierung algebraische Variablen / Initialisation of
algebraic variables
   Sum5.Input3                       = upper_Signal5.Value;
   Sum5.Input1                       = upper_Signal4.Value;

// Ausgabe algebraische Berechnungen / Output of algebraic
calculations
   *upper_Signal5.Value              = FuncGen5.Output (S);
   *upper_Signal4.Value              = FuncGen4.Output (S);
   .
   .
   .
   SimOut[0]         = SignalAusgang1.EingangsFunktion1 (S);
   SimOut[1]         = SignalAusgang1.EingangsFunktion2 (S);
   SimOut[2]         = SignalAusgang1.EingangsFunktion3 (S);
   .
   .
   .
}
```

 © FLUIDON® GmbH 2008

6.10 Limitations

In the *Limitations* section, the value limitations of the variables which are integrated are updated.

Example:

```
/*-----------------------------------------------------*/
/*  Begrenzungen / Limitations                         */
/*-----------------------------------------------------*/

void Limitations (LDOUBLE *y, SIMSTRUCT *S)
{
   Cylinder.velocity     = &y[3];
   Cylinder.stroke       = &y[4];
   Cylinder.Load         = null;
   Cylinder.MechF        = null;

   y[3] = Cylinder.velocityLimit(S);
}
```

6.11 Data

Dimensioning of the variable fields is effected in the *Data* section. The number of hydraulic nodes as well as the total number of variables and the total number of integrated variables are mentioned here.

Example:

```
/*-----------------------------------------------------*/
/*  Daten / Data                                       */
/*-----------------------------------------------------*/

VARDATA* Data()
{
   VARDATA *Pointer;
   Pointer = (VARDATA *) malloc (sizeof (VARDATA));
   Pointer->HydroKnotenCounter          = 3;
                    // Anzahl der Hydraulikknoten
                    // number of HydroNodes
   Pointer->YCounter                    = 6;
                    // Gesamtzahl der Variablen
                    // total number of variables
   Pointer->IntKnotenCounter            = 5;
                    // Gesamtzahl der integrierten Variablen
                    // total number of integrated variables
   return (Pointer);
```

```
}
```

 © FLUIDON® GmbH 2008

6 Programming of new components

The previous chapter explained the basic functioning of DSH*plus* and the basic structure of the component and model source code. The following chapters are to show how the user of DSH*plus* can program new components and add them to the library by him/herself. Therefore, the DSH*plus* specific definitions and functions and the possibility of integrating external functions is explained. And finally, the necessary settings for the compiler and the linker are explained in detail.

1 Extension of the component library

The following sections give an explanation of the creation of a new component. First of all, the programming language and the most favorable procedure with programming of new components are described, and then the rules which have to be taken into account when programming new components are shown.

1.1 Programming language of the component library

The components of the component library are stored as C++ source code in DSH*plus*. A corresponding syntax is used which is integrated into a DSH*plus* systematic. Mathematical functions are used according to the definitions in *deftyp.h* or as per the C standard.

1.2 Programming of new components

The best way for programming a new component is to adapt a component from the DSH*plus* specific library which is similar to the new component. In this way, the program structure and perhaps also the parameters and variables and already existing functions and algorithms can be used.

No modifications should be carried out in the files of the DSH*plus* specific library! These libraries are updated with a program update and this means that the data of the modified components would be lost. It is therefore recommended to copy the components which have to be

 © FLUIDON® GmbH 2008

modified into the files for user-defined components (*AnyName*.*) at any case and to carry out the modifications there.

Proceed as follows in order to modify a component:

1. Copy the source code of the component which you want to modify in your *AnyName.dat* file. You can get the source code from the source of the simulation model in which you have used the pattern component.

2. Change the component name (*class NewComponent*) of the component copied into the file *AnyName.dat* and in the comment lines, give a short description of your new component or the modifications made with regard to the old component.

3. Make the adaptation of the source code.

4. Save the changes made in the *AnyName.dat*.

5. Open the file *StdLib3.par* (library path) and copy the parameters of the component which is used as model into the file *AnyName.par*.

6. Change the component name (*element NewComponent*) of the component copied into the file *AnyName.par*.

7. Under *// Parameters,* enter the parameters of the new component, their standard values, the conversion factors, and the standard units (separated by tab stops) and delete unnecessary parameters.

```
element NewComponent
// Parameter
  Parameter   Standard value     Factor    Standard unit
  ...
// Variables
  Variable    Standard value     Factor    Standard unit
  ...
// End
```

Proceed in the same way with the variables: Under *// Variables,* enter the variables of the new component, their standard values, the

conversion factors and the standard units (separated by tab stops) and delete unnecessary variables.

Save the *AnyName.par* file.

8. If some units are required for your new component which are not yet included in the *StdLib3.ein* file, include them into your *user3.ein* file. To do this, into the *AnyName.ein* file enter the unit group to which you want to allocate your new unit, the unit and the conversion factor to the SI unit (separated by tab stops). If for example Decimetre had been entered as new unit, it would have been allocated to the unit group length. The conversion factor would be Meter 0,1 relative to the SI unit.

   Example:

   Length ➔ dm ➔ 0.1

```
//Group                Unit              Factor
Unit group             Unit              Conversion factor
```

   Save the *AnyName.ein* file.

9. Call the component editor of DSH*plus* for designing a new symbol. Create a symbol for the new component and allocate it the component name *NewComponent*. Allocation of the component to a group is also effected in the component editor.

10. For controlling the new component, create a test model in DSH*plus* which enables you to check the functioning of the new component.

11. Create and compile the model with the program function **Create + Compile** in the dialog **Status model build up** which is called by means of the menu point **Create model...** in the **Model** menu.

12. If error messages are displayed in the compiler window or if the new component does not yet show the behaviour required in the test simulations, correct the source code of the component in your *user3.dat* file.

 © FLUIDON® GmbH 2008

13. Save the *AnyName.dat* file after the modifications have been made.

14. Repeat steps 12 and 14 as long as all error messages are remedied and the component shows the behaviour required in the test simulations.

Notes:

- If the editor you use has a compiler or the compiler is accessible from the editor, it is recommendable to compile the source code of the new component first and eliminate programming errors before checking the behaviour of the component in a test model in DSH*plus*.

- Under certain circumstances it can be useful to carry out debugging in the source code of the entire model. This procedure is described in chapter 6, .

1.3 Programming rules

Observe the following rules when programming a new component:

- The name of the component is not to have any blanks.

- With parameters, variables and auxiliary parameters, observe the use of small and capital initial letters (*Stroke* and *stroke* are different variables!).

- Only block comments are to be used in the component description (*/* Comment */*). Line comments (*// Comment*) are not permitted as „//“ is reserved for marking of keywords.

- No comments are to be made in the parameter file.

- Please see that a blank is made between // and the keyword.

- The parameters for the limit values of a variable consist of the variable name to which *min* or *max* is added (e. g. *Strokemin*, *Strokemax*).

- Output values are only declared as variables if they are used for calculation of other output values.

- With the component INTEGRATOR, the variable *Output* is not required in another component function. It is therefore not listed under variables but only as function comment.

Examples:

```
class Integrator
...
// Variables
LDOUBLE *Input;
...
// Functions
// Output
LDOUBLE OutputFunction(SIMSTRUCT *S)
{
   return (*Input);
}
...
```

- If the output variable is used in a component function, it is also to appear in the variable list. This is the case with the PT1 component as the variable Output is used for calculation of the output signal itself.

```
class PT1 {
...
// Variables
  LDOUBLE *Input;
  LDOUBLE *Output;

...
// Functions
// Output
  LDOUBLE OutputDeriv(SIMSTRUCT *S)
  {
       return ((Kp * *Input - *Output) / T);
  }
};
// End
```

- Algebraic values are not to be listed as variables.

 © FLUIDON® GmbH 2008

- Hydraulic connections have to be marked with names starting with „*p*“. The related pressure variables have the same name as the hydraulic connection.

- Pneumatic connections have to be marked with names starting with „PN“. The related pressure variables have the same name as the connection, the temperature variable is composed of a „T“ and the name of the connection without „PN“ (PN1, T1).

- The designations of signal connections are not subject to any rule. The related variables need to have the same names in the source code.

- The names of the mechanic connections can also be chosen freely. With the designation of the three components of a mechanical connection, however, some particularities have to be observed (see following item).

- The variables of the three components of a mechanical connection are marked with the name of the connection to which, for translatoric signals, „*x*“ is added for the stroke, „*v*“ for the velocity and „*F*“ for the force. With rotary signals, „*phi*“ is added for the angle, „*omega*“ for the angular velocity and „*M*“ for the moment.

- In the DSH*plus* specific library, mechanical connections are designated with „*Mech*“ or „*Mech1*“ and „*Mech2*“. This, however, is not absolutely necessary.

Note:

- In the variable list of the components, only components of mechanical connections which are input variables, may appear.

2 DSH*plus* specific definitions and functions

For creating of components, different definitions and functions have been stored in DSH*plus* which are to facilitate programming. They are part of the link library dsh36_DSHTools and they are defined in the

header files *deftyp2.h, dglstruc.h, dgltool.h, dglthermotools.h, dglpipes.h, pneutools.h* and *meldung.h.*

These definitions and functions are shown in the following chapters.

2.1 General

Integration of the simulation time into the source code

The current simulation time *t* can be integrated as variable into the source code of a component. The function can be called as follows:

```
// Functions
// Output algebra
LDOUBLE OutputFunction (SIMSTRUCT *S)
{
   Simulationtime = S->t;
   ...
}
```

Integration of the step size into the source code

The current simulation step size *h* of the integrator can be transferred as functional parameter. The function can be called as follows:

```
// Functions
// Output algebra
LDOUBLE OutputFunction (SIMSTRUCT *S)
{
   StepSize = S->h;
   ...
}
```

Stage of the integration method

The stage (first or other) of the integration method can be called as function parameter. The function call is:

 © FLUIDON® GmbH 2008

```
// Functions
// Output algebra
LDOUBLE OutputFunction (SIMSTRUCT *S)
{
   if (S->Flags.FirstStep == true)
   {
      ...;
   }
   else
   {
      ...;
   }
   ...;

}
```

FirstStep is true if the differential equation is called in the first stage of the integrator, in other cases it is false.

Auxiliary function dependent on starting values

Should certain functions which depend on the variables are only to be carried out at the beginning of the simulation, they cannot be carried out under *// AuxiliaryFunction* as the variables have not yet been assigned. They have to be calculated by a function which is only be carried out once at the beginning of the simulation.

How to proceed:

1. Define the auxiliary parameter *FirstRun*.

```
// AuxiliaryParameter
  int FirstRun;
```

2. Initialize the auxiliary parameter in the auxiliary function to 1.

```
// AuxiliaryFunction
   void AuxiliaryFunction(SIMSTRUCT *S)
   {
      FirstRun = 1;
      ...
   }
```

3. Have the function only calculated if *FirstRun = 1* and then set *FirstRun* to 0.

```
// Functions
// VariableName
   LDOUBLE VariableNameFunction(SIMSTRUCT *S)
   {
      if (FirstRun)
      {
         ...
         FirstRun = 0;
      }
   }
   ...
};
```

2.2 Deftyp2.h

The file *deftyp2.h* contains macros and definitions that allows an independence of different compilers and a simplification of modifications.

2.2.1 Data format depending functions

In order to avoid compatibility problems with different data formats (80 Bit or 64 Bit), the most important mathematical functions have been newly defined in DSH*plus*. These functional macros should be always used.

```
#define LDOUBLE long double
#define SIN     sinl
#define ASIN    asinl
#define COS     cosl
#define ACOS    acosl
#define TAN     tanl
#define ATAN    atanl
#define ATAN2   atan2l
#define HYPOT   hypotl
#define SQRT    sqrtl
#define EXP     expl
#define LOG     logl
#define LOG10   log10l
#define POW     powl
#define POW10   pow10l
#define CEIL    ceill
#define FLOOR   floorl

#define FMOD    fmodl
```

 © FLUIDON® GmbH 2008

```
#define MODF    modfl
#define FABS    fabsl
#define STRTOD _strtold
#define ATOF   _atold
#define EPSILON LDBL_EPSILON
#define EPS     FLT_EPSILON
#define EPS100  100 * EPSILON
```

Example:

In the source code of a component, the definition of a *Parameter* of the data type *long double* is:

```
// Parameter
  LDOUBLE Parameter;
```

The function for calculation of the absolute value of a *Variable* of the data type *long double* is:

```
AbsoluteValue = FABS (Variable);
```

2.2.2 Mathematical constants

Furthermore, fixed numeric values are allocated to frequently used constant values - especially constant terms from the constants π and e. Calculation time can be saved in this way as the values have not to be calculated again with every calculation step.

```
#define M_E          2.71828182845904523536
#define M_LOG2E      1.44269504088896340736
#define M_LOG10E     0.434294481903251827651
#define M_LN2        0.693147180559945309417
#define M_LN10       2.30258509299404568402
#define M_PI         3.14159265358979323846
#define M_PI_2       1.57079632679489661923
#define M_PI_4       0.785398163397448309616
```

```
#define M_1_PI       0.31830988618379067153
#define M_2_PI       0.63661977236758134308
#define M_1_SQRTPI   0.56418958354775628695
#define M_2_SQRTPI   1.12837916709551257390
#define M_SQRT2      1.41421356237309504880
#define M_SQRT_2     0.70710678118654752440
```

Explanations:

Constant	Mathematical notation	Value
M_E	e	*2.71828182845904523536*
M_LOG2E	$\log_2 e$	*1.44269504088896340736*
M_LOG10E	$\log e$	*0.43429448190325182765*
M_LN2	$\ln 2$	*0.69314718055994530942*
M_LN10	$\ln 10$	*2.30258509299404568402*
M_PI	π	*3.14159265358979323846*
M_PI_2	$\frac{\pi}{2}$	*1.57079632679489661923*
M_PI_4	$\frac{\pi}{4}$	*0.78539816339744830962*
M_1_PI	$\frac{1}{\pi}$	*0.31830988618379067154*
M_2_PI	$\frac{2}{\pi}$	*0.63661977236758134308*
M_1_SQRTPI	$\frac{1}{\sqrt{\pi}}$	*0.56418958354775628695*
M_2_SQRTPI	$\frac{2}{\sqrt{\pi}}$	*1.12837916709551257390*
M_SQRT2	$\sqrt{2}$	*1.41421356237309504880*
M_SQRT_2	$\frac{\sqrt{2}}{2}$	*0.70710678118654752440*

© FLUIDON® GmbH 2008

Example:

The formula for the calculation of the cross-sectional area of a cylinder piston from the parameter *Piston diameter* in the source code of a component is:

```
Area = M_PI_4 * sqr (PistonDiameter);
```

2.2.3 Mathematical functions

In addition, some functions suitable for hydraulics and which do not belong to the standard functions of C++, have been defined and are available to the user with programming (Macro definitions).

```
/* File deftyp2.h */

#ifdef __cplusplus
#define max(x,y)  (((x) > (y)) ? (x) : (y))
#define min(x,y)  (((x) > (y)) ? (y) : (x))
#endif

#define sign(x)   (((x) < 0) ? -1 : 1)
#define psqrt(x)  (SQRT (FABS (x)) * sign (x))
#define sqr(x)    ((x) * (x))
#define cub(x)    ((x) * (x) * (x))
#define h4(x)     ((x) * (x) * (x) * (x))
#define h5(x)     ((x) * (x) * (x) * (x) * (x))
```

Explanations:

Function call	Mathematical notation
max(x,y)	$Max(x,y) = x$ for $x > y$ $max(x,y) = y$ for $x \leq y$
min(x,y)	$min(x,y) = x$ for $x \leq y$ $min(x,y) = y$ for $x > y$
sign(x)	$sign(x) = -1$ for $x < 0$ $sign(x) = 0$ for $x = 0$ $sign(x) = 1$ for $x > 0$
psqrt(x)	$psqrt(x) = sign(x) \cdot \sqrt{\lvert x \rvert}$

Function call	Mathematical notation
sqr(x)	$sqr(x) = x^2$
cub(x)	$cub(x) = x^3$
h4(x)	$h4(x) = x^4$
h5(x)	$h5(x) = x^5$

Note:

- The function *psqrt (x)* is required for calculation of the Bernoulli equation in order to be able to find the roots of negative pressure differences.

2.3 dglstruc.h

The header file *dglstruc.h* defines some structures that are used in the simulation model and for the transfer of data from DSH*plus* to the simulation model.

Since the structures are used just in the automatic functions and not in user defined components, they will not be explained in detail.

2.4 dgltool.h

With the header file dgltool.h the user has a number of functions that are often used in the hydraulics and in the DSH*plus* simulation models. These functions can be called while programming new components, they need not to be newly implemented every time.

2.4.1 Hydraulic functions

Coefficient of an orifice

The coefficient of an orifice can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcCoefficient.* The function can be called as follows:

 © FLUIDON® GmbH 2008

```
Value = CalcCoefficient (VolumeFlow, PressureDifference);
```

Note:

The function *CalcCoefficient* is calculated as per the formula

$$Value = \frac{VolumeFlow}{\sqrt{PressureDifference}}$$

Coefficient of an adjustable orifice

The coefficient of an adjustable orifice can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcAdjustCoefficient.* The function can be called as follows:

```
Value = CalcAdjustCoefficient (VolumeFlow,
                               PressureDifference, Signal);
```

Note:

The functional parameter *Signal* is the maximum input signal of the function. The coefficient refers to the maximum opening of the orifice.

The function *CalcAdjustCoefficient* is calculated as per the formula

$$Value = \frac{VolumeFlow}{\sqrt{PressureDifference} \cdot Signal}$$

Conductance of a throttle

Conductance of a throttle can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcConductance.* The function can be called as follows:

```
Value = CalcConductance (VolumeFlow, PressureDifference);
```

Note:

The function *CalcConductance* is calculated as per the formula

$$Value = \frac{VolumeFlow}{Pressure Difference}$$

Conductance of an adjustable throttle

Conductance of an adjustable throttle can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcConductance.* The function can be called as follows:

```
Value = CalcAdjustConductance (VolumeFlow,
                               PressureDifference, Signal);
```

Note:

The functional parameter *Signal* is the maximum input signal of the function. The conductance refers to the maximum opening of the throttle.

The function *CalcAdjustConductance* is calculated as per the formula

$$Value = \frac{VolumeFlow}{Pressure Difference \cdot Signal}$$

Conductance of a technical resistor

Conductance of a technical resistor can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcConductanceExpo*. The flow characteristics is adjusted by means of the exponent (*Expo*). The function can be called as follows:

```
Value = CalcConductanceExpo (VolumeFlow, PressureDifference,
                             Expo);
```

 © FLUIDON® GmbH 2008

Note:

The function *CalcConductanceExpo* is calculated as per the formula

$$Value = \frac{VolumeFlow}{(PressureDifference)^{Expo}}$$

The exponent is in the range of *Expo = 0,5..1*.

Conductance of an adjustable technical resistor

Conductance of an adjustable technical resistor can be calculated from its parameters *Volume flow* and *Pressure difference* by means of the function *CalcAdjustConductanceExpo*. The flow characteristics is adjusted by means of the exponent (*Expo*). The function can be called as follows:

```
Value = CalcAdjustConductanceExpo (VolumeFlow,
                                   PressureDifference,
                                   Signal, Expo);
```

Note:

The functional parameter *Signal* is the maximum input signal of the function. The conductance refers to the maximum opening of the resistor.

The function *CalcAdjustConductanceExpo* is calculated as per the formula

$$Value = \frac{VolumeFlow}{(PressureDifference)^{Expo} \cdot Signal}$$

The exponent is in the range of *Expo = 0,5..1*.

Volume flow through variable orifice

The function *VarFlow* calculates the volume flow through a variable orifice with stroke offset (overlapping) from its parameters *Coefficient* and

Stroke Overlapping and from the variables *Pressure difference* and *Stroke.* The function can be called as follows:

```
Value = VarFlow (Coefficient, PressureDifference, Stroke,
                 StrokeOverlapping);
```

Notes:

The volume flow through the orifice is calculated with every simulation step. The function is used for flow calculation with valves.

Volume flow through a circular section or triangular section

For realizing volume flows through specific area sections (triangle, circle), a particular category exists. In DSH*plus,* the function *TFlow-ThroughArea* is available with which volume flows and the arising flow forces can be calculated by means of circular and triangular sections.

If this function is to be used in a component, the following rules have to be taken into account when creating the source code:

1. The parameters which are to remain variable after model formation have to be declared in the parameter list.

```
// Parameter
   int  TriangleCircle /* Triangle: TriangleCircle = 1 */
                       /* Circle: TriangleCircle = 0   */
   int  Quantity       /* Number of triangular or      */
                       /* circular cross sections      */
   LDOUBLE AlphaD      /* Flow coefficient             */
   LDOUBLE Rho         /* Density of the fluid         */
   LDOUBLE Height      /* Triangle: Height             */
                       /* Circle: Diameter             */
   LDOUBLE Width       /* Triangle: Width              */
                       /* Circle: Width = 0)           */
   LDOUBLE Epsilon     /* flow angle for the flow      */
                       /* force calculation            */
   ...
```

 © FLUIDON® GmbH 2008

2. In the list of the auxiliary parameters, a *Special section* of the type *PTFlowThroughArea* is defined for every specific section.

```
// AuxiliaryParameter
   PTFlowThroughArea SpecialCrossSection;
   LDOUBLE FlowForce;

   ...
```

3. In the list of the variables, the pressure in front of the special section (*pP*), the pressure behind the special section (pA) and the stroke have to be defined as variables.

```
// Variables
   LDOUBLE *pP;
   LDOUBLE *pA;
   LDOUBLE *Stroke;
   ...
```

4. For every volume flow through a specific section, create an instance of the type *TFlowThroughArea* to which the parameters are transferred in the auxiliary function by means of the function new.

```
// AuxiliaryFunction
void AuxiliaryFunction (SIMSTRUCT *S)
{
   SpecialCrossSection = new TFlowThroughArea
                        (TriangleCircle, Quantity, AlphaD,
                         Rho, Height, Width, Epsilon);
}
```

5. This instance has to be released again in the exit function (delete).

```
// ExitFunction
void ExitFunction(SIMSTRUCT *S)
{
   delete (SpecialCrossSection);
}
```

6. The functions for the volume flow and the flow force are called in the functional part of the component descriptions:

```
// Functions
// Output algebra
LDOUBLE OutputValue(SIMSTRUCT *S)
{

   return ((*SpecialCrossSection)(*Stroke, *pP - *pA));

}

// velocity
LDOUBLE velocityDeriv(SIMSTRUCT *S)
{
   FlowForce  = SpecialCrossSection->FlowForce (*Stroke,
                                                *pP - *pA);
   ...
}
```

Note:

- The designations *TFlowThroughArea* and *TFlowThroughArea** are identical.

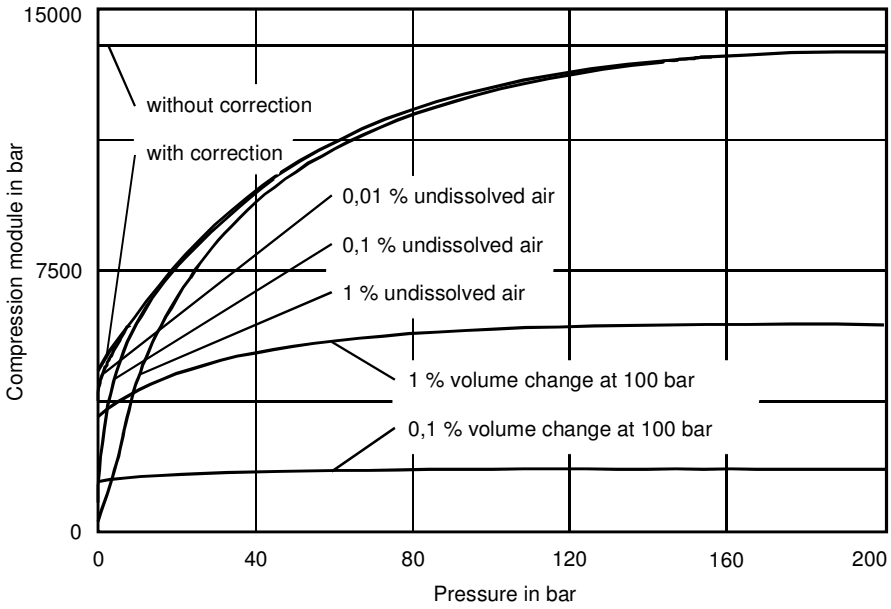
Compression modulus of the fluid

Dynamic calculation of a fluid power system is done in DSH*plus* using a variable pressure dependent compression module of the fluid (replacement compression module). For pressure calculation in the node, a compression module is used which is dependent on the portion of undissolved air, on the percentage of the volume change with a reference pressure and on the pressure.

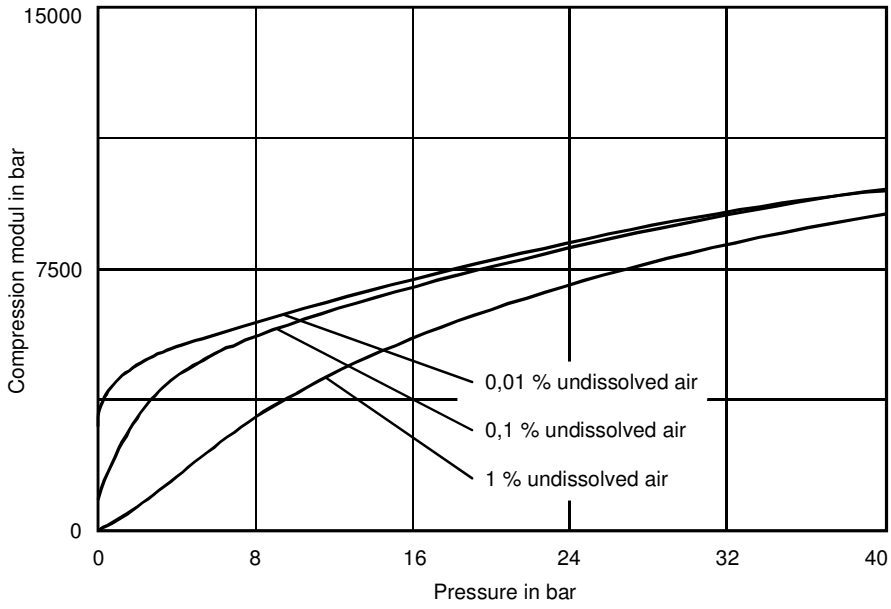
Up to a limit of *0.01 mbar*, an expansion of undissolved gas in oil is calculated, then the compression module is kept constant.

The following illustrations show the dependence of the compression module.

 © FLUIDON® GmbH 2008



Compression module of fluid in the range of 0 to 200 bar



Compression module of fluid in the range of 0 to 40 bar

Note:

Internally, DSH*plus* calculates with relative pressure as is customary in hydraulics and not with absolute pressure. Thus, the lower limit of the pressure is *-1 bar*. ($p_{DSHplus} = p_{absolute} - p_{Atmosphere}$)

The functions *EOelCorrection* and *EOel_Correction* are available for integration of the compression module of the fluid into the source code of the components.

The function *EOel_Correction* calculates the replacement compression module of the fluid depending on the pressure and the portion of undissolved air. The function can be called as follows:

```
Value = EOel_Korrektur (EOil, PartUndissAir,
                        PolytropicExponent, *Pressure);
```

The function *EOelCorrection* calculates the replacement compression module of the fluid only depending on the pressure. The function can be called as follows:

```
Value = EOelKorrektur (EOil, *Pressue);
```

A combination of both functions is generally used.

Viscous friction inside a tube

The function *CoefficientOfFriction* calculates the coefficient of viscous friction inside a tube in dependence of the reynolds number (*Re*) and the relative roughness (*relRoughness*) of the tube.

```
Value = CoefficientOfFriction (Re, relRoughness, epsilonf);
```

Note:

The return value is calculated with an iteration. *epsilonf* is a parameter for the accuracy of the iteration.

Laminar factor

The function LaminarFactor calculates a correction factor depending on the pressure difference. This factor limits the gradient of the orifice flow for low pressure differences. Thus the numerical stability of the system and the function of the step size control increase.

 © FLUIDON® GmbH 2008

```
Value = LaminarFactor (DeltaP);
```

Note:

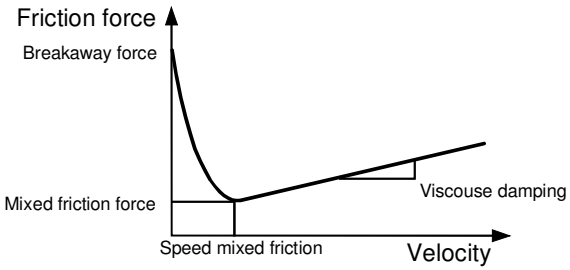
The use of the function is just useful in combination with an orifice flow. With laminar flow the function has no effect.

2.4.2 Other functions

In DSH*plus,* a number of general functions are available which are constantly used. They can be called when new components are programmed and do not have to be re-implemented every time. These functions are explained in the following sections.

Velocity dependent friction

Velocity dependent friction can be calculated by means of the function *StribeckFrictionForce* according to the Stribeck graph.



Stribeck graph

The function can be called as follows:

```
Value = StribeckFrictionForce (*velocity, FOutside,
                               BreakawayFrictionForce,
                               MixedFrictionForce,
                               SpeedMixedFriction,
                               ViscousDamping);
```

Notes:

The course of the Stribeck graph is defined by means of the parameters *BreakawayForce*, *MixedFrictionForce*, *SpeedMixedFriction* and *ViscoseDamping*. *FOutside* is the sum of all external forces. It is used as reference force in the calculation of the velocity dependent friction as the friction forces can never be higher than the forces applied.

Look-up tables

Look-up tables indicate the relations between two or more variables which cannot be described by a mathematical function or can only be described with difficulty.

In DSH*plus,* one-dimensional, two-dimensional and three-dimensional look-up tables are available as signal transmitting elements or as instrument for parameterization of components.

A look-up tables is put in via the look-up tables editor which is described in chapter 4, 3.11.

If you want to use look-up tables in a component, some rules have to be complied with when creating a source code:

1. Declare the numbers of the look-up tables in the parameter list as parameters (*FieldOfCharacteristicsVariableName*). The parameter names for the numbers of the look-up tables have to include the word *FieldOfCharacteristic*.

   Look-up tables are allocated by names in the model, however, internally managed by numbers. Thus, an *int* value is indicated as parameter.

```
// Parameter
   int FieldOfCharacteristicsVariableName;
```

 © FLUIDON® GmbH 2008

2. In the list of the auxiliary parameters, declare a *FieldOfCharacteristicsInternalName* of the type *PTCharacteristic* for every look-up tables.

```
// AuxiliaryParameter
   PTCharacteristic FieldOfCharacteristicsInternalName;
```

3. The input values of the look-up tables have to be defined as variables in the list of variables.

```
// Variables
   LDOUBLE *Input;
```

4. For every look-up tables in the auxiliary function generate an instance of the type TCharacteristic by means of the function *new* to which the name and the look-up tables data are transferred (*DatIn* is a file handle which is automatically transferred to the auxiliary function).

```
// AuxiliaryFunction
void AuxiliaryFunction (SIMSTRUCT *S)
{
   FieldOfCharacteristicsInternalName
      = new TCharacteristic (FieldOfCharacteristics
                             VariableName, S->ModellPfad);
}
```

5. This instance has to be released again in the exit function (delete).

```
// Exitfunction
void Exitfunction(SIMSTRUCT *S)
{
   delete (FieldOfCharacteristicsInternalName);
}
```

6. Look-up tables are called by their *FieldOfCharacteristicsInternalName* in the functional part of the component descriptions:

```
// Functions
// Output algebra
LDOUBLE OutputValue(SIMSTRUCT *S)
{
  return ((*FieldOfCharacteristicsInternalName)(*Input));
}
```

Notes:

- The designations *PTCharacteristics* and *TCharacteristics** are identical.

- The procedure for two-dimensional and three-dimensional look-up tables is effected in the same way. Two-dimensional look-up tables are called as follows:

```
Value = (*FieldOfCharacteristicsInternalName)
                                        (*Input1, *Input2);
```

Three-dimensional look-up tables are called as follows:

```
Value = (*FieldOfCharacteristicsInternalName)(*Input1,
                                         *Input2, *Input3);
```

Ring memory

In DSH*plus*, data structures and access functions for pointed lists with one value (*TRing1*) and with two values (*TRing2*) are available which can be used for idle times, for example.

The pointed lists *TRing1* or *TRing2* are structures the elements of which include the simulation time, one or two variables, one pointer to the previous element and one pointer to the subsequent element.

If you want to use a pointed list *TRing1* in a component, some rules have to be complied with when creating a source code:

 © FLUIDON® GmbH 2008

1. In the parameter list, declare the *IdleTime* or the parameters from which the idle time is calculated.

```
// Parameter
   LDOUBLE IdleTime;
```

2. In the list of the auxiliary parameters, a read pointer (*ReadPtr*) and a write pointer (*WritePtr*) of the file type *PTRing1* have to be defined. The auxiliary parameter *Distance* is defined for the number of memory elements between the read pointer and the write pointer. The auxiliary parameter *FirstRun* has to be defined for initialization of the ring memory when simulation is started.

```
// AuxiliaryParameter
   int     Distance;
   PTRing1 ReadPtr;
   PTRing1 WritePtr;
   int     FirstRun;
```

3. In the list of the variables, declare the current values of the function as input variables.

```
// Variables
   LDOUBLE *Input;
```

4. In the auxiliary function, initialize the ring memory with the function CreateRing1 to which the read pointer and the write pointer are transferred.

```
// AuxiliaryFunction
   void AuxiliaryFunction(SIMSTRUCT *S)
   {
      FirstRun = 1;
      Distance = CreateRing1 (&ReadPtr, &WritePtr);
   }
```

5. In the exit function release the ring memory again by means of the function FreeRing1.

```
// ExitFunction
   void ExitFunction(SIMSTRUCT *S)
   {
      FreeRing1 (WritePtr);
   }
```

6. In the functional part of the component description, the values which are behind the current simulation time *t* by the time interval *IdleTime*, are called by means of the function *Ring1Read*. The auxiliary parameter *Distance* is calculated by means of the function *Ring1Write*. When starting the simulation, the pointed list has to be initialized by means of the function *InitRing1*.

```
// Functions
// Output algebra
   LDOUBLE Output (SIMSTRUCT *S)
   {
      if (FirstRun)
      {
         InitRing1 (ReadPtr, S->t, *Input);
         FirstRun = 0;
      }

   Distance = Ring1Schreiben (&ReadPtr, S->t,
                              *Input, Distance);

   return (Ring1Lesen (&ReadPtr, S->t, IdletTime,
                       &Distance));
   }
};
```

Note:

- The designations *PTRing1* and *TRing** are identical.
- The procedure with a ring memory with two values is the same.

Filter

With the class *TFilter* you have access to some funcions which represents a signal filter. In the component manual Control and Mechanics

 © FLUIDON® GmbH 2008

the usage of the filter class is explained in detail with the component ButterworthFilter. Likewise the parameters of the class are explained.

If you want to use a filter in an own component you have to execute the following steps:

1. Declare a parameter of the type TFilter in the list of auxiliary parameters for every filter.

```
// AuxiliaryParameter
  char* ElementName;
  char* ParameterName;
  TFilter *Filter;
  LDOUBLE result;
  LDOUBLE Time;
```

2. Create an instance of type TFilter for the filter with the function new inside the auxiliary function.

```
   if (FilterOrder > 0)
   {
       Filter = new TFilter (FilterFlag,
                             SignalPickupTime,
                             FilterOrder, OmegaC1,
                             AmplificationC, OmegaC2);
   }
   else
   {
       Filter = new TFilter (FilterFlag,
                             SignalPickupTime,
                             OmegaC1, OmegaR1,
                             AmplificationC,
                             AmplificationR, OmegaC2);
   }
```

3. Inside the exit function this instance has to be deleted.

```
// ExitFunction
  void ExitFunction(SIMSTRUCT *S)
  {
     if (Filter != NULL)
        delete Filter;
  }
```

4. With the parameter *SignalPickupTime* and the auxiliary parameter *Time* equidistant points are created that are necessary for the function of the filter.

```
// Output algebra
  LDOUBLE FilterFunction (SIMSTRUCT *S)
  {
     if (Time - S->t < EPS)
     {
        Time = S->t;

        result = (*Filter) (*input);

        Time += SignalPickupTime;
        return (result);
     }
     return (result);
  }
```

GetParameter

The function *GetParameter* loads Parameter of the internal data structure of DSH*plus* by means of the function parameter *name*. The function can load values with the type *LDOUBLE*, *int* and *char*.

```
void GetParameter (DATA* Dat, char* name, LDOUBLE & value);
void GetParameter (DATA* Dat, char* name, int & value);
void GetParameter (DATA* Dat, char* name, char* & value);
```

2.5 dglthermotool.h

The file *dglthermotool.h* contains the class *TFluidParameter*. This class is used for the calculation of oil data both in the thermo-hydraulic library and in the standard hydraulic library. It is normally available in every simulation model. The data of the class are set in DSH*plus* (menu *Parameter / Fluid options*) and transmitted to the model. Inside the model you have access to the central values of viscosity, density, oil temperature, and environment temperature. In order to consider local differences in temperature the functions of viscosity and density have an additional parameter for the temperature difference.

 © FLUIDON® GmbH 2008

In the source code of the model the class is initialised directly after the header files:

```
#include "dgltool.h"

TFluidParameter FluidParameter;
```

In the initialisation function the data are loaded from the parameter list:

```
void Initialisation (DATA *dat, SIMSTRUCT *S)
{
   ...
   GetParameter (dat, "FluidParameter.Typ",
                      FluidParameter.Typ);
   GetParameter (dat, "FluidParameter.Name",
                      FluidParameter.Name);
   GetParameter (dat, "FluidParameter.BasisNue",
                      FluidParameter.BasisNue);
   GetParameter (dat, "FluidParameter.BasisRho",
                      FluidParameter.BasisRho);
   GetParameter (dat, "FluidParameter.T",
                      FluidParameter.T);
   GetParameter (dat, "FluidParameter.TUmgebung",
                      FluidParameter.TUmgebung);
   GetParameter (dat, "FluidParameter.CP0A0",
                      FluidParameter.CP0A0);
   GetParameter (dat, "FluidParameter.CP0A1",
                      FluidParameter.CP0A1);
   ...
   FluidParameter.AuxiliaryFunction (dat);
...
}
```

The access to the oil data inside a component looks like this:

```
Rho           = FluidParameter.getRho (TDifferenz);
KinViscosity  = FluidParameter.getNue (TDifferenz);
```

2.6 dglpipes.h

The file *dglpipes.h* contains the classes for the calculation of tubes and pipes. These classes are only used in special tube-/pipe-models. Therefore they are not explained in detail here.

2.7 pneutools.h

The header file *pneutools.h* defines a number of functions that are often used in pneumatic components. They are helpful to reduce programming effort.

Mass flow

The mass flow through a pneumatic resistor can be calculated with the values pressure *p* and temperature *T* of each side and the parameters *C*, *b,* and polytropic exponent *n*. The function call looks like:

```
value = Massflow (p1, p2, T1, T2, C, b, n);
```

Heat flow

The assimilated or dissipated heat flow of a surface can be calculated with the values temperature of the surface (*TempSurface*) and temperature of the fluid (*TempFluid*) and the parameters *Surface* and the value for the heat transfer (*HeatTransfer*). The function call looks like:

```
value = Heatflow (HeatTransfer, Surface,
                  TempSurface, TempFluid);
```

Calculation of gas mass

The gas mass of a volume can be calculated with the parameters pressure (*p*), volume (*Vol*), and temperature (*Temp*). The function call looks like:

```
value = MassOfGasEquation (p, Vol, Temp);
```

 © FLUIDON® GmbH 2008

Calculation of gas pressure

The pressure of a volume can be calculated with the parameters gas mass (*Mass*), volume (*Volume*), and temperature (*Temp*). The function call looks like:

```
value = PressureOfGasEquation (Mass, Vol, Temp);
```

Calculation of gas volume

The gas volume can be calculated with the parameters pressure (*p*), gas mass (*Mass*), and temperature (*Temp*). The function call looks like:

```
value = VolumeOfGasEquation (Mass, p, Temp);
```

Calculation of gas temperature

The gas temperatur of a volume can be calculated with the parameters pressure (*p*), volume (*Volume*), and gas mass (*Mass*). The function call looks like:

```
value = TempOfGasEquation (Mass, Vol, p);
```

2.8 meldung.h

The header file *meldung.h* is needed for the connection between the model and the protocol function of DSH*plus* (menu *Simulation / Messages*). It contains a list of message types and a declaration of the message function. The model is automatically connected to the protocol function. To achieve this a pointer to the message function is declared and allocated in the initialisation function.

```
#include "dgltool.h"

TFluidParameter FluidParameter;
pMessage Message;

...

void Initialisation (DATA *dat, char* ModellPfad)
{
   ...
   Message = dat->Message;
   ...
}
```

In a component source code the message is used as following:

```
(*Message) (Error, ElementName, "Parameter xy wrong");
```

The message types have the following meaning:

Message type	Explanation
Message	output of a message without opening of the message window, for internal use
Warning	output of a warning message, the color of the component is set to warning (dark red), the message window open not automatically
Error	output of an error message, the color of the component is set to error (red), the message window opens automatically
sLeer	output of a line, for internal use
Statistik	for internal use only
SimStop	immediate stop of the simulation, the message window opens not automatically
Info	output of an info message, the message window opens automatically

 © FLUIDON® GmbH 2008

3 Modification of the source code of the model

The source code of a simulation model is automatically created by DSH*plus* so that, in general, the user does not have to carry out any modifications in the model source code. Under certain circumstances, a lot of time can be saved, however, if corrections are directly made in the model source code when developing a new component or when optimizing a model.

The following two sections describe the best way to proceed and the rules which have to be observed.

3.1 How to proceed when developing a component

If modifications in the source code of a component are not made in the file of the user-defined components but in the source code of a test model the model source code can be compiled after modification and the behaviour of the component can be directly tested in the test model. This means that the process of model creation is not to be carried out after every modification. It is not until the component shows the correct behaviour that the correct component description is copied into the *User3.dat* file.

If you want to carry out modifications direct in the source code of a model for a more simplified component development proceed as follows:

1. Call **EDITSOURCETEXT** in the **MODEL** menu.

   Or:

2. Open the source code of the model (*dshnn__1.cpp*) in which you want to adapt a component, in a text editor or in the Integrated Development Environment of Borland. The project number (*nn*) can be inquired in the dialog **INFO ON CURRENT PROJECT** which can be called by means of the menu point **INFO** in the **PROJECT** menu.

3. Carry out adaptation of the component in the model source code. Please strictly adhere to the programming rules (see chapter 6, 3.2).

4. Save the modifications and change from the editor to DSH*plus*.

5. Compile the model again and then correct in the source code all the error messages shown in the compiler window.

   Attention:

   - You are not allowed here to use the command **CREATE** in the dialog **STATUS MODEL BUILD UP** as otherwise all the source code modifications you made will be overwritten.

6. If the compiler window shows any error messages or if your model does not yet show the behaviour you desired in the test simulations carried out after compilation correct the source code of the component in your model file.

   Save the model file after modifying the modifications.

7. Repeat steps five to seven as long as all error messages are removed and the model shows the behaviour you desired in the test simulations.

8. In order to take over the new or modified component into your user-defined library, copy the component source code from the model file into the *AnyName.dat* file via the intermediate memory and delete the old source code of the new component.

9. If you want to give the modified component a new name, change the category name of the component copied into the *AnyName.dat* file and then carry out steps five to ten of chapter 6, 1.2, if necessary.

 © FLUIDON® GmbH 2008

Note:

- This method is only suitable for smaller modifications in the source code of the new component. If there are larger modifications it is recommendable to proceed as described in chapter 6, 1.2.

- If you have included new parameters in a component of the model they have to be inserted into the parameter list of the component by means of the function **EDIT PARAMETERS** prior to a simulation run. To do this, the parameter is entered into the list with a dot between the component name and the parameter designation including the standard value and the unit.

3.2 Programming rules

When carrying out any modifications in the source code of a model, please observe the following programming rules:

- If you want to include new ***Parameters*** or ***Auxiliary parameters*** into your component, they have to be listed in the Parameter definition or Auxiliary parameter definition of the component description. Furthermore, new parameters have to be included into the parameter list in section *Initialization* at the end of the model file. In initialization, allocation of the parameter values is effected by means of the function *LoadParameters (ParameterName)*. The component name is placed before the parameter name and separated by a dot.

```
ComponentName.Parameter = LoadParameter (dat,
                                "ComponentName.Parameter");
```

- If you want to include additional variables into your new component, they have to be included into the variable definition of the component description. Furthermore, new variables have to be listed in the value allocation of the differential equation system under *Differential equations* in the model file.

- As soon as you include variables the value range of which is limited, these variables have to be mentioned with their respective up-

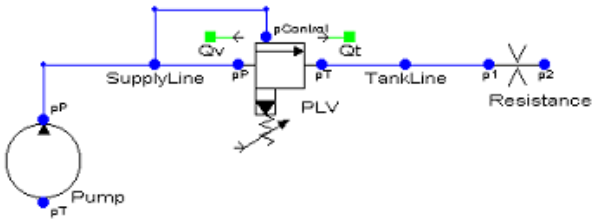
per and lower limits in the *Limit values* section in the model file. Limit values can also be transferred as parameters here.

- If variables are included into the code which are limited by limiting functions (variable name *limitation*), their value limitations have to be constantly updated. This is done by transferring the value to the variable which is to be limited in the *Limitations* section of the model file.

- If the number of variables has changed in the model, their number has to be corrected in the *File* section of the model file. In general, the *Total number of variables* and the *Total number of integrated variables* have to change by the same value. This number of variables is extremely important as it is used for dimensioning the fields for the variable pointers, which inevitably leads to a crash of the simulation if the number is too small.

3.3 Programming example

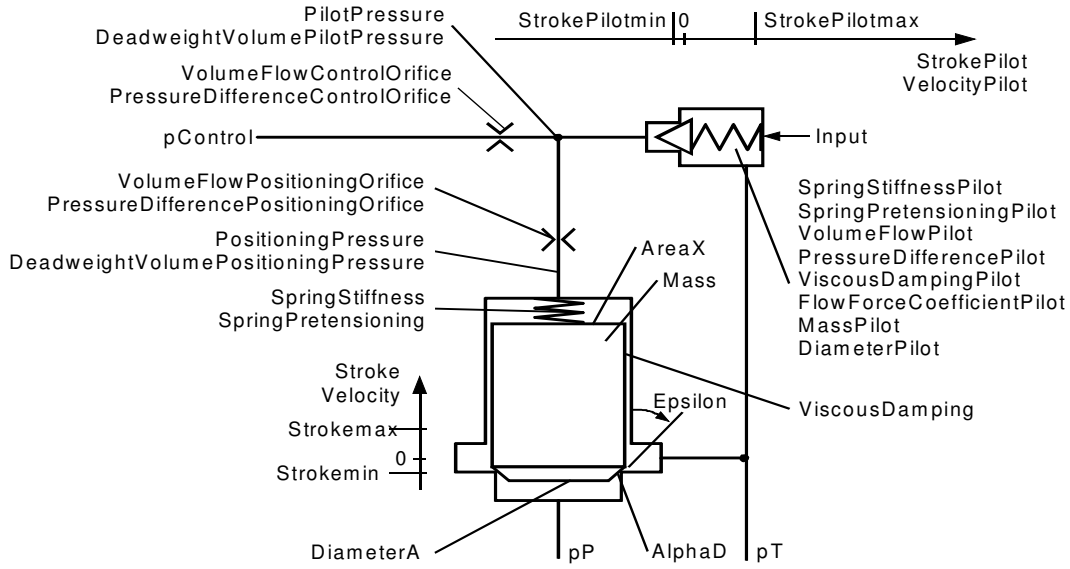
The example of a pressure relief valve as component of a simple model is used to show how a component in a model can be adapted to the specific requirements without creating a new model.



Model structure

The model is a pressure relief valve with pilot stage which is supplied by a constant pump and to which a resistor in the form of an orifice is connected downstream at the side of the reservoir.

 © FLUIDON® GmbH 2008



Structure of the pressure relief valve

The pressure relief valve corresponds to the representation shown in the illustration.

The pump pressure at the P connection acts on the main control piston. At the same time, the spring-loaded side of the main control piston is pressurized with the pressure at the control connection (here the pump pressure). This means that it is almost hydraulically compensated and is only kept closed by means of the force of the main piston spring. Furthermore, the pressure is applied at the pilot piston. As soon as the pressure exceeds the value adjusted at the pilot fields, the pilot cone lifts up from its seat and control oil can flow off to the reservoir. A pressure drop occurs at the main control piston because of the nozzle adjustment which leads the piston to lift off its seat and release connection from the pump to the reservoir.

Problem:

- The reservoir connection of the pilot is to be damped by a nozzle (orifice) from the side of the reservoir. In order to do this, the source code is to be enlarged by the functioning of this nozzle.

How to proceed:

Calculation of the pressure which appears in the pilot is effected by means of a volume flow balance of the volume flows which enter or

leave the idle volume by means of the pilot piston and the reservoir nozzle:

$$p = \frac{E'_{Öl}}{V_{idle}} \int (Q_{in} - Q_{out}) dt \; .$$

1. Call the DSH*plus* editor (**EDIT MODEL SOURCE CODE**). The source code of the model appears which is introduced by a commenting model header. Look for the source code of the pressure relief valve.

```
// Title:    Programming example
// Author:   FLUIDON
// Info:     Example model to describe the change of
//           component source code in a simulation model.
// created:  28.05.2002  15:25

#include "dgltool.h"

/*-------------------------------------------------*/
/*  Component description                          */
/*-------------------------------------------------*/
```

   The source code of the pressure relief valve is commented by a short component description. It is followed by the declarations of the parameters, the auxiliary parameters and the variables. In order to describe the newly added nozzle in the form of an orifice, the parameter list is completed by the characterizing parameters (**volume flow** and **pressure drop**).

   In order to calculate the pressure state in the pilot, a volume flow balance is to be made for the new volume. Furthermore, the volume included in the pilot has to be indicated as **idle volume** for calculating the pressure build-up. The new parameters can be put in at any position into the parameter list.

 © FLUIDON® GmbH 2008

```
class PLVpiloted {
/*-------------------------------------------------------*/
/*   Piloted pressure relief valve                       */
/*                                                       */
/*              Orifice1                                 */
/*    Control---X------0-------Pilotvalve                */
/*                     | Pilotpressure  |                */
/*                     |                |  PressureZ     */
/*                     X Orifice2       |                */
/*                     |                X  new Orifice   */
/*                     | PositioningPres|sure            */
/*                  ____/____           |  pT            */
/*                   Mainvalve-----------0----Reservoir  */

/*                     |                                 */

/*   ControlInlet -----0                                 */
/*                     |                                 */
/                      pP                                */
/*                                                       */
/*   (pP, pT, Stroke, Velocity)                          */
/*-------------------------------------------------------*/
public:

// Parameter
  LDOUBLE AlphaD;
  LDOUBLE Epsilon;
  LDOUBLE Strokemax;
  LDOUBLE Strokemin;
  LDOUBLE SpringStiffness;
  LDOUBLE SpringPretensioning;
  LDOUBLE ViscousDamping;
  LDOUBLE Mass;
  LDOUBLE DiameterA;


  LDOUBLE AaToAx;
  LDOUBLE VolumeFlowControlOrifice;

  LDOUBLE PressureDifferenceControlOrifice;
  LDOUBLE VolumeFlowPositioningOrifice;
  LDOUBLE PressureDifferencePositioningOrifice;
  LDOUBLE DeadweightVolumePositioningPressure;
  LDOUBLE DeadweightVolumePilotPressure;
  LDOUBLE VolumeFlowPilotReservoir;        | Parameter
of
  LDOUBLE PressureDifferencePilotReservoir;| new Orifice
  LDOUBLE DeadweightVolumePilotReservoir;  | Idle volume
                                           | Pilot and
                                           | Orifice
  LDOUBLE VolumeFlowPilot;
  LDOUBLE PressureDifferencePilot;
  LDOUBLE StrokePilotmax;
  LDOUBLE StrokePilotmin;
  LDOUBLE SpringStiffnessPilot;
  LDOUBLE SpringPretensioningPilot;
  LDOUBLE ViscousDampingPilot;
  LDOUBLE FlowForceCoefficientPilot;
  LDOUBLE MassPilot;
  LDOUBLE DiameterPilot;
  LDOUBLE Rho;
  LDOUBLE EOil;
```

A further auxiliary parameter is included for calculating the orifice coefficient only once from the parameters.

```
// AuxiliaryParameter
  LDOUBLE AreaX;
  LDOUBLE AreaA;
  LDOUBLE AreaB;
  LDOUBLE AreaPilot;
  LDOUBLE Coefficient;

  LDOUBLE CoefficientPilot;

  LDOUBLE Acceleration;
  LDOUBLE AccelerationPilot;
  LDOUBLE QMain;
  LDOUBLE Orifice1Coefficient;
  LDOUBLE Orifice2Coefficient;
  LDOUBLE VarVolume;
  LDOUBLE FlowForceCoefficient;
  LDOUBLE QReservoir;
  /*Orifice Coefficient */
  LDOUBLE OrificePilotReservoirCoefficient;
```

For calculation of the pilot pressure which builds up in the pilot, an additional variable is included in the variable list. No signal outputs such as the volume flow output of a component have to be mentioned in the variable list.

Calculation of the orifice coefficient is included in the declaration of the **Auxiliary functions**. The term ***CalcCoefficient (...,...)*** calls a library function for calculating the orifice coefficient. As an alternative, the formula of calculation can directly be put in.

```
// Variablen
  LDOUBLE *pP;
  LDOUBLE *pT;
  LDOUBLE *pControl;
  LDOUBLE *Stroke;
  LDOUBLE *StrokePilot;
  LDOUBLE *Velocity;
  LDOUBLE *VelocityPilot;
  LDOUBLE *PositioningPressure;
  LDOUBLE *PilotPressure;
  LDOUBLE *Input;
  /* Pressure between Pilot and new orifice */
  LDOUBLE *PressureZ;

// AuxiliaryFunction
  void AuxiliaryFunction(SIMSTRUCT *S)
  {
```


© FLUIDON GmbH 2008

```
                        * SQRT (2 / Rho);
```

```
    CoefficientPilot = CalcAdjustCoefficient
                      (VolumeFlowPilot,
                       PressureDifferencePilot,
                       StrokePilotmax - StrokePilotmin);

    OrificePilotReservoirCoefficient =
       CalcCoefficient (VolumeFlowPilotReservoir,
                        PressurDifferencePilotReser-
voir);

    Orifice1Coefficient =

       CalcCoefficient (VolumeFlowControlOrifice,

                        PressureDifferenceControlOrifice);
    Orifice2Coefficient =
       CalcCoefficient (VolumeFlowPositioningOrifice,
                   PressureDifferencePositioningOrifice);

    AreaA      = M_PI_4 * sqr (DiameterA);
    AreaPilot  = M_PI_4 * sqr (DiameterPilot);

    if (AaToAx)
       AreaX = AreaA / AaToAx;
    else
       AreaX = AreaA;

    AreaB      = AreaX - AreaA;
    QMain      = 0;
    QReservoir    = 0;

    if (Mass <= 0)
       Mass = 10;
    if (MassPilot <= 0)
       MassPilot = 10;

    FlowForceCoefficient =   Rho * COS (Epsilon)
                           / (DiameterA * M_PI);
  }
// ExitFunction
```

Calculation of the variables is effected in the **Functions declaration part**. The variables include for example the pressures which build up and which are calculated from the integration of the volume flows.

If this is a hydraulic component connection the volume flow flowing through this connection is calculated. These volume flows are balanced in the volume nodes and the pressure in the volume nodes is

calculated from there (These calculations are effected in the differential equation part at the end of the file).

The calculated pressures on their part are applied on the closed component connections. Function for the calculation of the volume flows is started by a comment line which contains the variable to which the pressure is to be allocated, in the form of a keyword. The pressures **pP**, **pT** and **pControl** are hydraulic component connections which have always to be started with a small **p**. The pressures **PilotPressure** and **PositioningPressure** are pressures which build up in internal idle volumes. As their pressure calculation cannot be effected in volume nodes, the pressure gradients are calculated from the entering and leaving volume flows, integrated into the background and transferred to the variable in the comment line.

If the reservoir nozzle is taken into account the volume flow is changed because of the new pressure distribution at the reservoir connection **pT**.

Important:

The section of the formula printed in ***italics*** is therefore replaced by the section printed in **bold** type.

```
// Functions
// pP /* State variable as keyword in the comment line */
/* function declaration    */
  LDOUBLE VolumeFlowP(SIMSTRUCT *S)
  {                          /* of the flow calculation */
     return (-(QMain = (VarFlow (Coefficient, *pP - *pT,
                                *Stroke, Strokemin))));
  }

// pT
  LDOUBLE VolumeFlowT(SIMSTRUCT *S)
  {
    return (  VarFlow (Coefficient, *pP - *pT, *Stroke,
                       Strokemin)
/*  obsolete source code                                  */
/*   + VarFlow (CoefficientPilot, *PilotPressure - *pT,*/
/*              *StrokePilot, StrokePilotmin));        */
/*  new source code
*/
            + OrificePilotReservoirCoefficient
              * psqrt (*PressureZ - *pT));
  }

// pControl
  LDOUBLE VolumeFlowControl(SIMSTRUCT *S)
```


© FLUIDON® GmbH 2008

```
    return (-(Orifice1Coefficient * psqrt (*pControl -
                                        *PilotPressure)));
  }
```

- The nozzle in the reservoir connection of the pilot is taken into account by calculating the volume flow and the pressure gradient ***PressureZDot*** arising from it. To do this, a new function is introduced which enables calculation of the pressure built-up in the new volume between the pilot and the nozzle.

```
// PressureZ
  LDOUBLE PressureZDot(SIMSTRUCT *S)
  {
     return ((OrificePilotReservoirCoefficient
             * psqrt (*pT - *PressureZ)
             + VarFlow (CoefficientPilot,
                        *PilotPressure - *PressureZ,
                        *StokePilot, StokePilotmin))
        * EOilCorrection (EOil, PressureZ)
        / DeadweightVolumePilotReservoir);
}
```

The calculation equation of the **PilotPressure** has also to be adapted to the new pressure distribution as the pilot is no longer directly pressurized with the reservoir pressure **pT**. To do this, the pressure **pT** has to be replaced by the **PressureZ**.

```
// PilotPressure
  LDOUBLE PilotPressureDot(SIMSTRUCT *S)
  {
/*  QReservoir = VarFlow (CoefficientPilot,            */
/*                        *PilotPressure - *pT,        */
    QReservoir = VarFlow (CoefficientPilot,
                          *PilotPressure - *PressureZ,
                         *StokePilot, StokePilotmin);

    return ((  Orifice1Coefficient
               * psqrt (*pControl - *PilotPressure)
             - Orifice2Coefficient
          * psqrt (*PilotPressure - *PositioningPressure)
             - QReservoir)
           * EOilCorrection (EOil, PilotPressure)
           / DeadweightVolumePilotPressure);}
```

Calculation of the main piston acceleration which is integrated into the central integration to the main piston speed and is allocated to the variable velocity, is effected in the function ***velocityDot()***.

```
// Velocity
  LDOUBLE VelocityDot(SIMSTRUCT *S)
```

```
  {
     LDOUBLE Flowforce;

     if (*Stroke - Strokemin)
        Flowforce = FlowForceCoefficient * sqr (QMain)
                  / (*Stroke - Strokemin);
     else
        Flowforce = 0;

     Acceleration = (- *PositioningPressure * AreaX
                     + *pT * AreaB
                     + *pP * AreaA
                     - ViscousDamping * *Velocity
                     - Flowforce
                     - SpringStiffness
                      * (*Stroke + SpringPretensioning))
                     / Mass;

     if (((*Stroke>=Strokemax) && (Acceleration>=0.)) ||
         ((*Stroke<=Strokemin) && (Acceleration<=0.)))
        return (0.);
     else
        return (Acceleration);
  }
```

Movement of the piston is limited to a defined range. As soon as the piston reaches the limit stop, speed is limited, i.e. it is set to zero. Because of the keyword **limitation,** the value returned by the function is not integrated so that the speed value can directly be returned to the speed variable. The piston travel is integrated from the speed.

```
// Velocity
  LDOUBLE VelocityLimit(SIMSTRUCT *S)
  {
     if (((*Stroke >= Strokemax) && (*Velocity >= 0.)) ||
         ((*Stroke <= Strokemin) && (*Velocity <= 0.)))
        return (0.);
    else
        return (*Velocity);
```


© FLUIDON GmbH 2008

```
// Stroke
  LDOUBLE StrokeDeriv(SIMSTRUCT *S)
  {
     return (*Velocity);
  }
```

The velocity of the pilot piston is calculated in the same way. The flow force coefficient and the pressure force, however, have to be adapted to the pilot piston.

```
// VelocityPilot
LDOUBLE VelocityPilotDot (SIMSTRUCT *S)
{
   AccelerationPilot = ( *PilotPressure * AreaPilot
                        - *pT * AreaPilot
                        - ViscousDampingPilot
                          * *VelocityPilot
                        - FlowForceCoefficientPilot
                          * FABS (*pP - *pT)
                          * (  *StrokePilot
                             - StrokePilotmin)
                        - SpringStiffnessPilot
                          * (  *StrokePilot
                             + SpringPretensioningPilot
                             + *Input))
                        / MassPilot;

  if (((*StrokePilot >= StrokePilotmax)
        && (AccelerationPilot >= 0.)) ||
      ((*StrokePilot <= StrokePilotmin)
        && (AccelerationPilot <= 0.)))
     return (0.);
  else
     return (AccelerationPilot);
}

// VelocityPilot
LDOUBLE VelocityPilotLimit (SIMSTRUCT *S)
{
   if (((*StrokePilot >= StrokePilotmax)
        && (*VelocityPilot >= 0.)) ||
       ((*StrokePilot <= StrokePilotmin)
        && (*VelocityPilot <= 0.)))
      return (0.);
    else
      return (*VelocityPilot);
  }
// StrokePilot
  LDOUBLE StrokePilotDeriv(SIMSTRUCT *S)
  {
     return (*VelocityPilot);
  }
```

The keyword **algebra** prevents integration of the return value of the function called.

```
// VolumeFlow algebra
  LDOUBLE VolFlow (SIMSTRUCT *S)

  {

     return (QMain + QReservoir);
  }
// VolumeFlowPilot algebra
  LDOUBLE VolFlowPilot (SIMSTRUCT *S)
  {
     return (QReservoir);
  };
```

The other component categories of the model are not concerned by the modification of the DBV and are therefore not listed.

The **model creation area** begins here, i.e. these source code sections are automatically created by DSH*plus* when a model is created in the **CREATE MODEL** menu. In the component definition, the component names selected by the user are allocated to the component types (*class type name*). As no new component has been inserted, no modifications are to be made here.

```
/*-------------------------------------------------*/
/*  Bauteildefinitionen / component definition     */
/*-------------------------------------------------*/

HydroNode                              ReservoirLine;
HydroNode                              SupplyLine;
PLVpiloted                             PLV;
Orifice                                Resistance;
Pump                                   Pump;
LDOUBLE *null;
LDOUBLE *TNorm;
LDOUBLE *EinBarAbs;;
static LDOUBLE *null;
```

Allocation of the component parameters entered in the simulation mode is effected in initialization. New parameters have therefore to be entered in this list starting with the name of the component.

 © FLUIDON® GmbH 2008

```
/*-----------------------------------------------------*/
/*  Initialisierung / Initialization                   */
/*-----------------------------------------------------*/

void Inititialization (SIMSTRUCT *S)
if (!null)
   {
      null  = (LDOUBLE *) malloc (sizeof (LDOUBLE));
      *null = 0;
   }
```

```
   if (!TNorm)
```

```
   {
      TNorm  = (LDOUBLE *) malloc (sizeof (LDOUBLE));
      *TNorm = Temp0;
   }
   if (!EinBarAbs)
   {
      EinBarAbs  = (LDOUBLE *) malloc (sizeof (LDOUBLE));
      *EinBarAbs = 1E5;
   }
```

```
...
   PLV.AlphaD      = LoadParameter(dat, "PLV.AlphaD");
   PLV.Epsilon     = LoadParameter(dat, "PLV.Epsilon");
   PLV.Strokemax   = LoadParameter(dat, "PLV.Strokemax");
   PLV.Strokemin   = LoadParameter(dat, "PLV.Strokemin");
   PLV.SpringStiffness     = LoadParameter(dat,
                    "PLV.SpringStiffness");
   PLV.SpringPretensioning = LoadParameter(dat,
                    "PLV.SpringPretensioning");
   PLV.ViscousDamping      = LoadParameter(dat,
                    "PLV.ViscousDamping");
   PLV.Mass                = LoadParameter(dat,
                    "PLV.Mass");
   PLV.DiameterA           = LoadParameter(dat,
                    "PLV.DiameterA");
   PLV.AaToAx              = LoadParameter(dat,
                    "PLV.AaToAx");

   PLV.VolumeFlowControlOrifice =  LoadParameter(dat,
                    "PLV.VolumeFlowControlOrifice");
   PLV.PressureDifferenceControlOrifice
                           = LoadParameter(dat,
                "PLV.PressureDifferenceControlOrifice");
   PLV.VolumeFlowPositioningOrifice = LoadParameter(dat,
                "PLV.VolumeFlowPositioningOrifice");
   PLV.PressureDifferencePositioningOrifice
                           = LoadParameter(dat,
           "PLV.PressureDifferencePositioningOrifice");
   PLV.DeadweightVolumePositioningPressure
                           = LoadParameter(dat,
           "PLV.DeadweightVolumePositioningPressure");
   PLV.DeadweightVolumePilotPressure = LoadParameter(dat,
           "PLV.DeadweightVolumePilotPressure");
   PLV.DeadweightVolumePilotReservoir
                            = LoadParameter(dat,
```

```
    PLV.VolumeFlowPilotReservoir = LoadParameter(dat,
              "PLV.VolumeFlowPilotReservoir ");
    PLV.PressurDifferencePilotReservoir
                          = LoadParameter(dat,
              "PLV.PressurDifferencePilotReservoir ");
    PLV.VolumeFlowPilot      = LoadParameter(dat,
              "PLV.VolumeFlowPilot");
    PLV.PressureDifferencePilot     = LoadParameter(dat,
              "PLV.PressureDifferencePilot");
    PLV.StrokePilotmax        = LoadParameter(dat,
              "PLV.StrokePilotmax");
    PLV.StrokePilotmin        = LoadParameter(dat,

              "PLV.StrokePilotmin");

    PLV.SpringStiffnessPilot  = LoadParameter(dat,
              "PLV.SpringStiffnessPilot");
    PLV.SpringPretensioningPilot = LoadParameter(dat,
              "PLV.SpringPretensioningPilot");
    PLV.ViscousDampingPilot    = LoadParameter(dat,
              "PLV.ViscousDampingPilot");
    PLV.FlowForceCoefficientPilot = LoadParameter(dat,
              "PLV.FlowForceCoefficientPilot");
    PLV.MassPilot             = LoadParameter(dat,
              "PLV.MassPilot");
    PLV.DiameterPilot         = LoadParameter(dat,
              "PLV.DiameterPilot");



    PLV.Rho                   = LoadParameter(dat,
              "PLV.Rho");
    PLV.EOil                  = LoadParameter(dat,
              "PLV.EOil");
```

Next, the **Auxiliary functions** are called during initialization. If auxiliary functions are added to a component which has not had any auxiliary functions up to now the call **AuxiliaryFunction ()** is to be entered here starting with the name of the component. The name of the component can be taken from the component declaration.

```
    ReservoirLine.AuxiliaryFunction(S);
    SupplyLine.AuxiliaryFunction(S);
    PLV.AuxiliaryFunction(S);
    Resistance.AuxiliaryFunction(S);
    Pump.AuxiliaryFunction(S);
}
```

 © FLUIDON® GmbH 2008

If new dynamic memory fields have been created when a component has been modified, these fields have to be released again at the end of simulation by calling the **exit functions**. To do this, the exit functions have to be entered here, starting with the component name.

```
/*-----------------------------------------------------*/
/*  ExitFunction                                       */
/*-----------------------------------------------------*/

void ExitFunction (SIMSTRUCT *S)
{


}
```

A lot of variables only exist in defined limits. These limit values are allocated to the variables in the following list. Allocation is effected to a pointer in which the variables have a defined position. This position is given to the variables in the declaration of the differential equations. **yLimit[0].Min = -1E5** is a *Reservoir.Pressure* which is limited below with the vacuum point (internal calculation is not made with the absolute pressure, but with the relative pressure. This means that the limit value is -1 bar).

The variable **PressureZ** has been included with the extension of PLV. This value is a pressure which has to be limited below.

```
/*-----------------------------------------------------*/
/*  Grenzwerte / Limits                                */
/*-----------------------------------------------------*/

void Limits (GRENZWERT *yGrenz, SIMSTRUCT *S)
{
   yGrenz[0].Min = -1E5;
   yGrenz[0].Max = 1.E99;
   yGrenz[1].Min = -1E5;
   yGrenz[1].Max = 1.E99;
   yGrenz[2].Min = PLV.Strokemin;
   yGrenz[2].Max = PLV.Strokemax;
   yGrenz[3].Min = PLV.StrokePilotmin;
   yGrenz[3].Max = PLV.StrokePilotmax;
   yGrenz[4].Min = -1.E99;
   yGrenz[4].Max =  1.E99;
   yGrenz[5].Min = -1.E99;
   yGrenz[5].Max =  1.E99;
```

```
   yGrenz[6].Max =  1.E99;
   yGrenz[7].Min = -1.E99;
   yGrenz[7].Max =  1.E99;
   yGrenz[8].Min = -1.E99;
   yGrenz[8].Max =  1.E99;
}
```

Pointer position

In the **differential equation part**, allocation of the values is effected from the central integration program to the variables. ***LDOUBLE t*** represents the integration time, ***LDOUBLE h*** represents the integration incrementations, ***LDOUBLE *y*** is the pointer of the variables and ***LDOUBLE *f*** comprises the derivation values of the variables.

New variable

A pointer has to be allocated to the newly included variable **PressureZ**. This means that all subsequent pointers are displaced by one value.

```
/*-----------------------------------------------------*/
/*  Differentialgleichungen / Differential equations */
/*-----------------------------------------------------*/

void Dgl (LDOUBLE t, LDOUBLE *y, LDOUBLE *f, LDOUBLE h,
SIMSTRUCT *S)
{
/* The declaration list of the state variables has to  */
/* be extended by the new variable. The variables is   */
/* assigned a pointer. The list starts with the        */
/* hydronodes followed be the other variables.         */
```

```
// Initialisierung der Knoten
// Initialization of nodes
   ReservoirLine.Pressure     = y;          //0
   SupplyLine.Pressure        = ++y;        //1
/* here are the other variables                 */
// Initialisierung integrierte Variablen
// Initialisation of integral variables
// Zuweisungen Knoten-Element
/// assignment node-element
   PLV.pP                     = SupplyLine.Pressure;
   PLV.pT                     = ReservoirLine.Pressure;
   PLV.pControl               = SupplyLine.Pressure;
   PLV.Stroke                 = ++y;        //2
   PLV.StrokePilot            = ++y;        //3
   PLV.Velocity               = ++y;        //4
   PLV.VelocityPilot          = ++y;        //5
   PLV.PositioningPressure    = ++y;        //6
   PLV.PilotPressure          = ++y;        //7
   PLV.PressureZ               = ++y;        //8
   PLV.Input                  = null;
   Resistance.p1              = ReservoirLine.Pressure;
   Resistance.p2              = null;
   Pump.pP                    = SupplyLine.Pressure;
```


© FLUIDON GmbH 2008

```
// Initialisierung algebraische Knoten
// Initialisation of algebraic nodes
   Qt.Value                    = ++y;        //9
   Qv.Value                    = ++y;        //10

// Initialisierung algebraische Variablen
// Initialisation of algebraic variables

// Ausgabe algebraische Berechnungen
// Output of algebraic calculations
   *Qt.Value   = PLV.VolFlow(S);
   *Qv.Value   = PLV.VolFlowPilot(S);

// Ausgabe fluidische Gleichungen
```

```
// Output of fluid equations
```

```
   ReservoirLine.PressureDeriv = ReservoirLine.EOil(S)
               / ReservoirLine.Volume
               *( Resistance.VolumeFlow1 (S)
                + PLV.VolumeFlowT(S));

   SupplyLine.PressureDeriv = SupplyLine.EOil(S)
               / SupplyLine.Volume
               *( PLV.VolumeFlowControl(S)
                + PLV.VolumeFlowP(S)
                + Pump.VolumeFlowP(S));
```

Increase pointer position by one

The derivation functions are allocated to the derivation vector. A derivation function has been included for calculating the new variable **PressureZ**. This derivation function has to be entered into the list for the same field position (**here 8**).

```
// Ausgabe der Ableitungen / Output of derivations
   f[0] = ReservoirLine.Pdot(S);
   f[1] = SupplyLine.Pdot(S);
   f[2] = PLV.StrokeDeriv(S);
   f[3] = PLV.StrokePilotDeriv(S);
   f[4] = PLV.VelocityDeriv(S);
   f[5] = PLV.VelocityPilotDeriv(S);
   f[6] = PLV.PositioningPressureDeriv(S);
   f[7] = PLV.PilotPressureDeriv(S);
   f[8] = PLV.PressureZDeriv(S);
}
```

The value limitations of the variables to be integrated have to be updated after every integration. These updates are carried out here. The value limitation for **PressureZ** has also to be entered here.

```
/*-------------------------------------------------*/
/*  Begrenzungen / Limitations                     */
/*-------------------------------------------------*/

void __pascal Begrenzungen (LDOUBLE *y)
{
   PLV.Stroke                = &y[2];
   PLV.StrokePilot           = &y[3];
   PLV.Velocity              = &y[4];
   PLV.VelocityPilot         = &y[5];
   PLV.PositioningPressure   = &y[7];
   PLV.PilotPressure         = &y[8];
   PLV.PressureZ              = &y[8];
   PLV.Input                 = null;

   y[4] = PLV.VelocityLimit(S);


   y[5] = PLV.VelocityPilotLimit(S);


}
```

Adapt number of variables

At this point, dimensioning of the fields of the variables is carried out. By enlarging the DBV by one reservoir nozzle, the total number of variables and the total number of variables to be integrated has increased by one each. This means that the total number of variables increases from 10 to 11 and the number of the variables to be integrated from 8 to 9.

```
/*-------------------------------------------------*/
/*  Daten / Data                                   */
/*-------------------------------------------------*/

VARDATA* Daten()
{
   VARDATA *Pointer;
   Pointer = (VARDATA *) malloc (sizeof (VARDATA));
// number of HydroNodes
   Pointer->HydroKnotenCounter = 2;
// total number of variables
   Pointer->YCounter           = 11;
// total number of integrated variables
   Pointer->IntKnotenCounter   = 9;
   return (Pointer);
}
```

 © FLUIDON® GmbH 2008

2. **Save** the modifications after termination of programming and change from the editor to DSH*plus*.

3. **Compile** the model again and remove all error messages in the source code listed in the compiler window.

   Attention:

   - You are not allowed to use here the command **CREATE MODEL** as otherwise all the source code modifications you made are overwritten.

4. If you have included new parameters into a component of the model they have to be included into the parameter list of the component by means of the function **EDIT PARAMETERS**. The parameter is entered into the list with a dot between the component name and the parameter designation including the standard value and the unit.

5. Before you start working with the modified model, it is recommendable to check its physical correctness.

4 Integration of external functions

DSH*plus* offers the possibility to integrate external functions into the source code of the components. This can be useful for example if you have a finished digital computer which has been developed for a system designed in hardware and you want to examine this controller together with the system in the simulation (Software-in-the-Loop).

These functions can be available as object file, as static or dynamic library (*lib*- or *dll*-File) or as C++ source code with their header files. If the function is available as *dl* file, another Import library is necessary which can be generated with the command line tool *implib.exe* for example.

External functions can be integrated into the component BLACKBOX specifically provided to this end or in any other component. The files in which the functions are stored have only to be entered under **Compiler and Linker** in the **OPTIONS** dialog.

Static libraries and import libraries of the DLLs are entered under **Additional Libs,** Header files under **Additional Includes**, and object files under **Additional Objects**. If the function is available as inline-defined header file, it has only to be mentioned under **Additional Includes** for integration. The paths to the files are entered under **Include files** and **Libs** if they deviate from the paths already entered.

The external functions integrated in this way can then be called in the source code of the components.

In the following, the procedure is explained with a very easy example.

The external function is defined in the file *Testfunc.cpp* and looks as follows.

```
#include "testfunc.h"

long double Testfunction (long double Input)
{
   return (Input * 2.5);
}
```

The prototype defined in the header file *Testfunc.h* belongs to the definition of this function.

```
long double Testfunction (long double Input);
```

An object file and a static library, if necessary, are now created from the source code. In this case, the same compiler with which the DSH*plus* simulation model is created, is to be used. The file can be called as follows:

bcc32 +dshdgl.cfg testfunc.cpp

The static library is then created in this way:

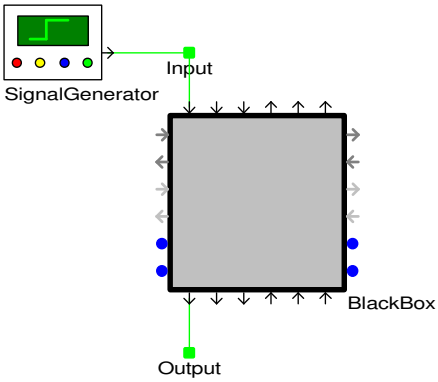
tlib testfunc.lib + testfunc.obj

 © FLUIDON® GmbH 2008

If you use a dynamic link library (**.dll*) any programming language with which a DLL for Windows can be created, can be used. For this case, other functions in the source code are necessary. For further information, please see the Compiler manual.

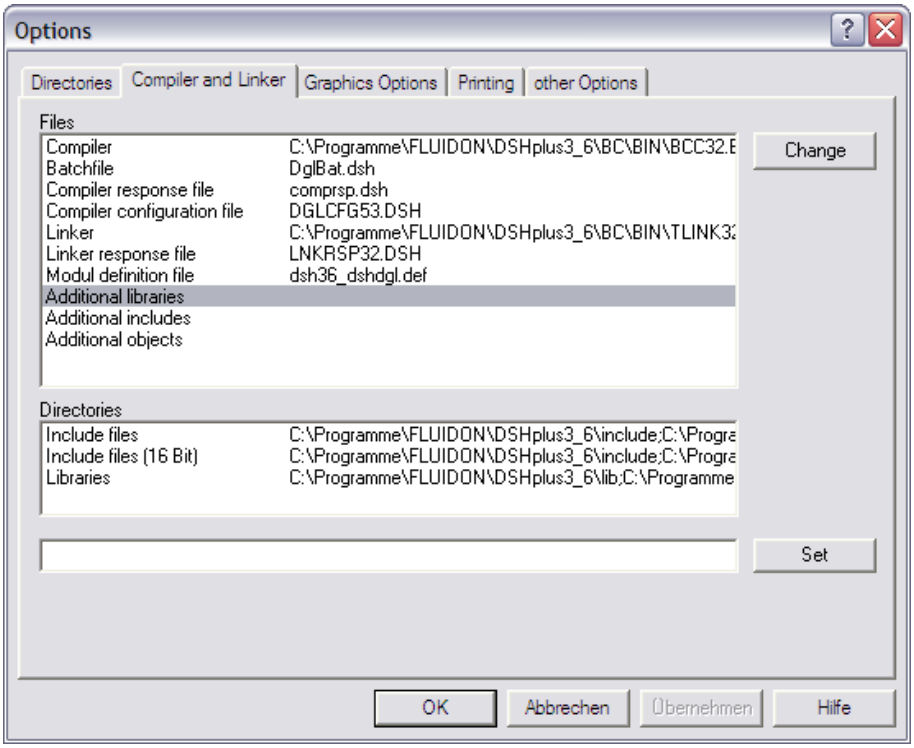
If the external function is now available as *lib-, dll-* or *obj* file or as C++ source code and you want to integrate it into the calculation of the model, proceed as follows:

1. Place the component BLACKBOX at the position of your simulation model where you want to integrate external functions.



Example model with BLACKBOX

2. The external functions are integrated into the model by entering the corresponding files under **Compiler and Linker** in the **OPTIONS** dialog.



3. Create the source code of the model by means of the program function **CREATE** in the dialog **Status model build up**.

4. It is now possible to edit the source code of the mode and to enter the calls of the external functions in the component BLACKBOX.

```
// Title:      BlackBox model
// Author:     FLUIDON
// Info:
// Datum:      02.03.2001  13:32

#include "dgl.h"
#include "dgltool.h"
#include "TESTFUNC.H"

/*--------------------------------------------------*/
/*  Bauteilbeschreibungen / component description   */
/*--------------------------------------------------*/

...
```

```
class Blackbox {
/*--------------------------------------------------*/
/*  Frame to insert an external function            */
/*  10 Parameters, 6 Inputs, 6 Outputs,             */
/*  4 hydraulical ports, each 2 trans. Inputs/Outputs, */
/*  each 2 rot. Inputs/Outputs                      */
```

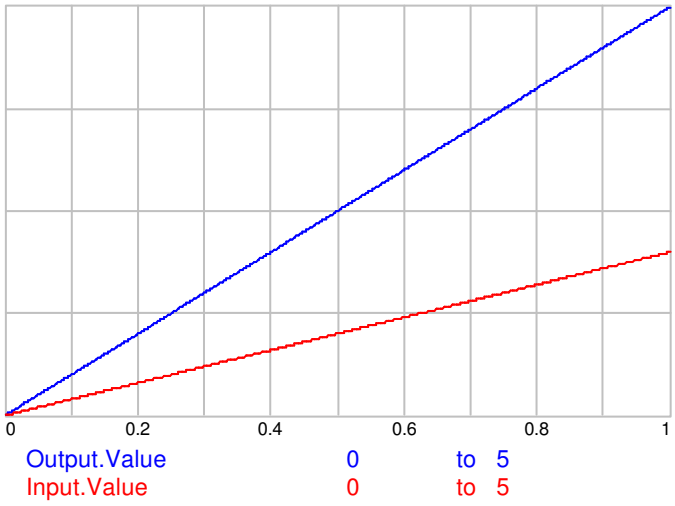
```
  public:
// Parameter
  LDOUBLE Parameter1;
  LDOUBLE Parameter2;

...

// Output4 algebra
  LDOUBLE Output4 (SIMSTRUCT *S)
  {
     return (Testfunction (*Input1));
  }
...
```

5. Then save the modifications of the model source code.

6. Simulation can be started after compilation of the model by means of the program function **COMPILE** in the dialog **Status model build up**.



Notes:

- After integration of the calls of the external functions in the model source code, you are no longer allowed to select the program function **CREATE** in the dialog **Status model build up** as otherwise the modifications made would be lost.

- External functions cannot only be called in the component BLACKBOX but in any other component. The procedure is the same.

- Both variables and parameters can be transferred to the external function.

 © FLUIDON® GmbH 2008

Appendix A Nodes

1 HydroNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
Pressure	V	bar	pressure at the node
Initial pressure	V	Bar	Initial pressure at the node

Parameters

Name	Value	Unit	Description
Volume	1	l	hydraulic fluid volume between the components
Temp. Difference	0	K	Deviation of the global temperature from the global system temperature
Bulk Modulus Calculation			Calculating options
look-up table			Calculation of the bulk-modulus with a look-up table
explicit by equation			equation based calculation
Bulk-Modulus	14000	bar	bulk-modulus of the hydraulic fluid including the surroundings at high pressure
Characteristics of Air			Correction of the bulk-modulus

 © FLUIDON® GmbH 2008

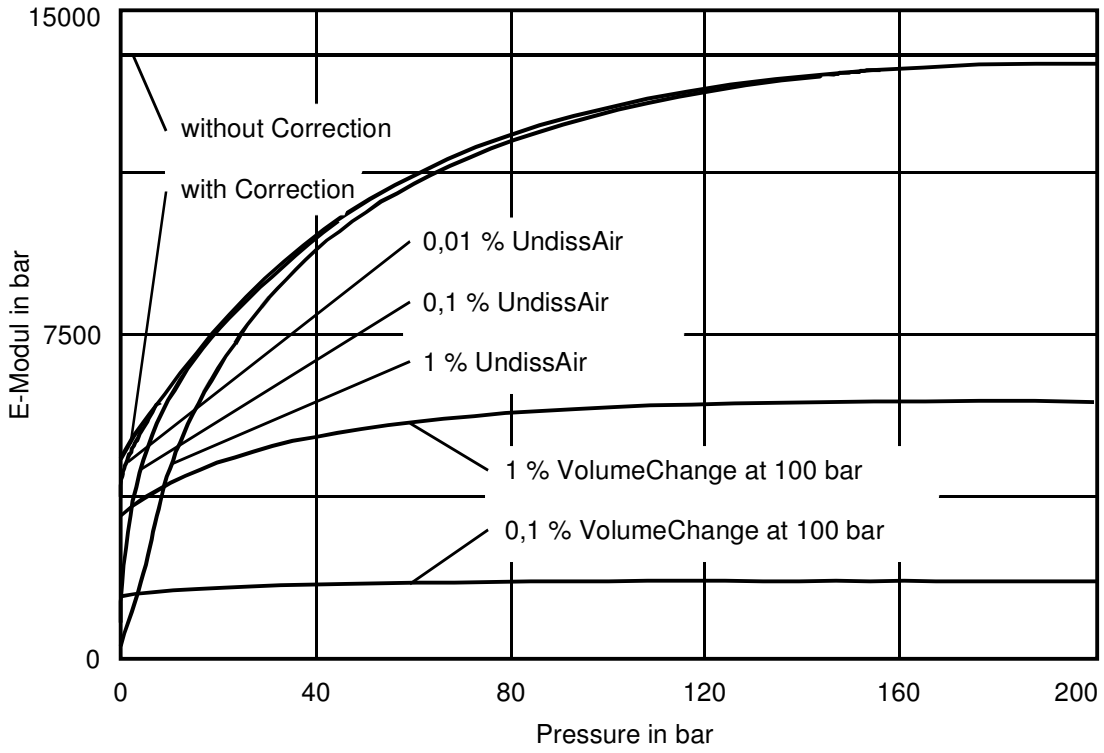
Name	Value	Unit	Description
Prop. undissolved Air	0.1	%	proportion of undissolved air
Polytropic exponent	1.1	-	polytropic exponent for the change of the air volume
Options			Characteristics of hoses
VolumeChange	0	%	volume change under *Reference* (parameter for hoses)
Reference	100	bar	Pressure, relating to *VolumeChange*
Limitations			
Max. Gradient	1E99	bar/s	gradient of maximum and minimum pressure drop
Cavitation Limit	-1	bar	pressure limit, at which cavitation occurs

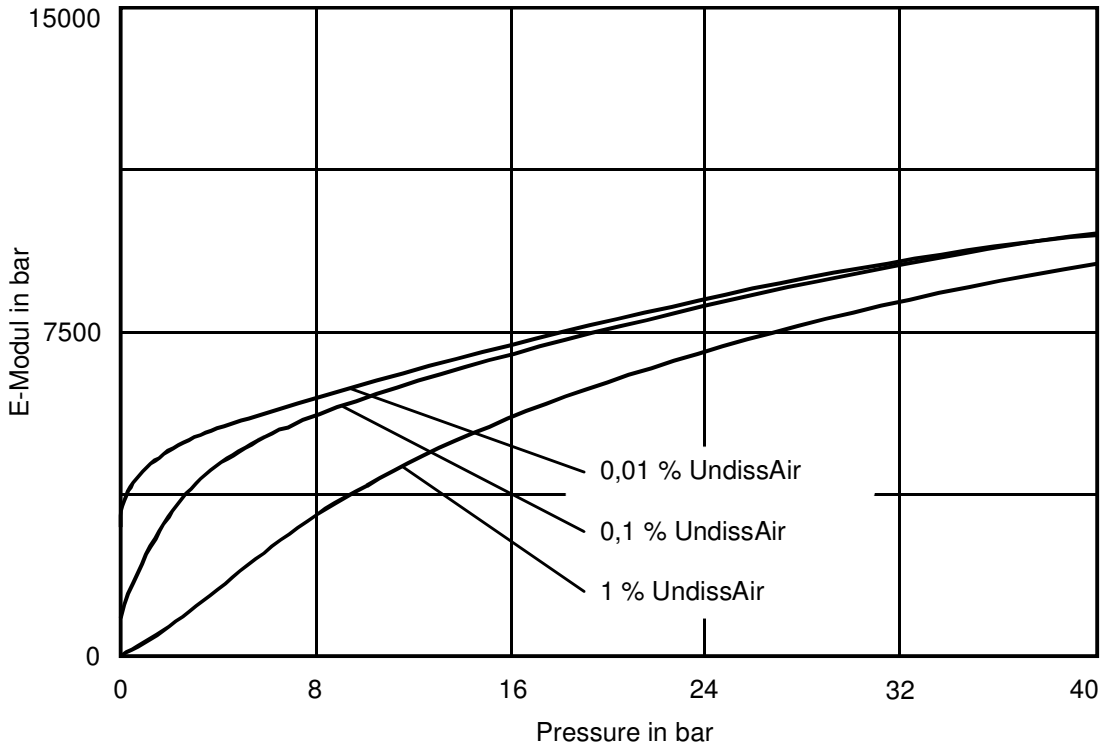
Notes

The HYDRONODE represents the volume of the hydraulics liquid between the connected components. The liquid volume consists of the volume of the lines, which connects the components, and the dead volumes. The dead volumes are those volumes, which are not able to be displaced (in e.g. valves: the volume between connection and control edge).

The parameterization of the node is done via a specific input dialog. If the *Correction* of the bulk-modulus is marked, the bulk modulus of the hydraulics liquid is corrected depending on the pressure. Otherwise, the bulk modulus remains constant.

The two following figures shows the influence of the correction of the bulk modulus with default setting (*E_Oil: 14000 bar*, *Polytropicexponent: 1,1*). The second figure shows an enlargement of the range of *0* to *40 bar* for different air proportions.





Via the parameters *PRESSUREGRADIENT* the gradient of the pressure raise and drop can be limited.

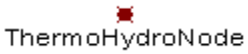
As many hydraulic connections as desired can be connected to a HYDRONODE. If a signal input is connected to a node, this connection works as a pressure sensor. If a signal output is connected to a hydraulic node, it changes into a SIGNALNODE (in the drawing area visible by

 © FLUIDON® GmbH 2008

the green colour). The pressure in the node is determined by the signal output.

HYDRONODES and the hydraulic lines between the components are represented in blue colour.

2 ThermoHydroNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
pressure	V	bar	pressure at the node
T	V	K	temperature of the volume

Parameters

Name	Value	Unit	Description
Volume	1	l	hydraulic fluid volume between the components
Temperature	293.15	K	Local temperature
Bulk Modulus Calculation			Calculating options
look-up table			Calculation of the bulk-modulus with a look-up table
explicit by equation			equation based calculation
Bulk-Modulus	14000	bar	bulk-modulus of the hydraulic fluid including the surroundings at high pressure
Characteristics of Air			Correction of the bulk-modulus
Prop. undissolved Air	0.1	%	proportion of undissolved air
Polytropic exponent	1.1	-	polytropic exponent for the change of the air volume
Options			Characteristics of hoses

 © FLUIDON® GmbH 2008

Name	Value	Unit	Description
VolumeChange	0	%	volume change under *Reference* (parameter for hoses)
Reference	100	bar	Pressure, relating to *VolumeChange*
Limitations			
Max. Gradient	1E99	bar/s	gradient of maximum and minimum pressure drop
Cavitation Limit	-1	bar	pressure limit, at which cavitation occurs

Notes

The THERMOHYDRONODE represents the volume of the hydraulics liquid between the connected components. The liquid volume consists of the volume of the lines, which connects the components, and the dead volumes. The dead volumes are those volumes, which are not able to be displaced (in e.g. valves: the volume between connection and control edge).

Beside the calculation of the pressure the temperature of the fluid is calculated in the THERMOHYDRONODE. The basis for this are heat flows that are generated or dissipated of thermo-hydraulic components.

The parameterization of the node is done via a specific input dialog. Normally the bulk-modulus depends on the fluid that is chosen in the central fluid parameters. It is corrected in dependence of pressure and temperature.

If the *Manual Setting of Bulk-modulus* is chosen, the given bulk-modulus is used for calculation.

If the *Correction of Bulk-modulus* is marked, the bulk modulus of the hydraulics liquid is corrected depending on the pressure. Otherwise, the bulk modulus remains constant.

Via the parameters *PressureGradient* the gradient of the pressure raise and drop can be limited.

As many hydraulic connections as desired can be connected to a THERMOHYDRONODE.

THERMOHYDRONODES and the hydraulic lines between the components are represented in dark red colour.

 © FLUIDON® GmbH 2008

3 HydroVLNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
p	V	bar	pressure
Q	V	l/min	flow
A	V	-	
C	V	-	

Parameters

Name	Value	Unit	Description
-	-	-	

Notes

The component HYDROVLNODE allows the connection of special hydraulic components without any volume.

The HYDROVLNODE transmits the signals for pressure (*p*), flow (*Q*) and the variables *A* and *C* between the connectors of the components. Just two components can be connected.

The HYDROVLNODES and the hydraulic lines between the components are represented in magenta colour.

4 PneumatNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
Pressure	Pneu	bar	Pressure at the node
T	Pneu	K	Temperature at the node
Pdot	V	-	internal auxiliary variable
Tdot	V	-	internal auxiliary variable

Parameters

Name	Value	Unit	Description
Volume	1	l	Gas volume between the components
Source	0	-	Switch for activation of pressure source function

Notes

The PNEUMATNODE represents the gas volume between the connected components as an ideal capacity without any resistance behaviour. The parametrizable volume consists of the volume of the line which connects the components and the dead volumes. The dead volumes are those gas volumes, which can not be displaced (in e.g. valves, the volume between connection and control edge). Furthermore, the correction volumes are automatically added to the node. An accumulator delivers e. g. a constant correction volume, and a cylinder a variable correction volume.

If the parameter *source* is marked, the pressure source function the node is switched on. As a result the PNEUMATNODE works as a constant pressure source and both the pressure and the temperature were adjusted over the initial values.

PNEUMATNODE and the pneumatic lines between the components are represented in the drawing area in a dark blue colour.

 © FLUIDON® GmbH 2008

5 MechTransNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
x	V	-	Stroke
v	V	-	Velocity
F	V	-	Force

Parameters

Name	Value	Unit	Description
-	-	-	

Notes

A MECHTRANSNODE transmits the signals for a stroke (*x*), a velocity (*v*) and a force (*F*) between the mechanical connections of different components. The MECHTRANSNODE and the connecting lines between the components are represented in the drawing area in dark grey colour.

6 MechRotNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
phi	V	-	angle
omega	V	-	angular velocity
M	V	-	torque

Parameters

Name	Value	Unit	Description
-	-	-	

Notes

A MECHROTNODE transmits the signals for an angle (*phi*), an angular velocity (*omega*) and a torque (*M*) between the mechanical connections of different components. The MECHROTNODE and the connecting lines between the components are represented in the drawing area in light grey colour.

 © FLUIDON® GmbH 2008

7 SignalNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
Value	V	-	transmitted signal

Parameters

Name	Value	Unit	Description
-	-	-	

Notes

A SIGNALNODE transmits a signal between the signal connectors of different components.

As many input connectors as desired can be connected to a SIGNALNODE, but only one output connector.

If a signal output is connected to a HYDRONODE, it changes into a SIGNALNODE. The pressure in the node is determined by the signal value.

SIGNALNODES and the connections between the components are represented in the drawing area in green colour.

8 DigitalSignalNode



Symbol

Connectors and Variables

Name	Type	Unit	Description
Wert	V	-	transmitted signal

Parameters

Name	Value	Unit	Description
-	-	-	-

Notes

A DigitalSIGNALNODE transmits a signal between the signal connectors of different digital components.

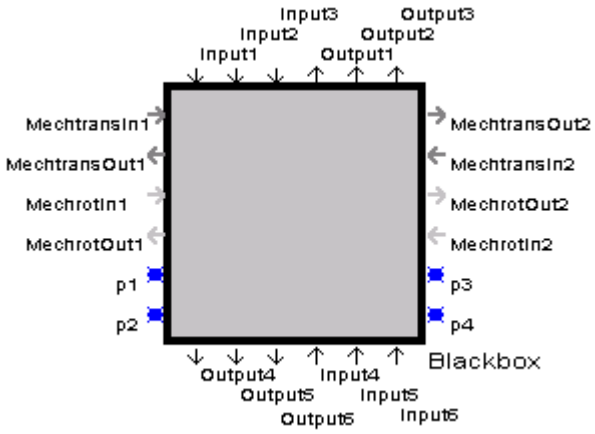
As many input connectors as desired can be connected to a DigitalSIGNALNODE, but only one output connector.

SIGNALNODES and the connections between the components are represented in the drawing area in dark green colour.

 © FLUIDON® GmbH 2008

Appendix B Component BlackBox

BlackBox:



Symbol

Connectors and Variables

Name	Type	Unit	Description
p1	Hydr	bar	
p2	Hydr	bar	
p3	Hydr	bar	
p4	Hydr	bar	
Input1	In	-	
Input2	In	-	
Input3	In	-	
Input4	In	-	
Input5	In	-	
Input6	In	-	
Output1	Out	-	
Output2	Out	-	
Output3	Out	-	
Output4	Out	-	
Output5	Out	-	
Output6	Out	-	

© FLUIDON® GmbH 2008

Name	Type	Unit	Description
MechtransIn1	Mech	-	
MechtransIn1F	Out	N	
MechtransIn1x	In	mm	
MechtransIn1v	In	m/s	
MechtransIn2	Mech	-	
MechtransIn2F	Out	N	
MechtransIn2x	In	mm	
MechtransIn2v	In	m/s	
MechrotIn1	Mech	-	
MechrotIn1M	Out	Nm	
MechrotIn1phi	In	rad	
MechrotIn1omega	In	rad/s	
MechrotIn2	Mech	-	
MechrotIn2M	Out	Nm	
MechrotIn2phi	In	rad	
MechrotIn2omega	In	rad/s	
MechtransOut1	Mech	-	
MechtransOut1F	In	N	
MechtransOut1x	Out	mm	
MechtransOut1v	Out	m/s	
MechtransOut2	Mech	-	
MechtransOut2F	In	N	
MechtransOut2x	Out	mm	
MechtransOut2v	Out	m/s	
MechrotOut1	Mech	-	
MechrotOut1M	In	Nm	
MechrotOut1phi	Out	rad	
MechrotOut1-omega	Out	rad/s	
MechrotOut2	Mech	-	
MechrotOut2M	In	Nm	
MechrotOut2phi	Out	rad	
MechrotOut2-omega	Out	rad/s	

Parameters

Name	Standard value/ standard unit	Description
Parameter1	0 -	
Parameter2	0 -	
Parameter3	0 -	
Parameter4	0 -	
Parameter5	0 -	
Parameter6	0 -	
Parameter7	0 -	
Parameter8	0 -	
Parameter9	0 -	
Parameter10	0 -	

Notes

BLACKBOX is an element for experiments with external functions (see chapter 6, 4).

© FLUIDON® GmbH 2008

Appendix C Examples of simulations

In this chapter exemplary simulation models are presented. In this manner, the customer gains a feeling for the set-up of own models. Furthermore, the contents of the component library and the broad application spectrum of the simulation program becomes clear.

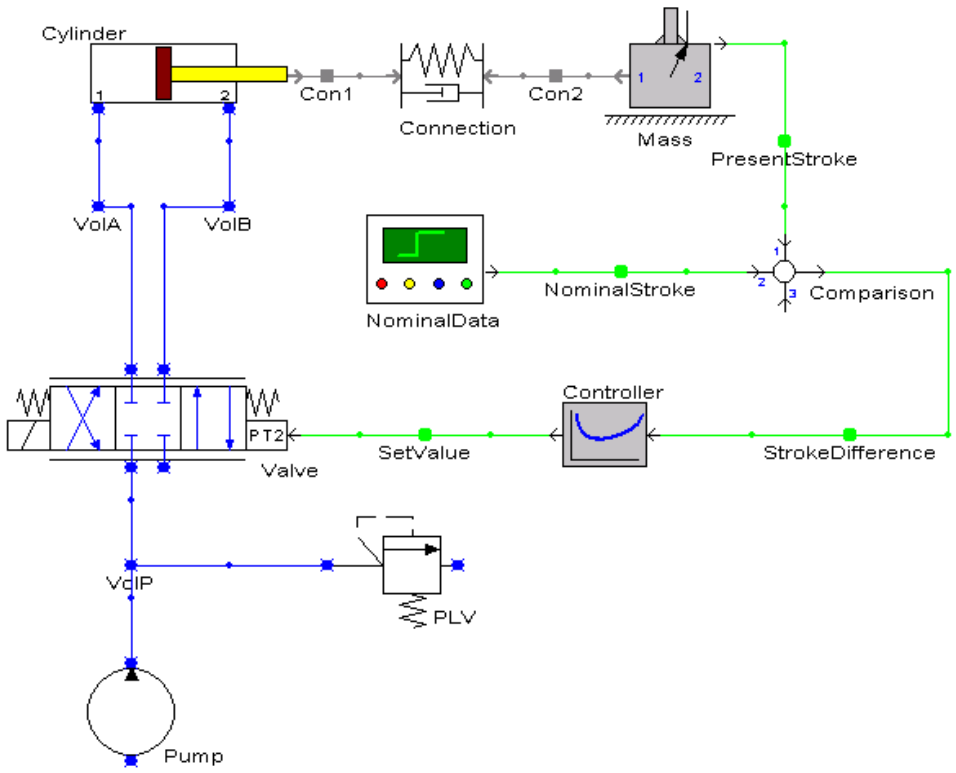
The models described here are in the demo model directory of DSH*plus* which is the selected directory after installation and is available to the customer for training with the simulation software.

1 Cylinder control

The first simulation example shows an analogous position control for a cylinder. The model consists of an unchangeable pump which supplies a servo valve with volume flow. The desired system pressure can be varied via the PRV. The valve controls a cylinder which is coupled with a movable mass over a spring damper element.

Corresponding to the value set (DEMANDEDSTROKE) by the component SETVALUE the servo valve is actuated by the output of the controller (CONTROLSIGNAL) for so long until the actual way measured off at the mass (PRESENTSTROKE) matches the demanded way. The controller closes the valve again.

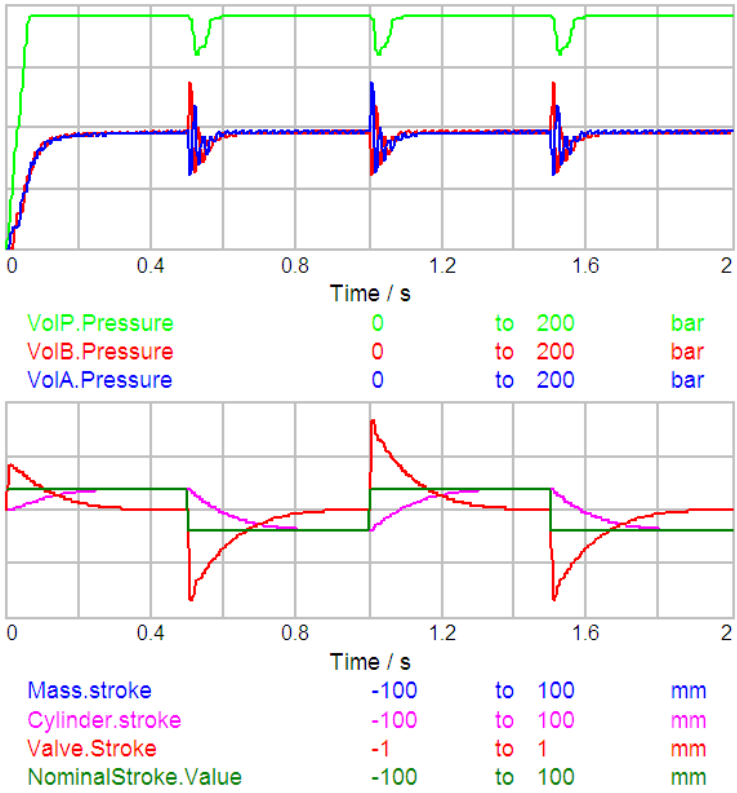
 © FLUIDON® GmbH 2008



Simulation model of a closed-loop control system

By means of this model very simply held simulation model of a closed-loop control system different effects typical for fluid power systems can be simulated. In such a way, the increase of pressure can be considered for each individual volume node and the node characteristics can also be influenced via its parameterization. In the picture, this is to be recognized clearly for the rise of the supply pressure VolP at the beginning of simulation.

As further signals, the valve signal, the cylinder way and the pressure in the cylinder chambers are displayed in the picture. That it is recognized clearly that the system tends to continuous oscillations which is caused by the controller parameters. The customer can here vary different control parameters in order to receive the system into a stable operating point.



Simulation graphics of the cylinder control

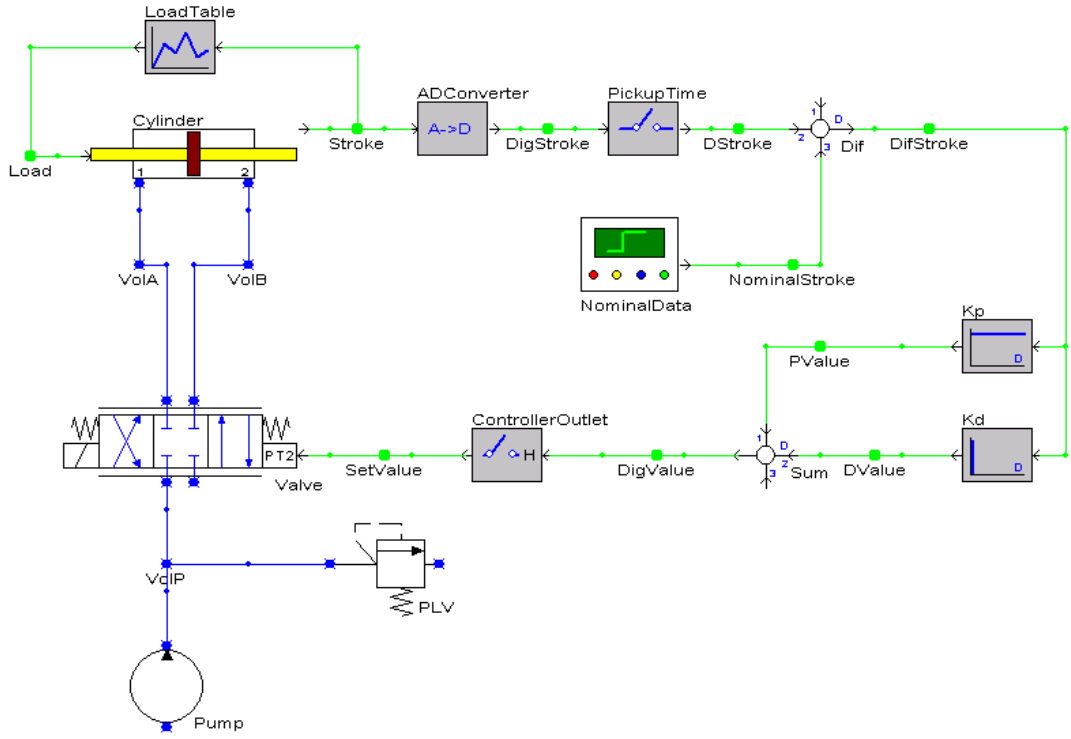
With the variation of the cylinder friction or the friction of the moved mass the influence of solid state- and/or viscous friction to the system cant be examined. In this manner, brake away effects which lead to embroidery slip effects during the control cycle can be simulated.

Furthermore, it is possible to modified the spring damper element or the valve's flow characteristic curves or its dynamic.

 © FLUIDON® GmbH 2008

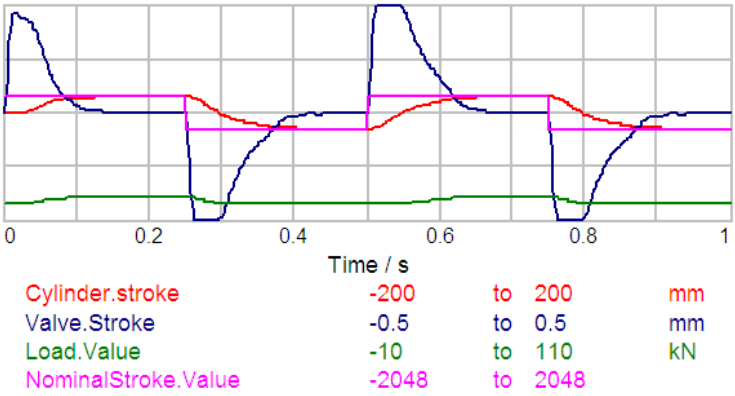
2 Digital cylinder control

The second simulation example shows in principle the same model construction as example 1. Merely the analogous control elements are replaced through digital components. As further innovation, the cylinder is not burdened by mechanical interaction with the moved mass but it burdened by a way dependent load over a look-up table.



Simulation model of the digital cylinder control

The simulation model of the digital cylinder control provides the user the possibility to test the influence of different resolutions of the A/D converter and/or of settings of the sampling or CPU time on the control effectiveness and the control characteristic of the system.



Trajectories of a step response test

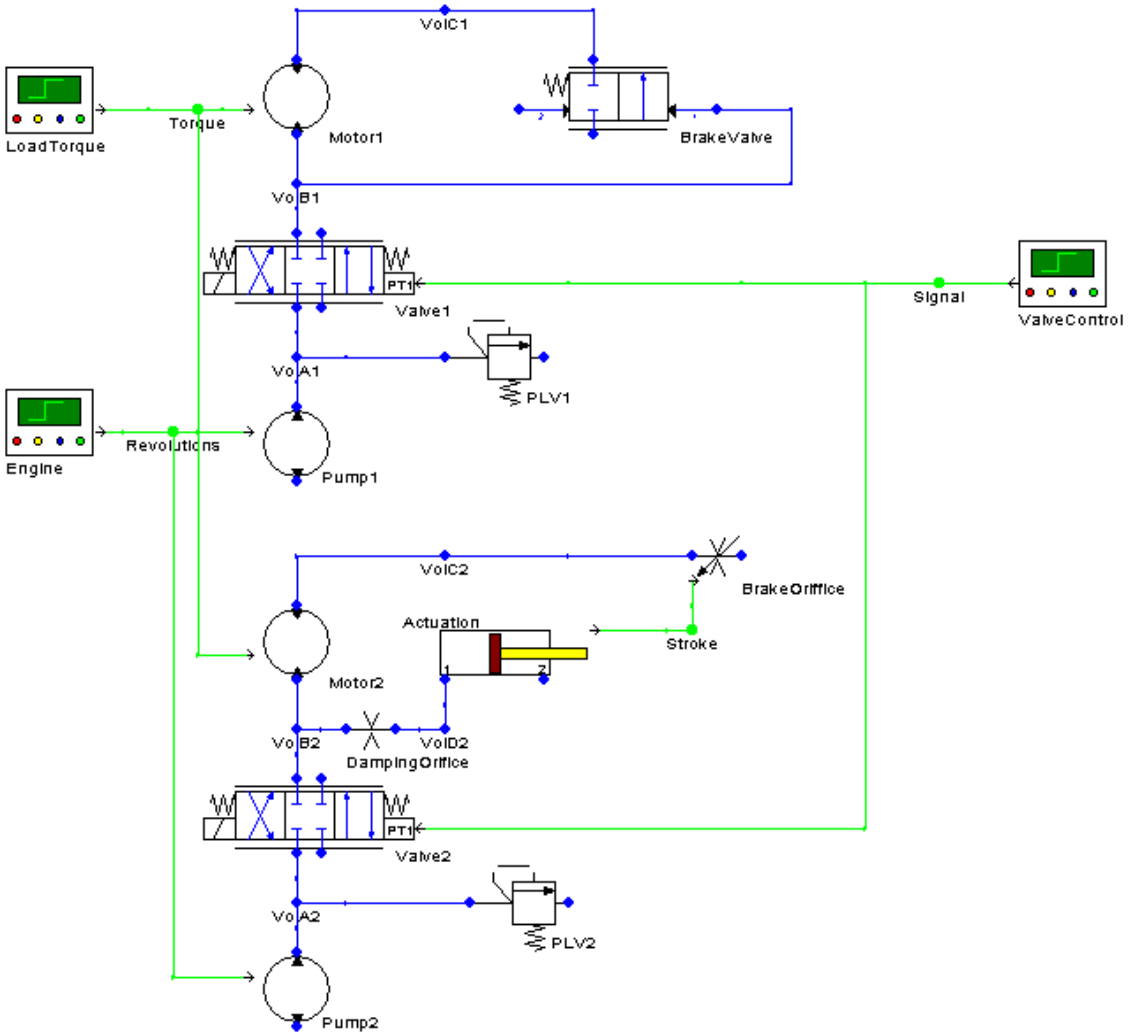
Through variation of the look-up table the stroke dependent load that affects the cylinder can be varied in a way that the working stroke of a press e.g. can be simulated. In addition, one can bring the cylinder into a normal position by means of the parameter ORIENTATION in the component cylinder in order to consider the mass strengths of the tie beam.

 © FLUIDON® GmbH 2008

3 Hydrostatic drive

The third simulation model represent a strongly simplified hydrostatic drive of an arbitrary vehicle. The upper model includes the deceleration valve integrated into the drive as a complex component, the lower model built up almost identically shows how the deceleration valve can be built up of single components.

Intention of this simulation model is it to show to the user how simply it is possible to model components with DSH*plus* those are not yet contained at the model library by a connecting of basic components.



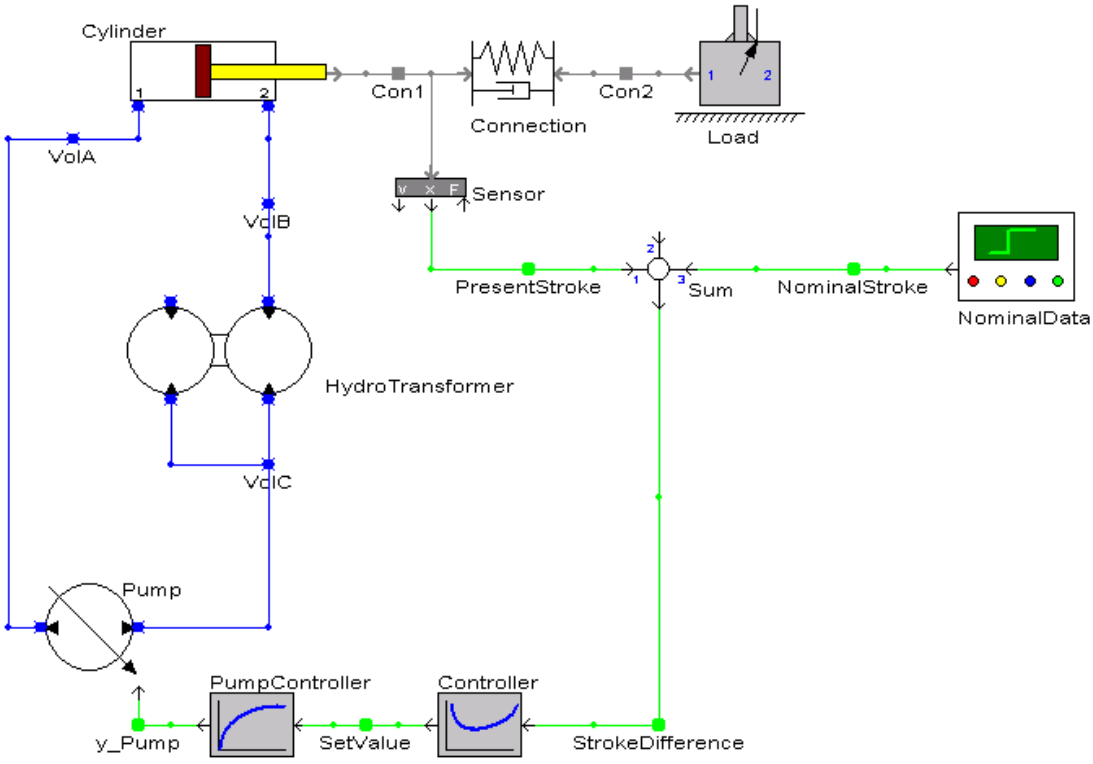
Simulation model of the drive complex (above) and modular (below)

During simulation the user has the possibility to adjust the pump volume flow via the component DEMANDEDROTATIONSPEED, the torque applied to the engine via LOADTORQUE and by means of the function generator VALVEEXCITATION the valve position. Different vehicle velocities and the change of the topographical profile can be represented during the simulation by a suitable combination of these parameters.

 © FLUIDON® GmbH 2008

4 Hydro transformer

The example hydro transformer shows how it is possible to operate a differential cylinder in the closed hydrostatic circuit with the aid of the double engine.

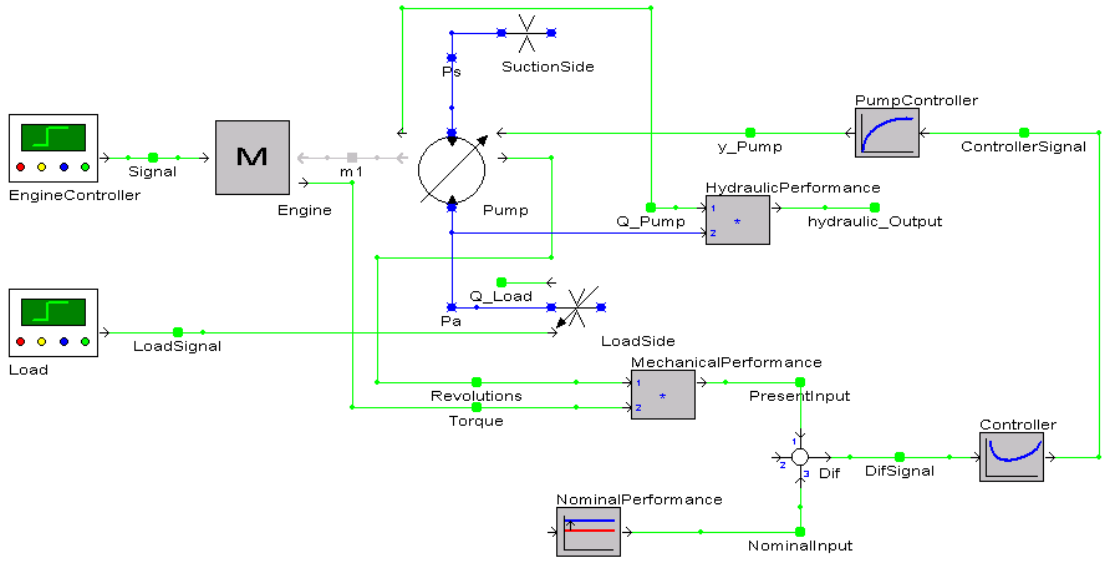


Simulation model of the double engine as a hydro transformer

If the displacement volumes of the hydraulic motors are adapted to the area ration of the cylinder, one can control the cylinder merely through the change of the pump flow.

5 Diesel engine driven pump

The simulation example of the diesel engine driven pump shows the user, that it is now possible with DSH*plus* to integrate also drive technology components into the calculation.



Simulation model of the diesel engine driven pump with testing environment

By suitable parameter selection the user can simulate a load regulated pump, that is driven by a diesel engine.

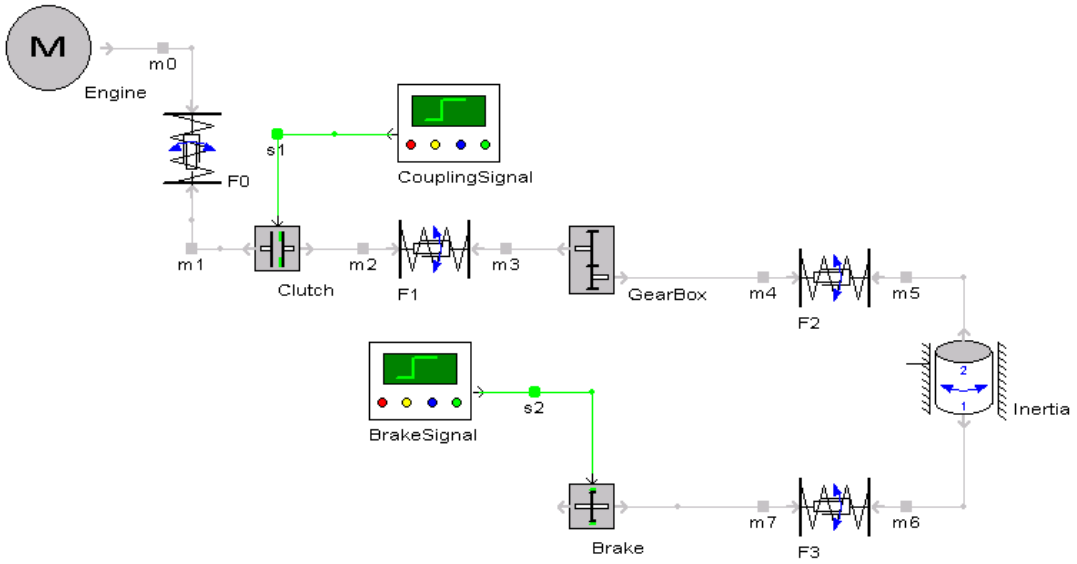
In addition to the calculation of the efficiencies and the input of different speed-load-curves of the engine the speed decrease of the diesel engine in the case of too much hydraulic load can be simulated.

© FLUIDON® GmbH 2008

6 Mechanical drive train

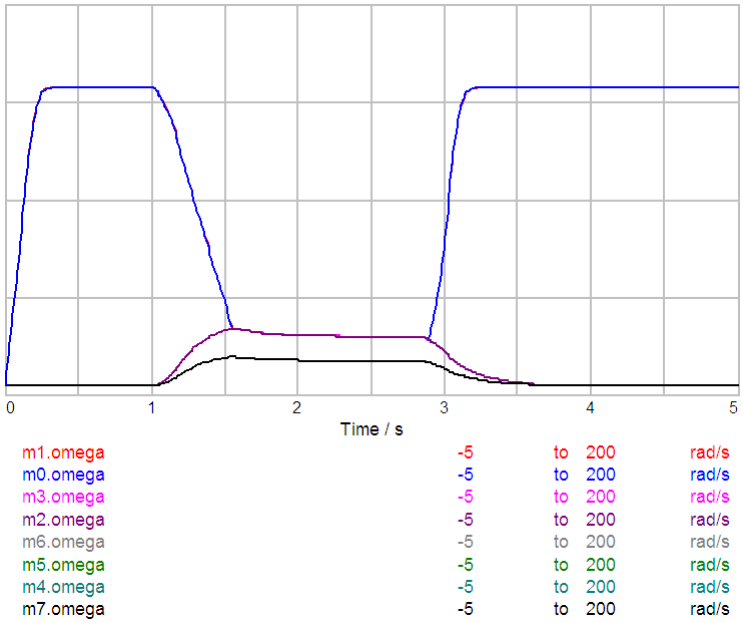
The simulation example mechanical drive train serves a introduction of the library of simply modulated mechanical and drive elements which is at this time under evolution.

The model includes an asynchronous motor that drives over a clutch and a gear an inertia, that again is coupled with a brake.



Simulation model of a mechanical drive train

In the example simulation the asynchronous motor has already started. After some time the clutch is then activated and one can see the coupling effects at the acceleration of the inertia. An comparison of the torque and of the rotational speeds before the gear and after the gear shows the influence of the losses parameterized in the gear.



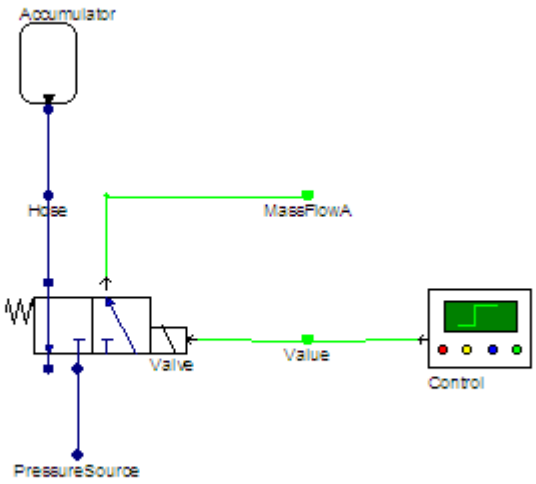
Simulation results of the drive train

After the clutch again has been released, the inertia is first slowed down by viscous friction and final slowed down to the standstill through the break.

 © FLUIDON® GmbH 2008

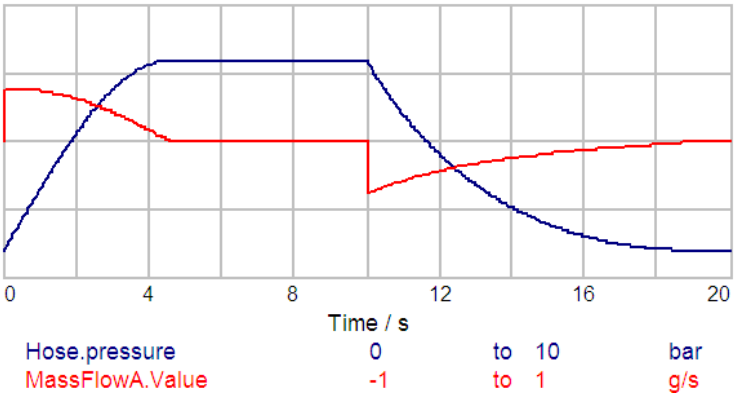
7 Pressurization and ventilation of a volume

In this simulation example the course of events by pressurization and ventilation of an accumulator is considered. This process appearing quite trivial at first can only be very difficult computed analytically since the mass flow via the valve changes constantly according to the accumulator pressure.



Simulation model to pressurize and ventilate a volume

The simulation model exist from an unchangeable pressure source, a spring returned 3/2 way switching valve accessed electrically, an accumulator and a function generator. The pneumatic node represents the hose between valve and storeroom. The state variables of the accumulator can be measured off here. In order to generate the mass flow over the valve as a simulated value the signal node MASSFLOWA is put on.



Simulation graphics of the pressurization and ventilation

The function generator specifies a signal to the switching valve through what the accumulator is combined with the pressure source. After a defaulted space of time, the signal is put to zero. As a result, the spring shifts the valve into the initial position and the accumulator is purged for the environment.

Two selected state variables of the model are represented over the simulation time in the simulation graphics.

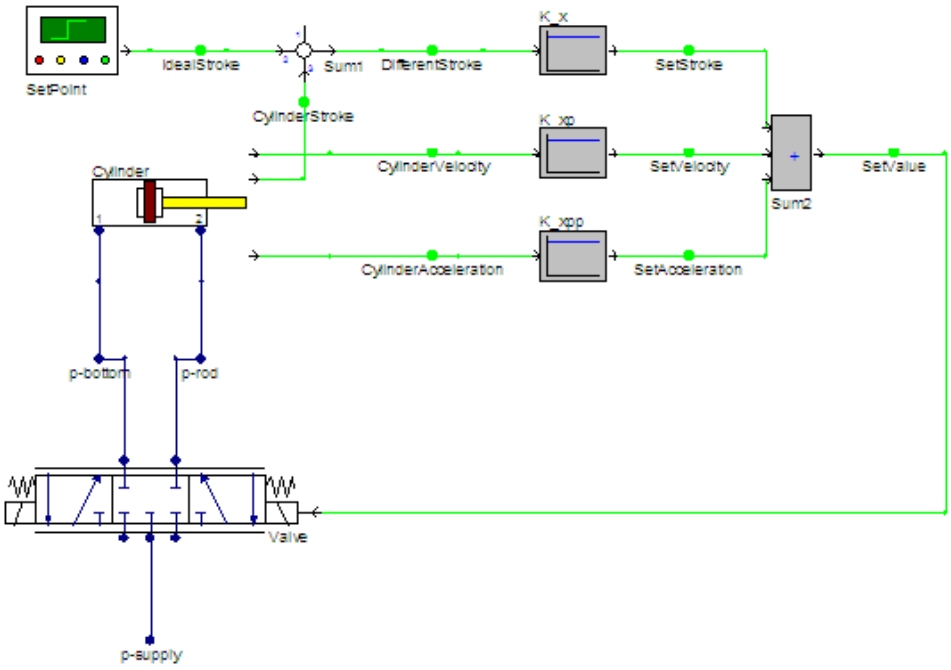
At first the accumulator pressure (blue) increases steeply from the ambient starting pressure of 1 bar. The mass flow (red) is at first constant, since the pressure before the valve is constant too and the accumulator pressure is still low - hypercritical mass flow. After the accumulator pressure has exceeded a specific boundary value, the mass flow becomes sub critical and declines. The gradient of the accumulator pressure decreases correspondingly more and more and fills up asymptotic onto working pressure from 8 bar.

In case of the ventilation of the system the accumulator pressure represents the pressure at the metering edge of the valve. For this reason, the mass flow at first is hypercritically, however it is not constant. Consequently, the storeroom pressure declines off steeply, but needs a very long time to achieve the ambient pressure of 1 bar.

 © FLUIDON® GmbH 2008

8 Cylinder in a closed-loop control system

The next simulation example is a cylinder in a closed-loop control system. This example clarifies the suitability of the simulation program for design of pneumatic systems. The optimization of the system can be carried out both at the components of the controlled system (cylinder, valve) and by the attitudes of the automatic controller.

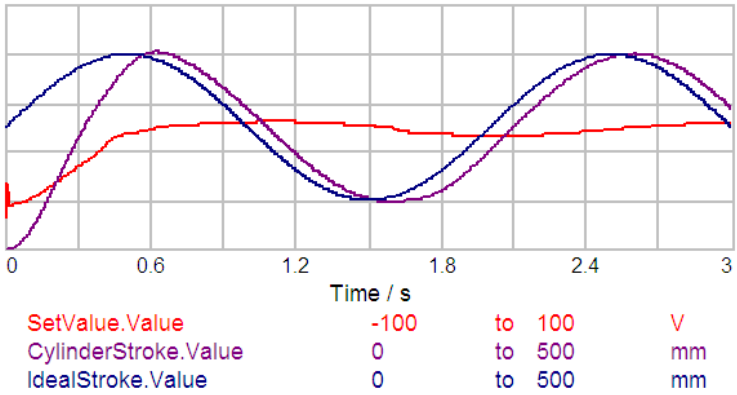


Simulation model of the cylinder in a closed-loop control system

The simulation model consists of a differential cylinder, an constant pressure source and a 5/3 way servo valve. The state variables of the cylinder are measured off at the cylinder and transmitted to an automatic controller with stroke, velocity and acceleration feedback. The manipulated variables are summarized at a summer and switched to the entry of the servo valve. The demanded stroke is applied via a function generator.

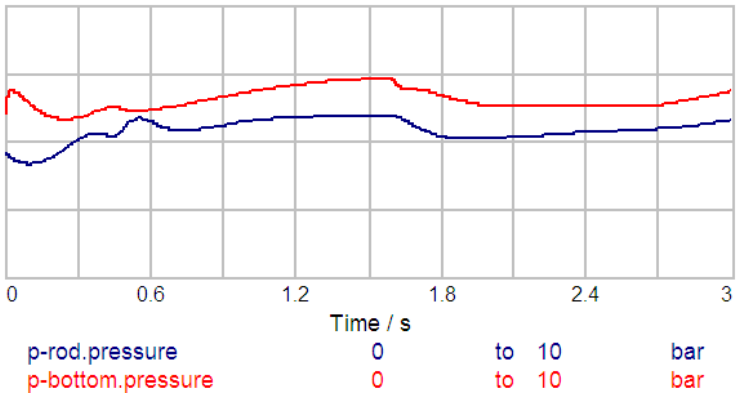
In the next simulation graphics, the stroke of the cylinder in the case of a sinusoidal input signal can be seen (rblue) from 100 to 400 mm at a frequency of 0.5 Hz.

The cylinder stroke (violet) nearly achieves its demanded position after 0.6 s almost without overshooting. The green curve shows the actuator signal which is delivered to the entry of the servo valve.



Trajectory of cylinder stroke, demanded stroke and actuator signal

The pressures of the cylinder chambers calculated in the simulation are represented in the following picture.



Trajectory of the pressure in the cylinder chambers

With the aid of the control system parameters the user is able to correct the dynamics of the system. In addition, he/she can implement easily modifications of the controlled system (bigger cylinder, smaller valves or different working pressures) through the parameterization of the components.

 © FLUIDON® GmbH 2008

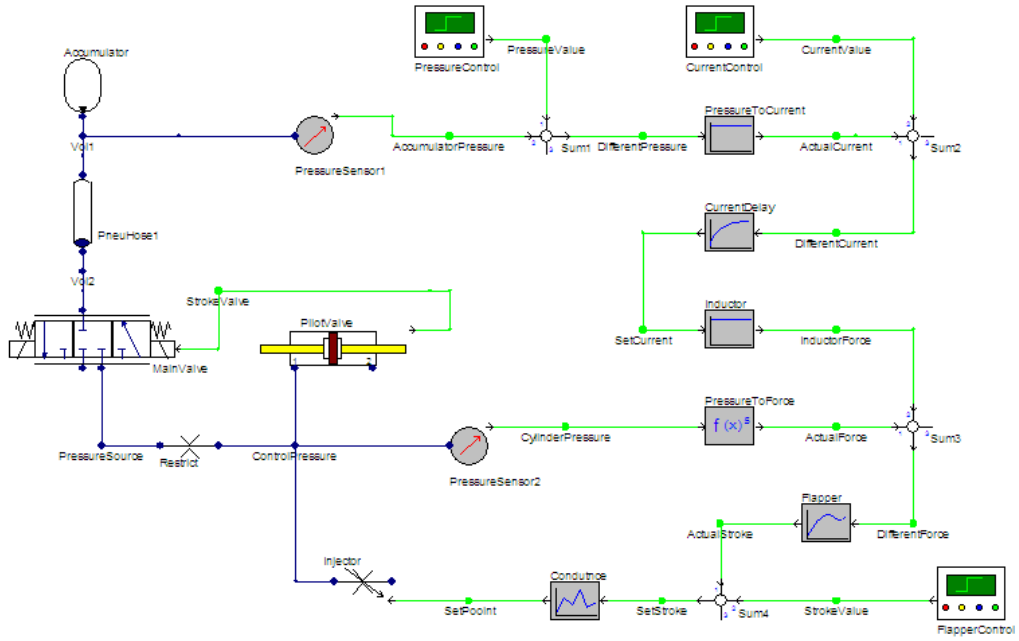
9 Modelling of a pressure regulating valve

This example represents the modelling of a pressure regulating valve (pressure governor). Although the component library of DSH*plus* already has two simplified pressure regulating valves, it should be shown with this example how components more complex can be made with DSH*plus* through the connection of single standard components.

The simulation model essentially consists of the pressure regulating valve (main stage, control spool, flapper-nozzle system), a consumer unit (hose, accumulator) and the signal chain.

The central component of the pressure governor forms the main stage, a 3/3-way-continuous valve. The stroke of a controlling spool, a piston-rod less spring returned cylinder, is applied to the entry of this valve. Onto the cylinder the control pressure fitting between the nozzle and the flapper-nozzle system is applied. If the control pressure increases, the control spool displaces itself to the left. As a result, the main stage connects the consumer unit with the pressure source. If the control pressure is reduced, the return spring presses the control spool to the right and this purges the consumer for the environment. The control spool is in the middle in the initial position i.e. the consumer is neither pressurized or ventilated.

© FLUIDON® GmbH 2008



Simulation model of modelled pressure regulator

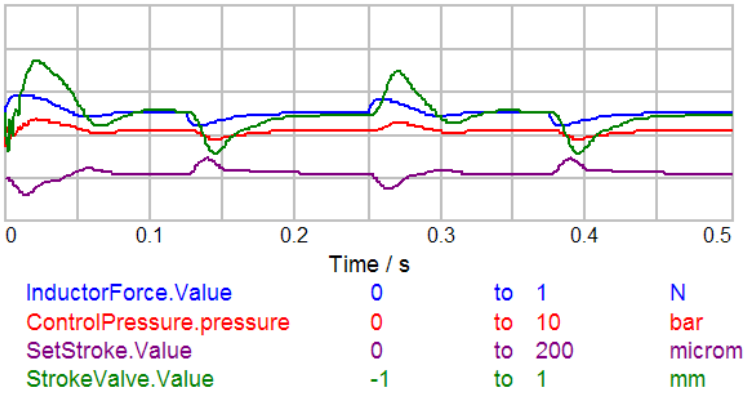
The regulation the control pressure occurred via a flapper-nozzle system that is deflected by a voice coil in a permanent magnetic circuit against a membrane spring. This closed-loop control system is implemented with the aid of the signal components of the controller library. Starting from the deviation between should and actual pressure, a setting current is first generated via a proportional element. In the sum with the offset current, a force proportional to the coil current results from this. The current construction is considered by a PT1 element. If one now forms force balance at the deflecting electrode considering the elastic and dampening behavior of the membrane spring, the high dynamic behavior of the deflecting electrode can be described with the aid of a PT2 element. In order to regard of the flow force, the control pressure is multiplied by the active area of the deflecting electrode and also included in the force balance. Finally the conductance of the adjusting nozzle is determined in dependence of the absolute distance (flapper-nozzle) from an one dimensional look-up table.

In the first simulation, the transient of the flapper is examined for the case that the reactance coil becomes charged merely with an offset current(zero current). In this case, the absolute distance of the flapper of the measuring nozzle must be chosen in such a way that the control

 © FLUIDON® GmbH 2008

spool is in the middle position after the transient and the pressurization and ventilation connections of the main stage are locked.

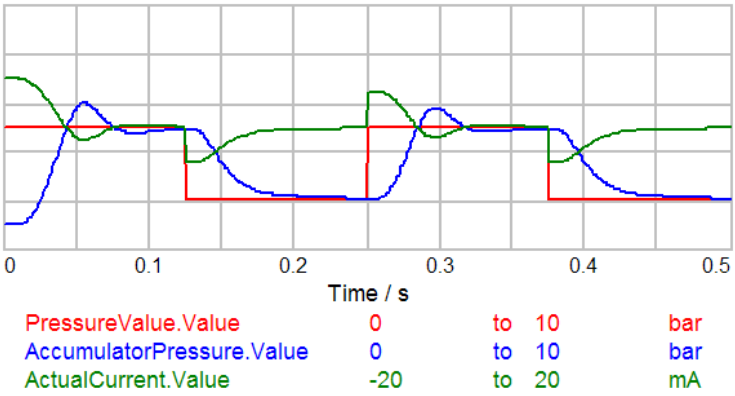
The trajectories of the control pressure, of the control stroke of the valve spool, of the coil force and of the distance (deflecting electrode measuring nozzle) over the time are represented in the following simulation graphics.



Simulation graphics for the transient of the control

After the control spool centered itself in middle position, the static state variables are stored and used as initial values for the subsequent pressure regulation.

A square wave is chosen as debit pressure (red) with a jump from 2 bar to 5 bar at a frequency of 4 Hz. One sees how the present pressure follows the should pressure (blue) relatively fast, the required value already after 0.05 s exceeds and still requires 0.05 s in order to work out onto the required value. At the declining flank, the pressure governor need due to the lower pressure difference more time to achieve the defaulted required value, remains yet more than overshoot free. The setting current is also represented (green), that is charged to the deflecting electrode and therefore causes the change of the control pressure.



Simulation graphics of the pressure regulating valve

© FLUIDON® GmbH 2008

Appendix D Miscellaneous

1 Configuration of compiler and linker

Before the program function **Making + Compiling** is carried out interactively in the dialog **STATUS MODEL BUILD UP**, the compiler and the linker must be configured. This can be done in the register board **Compiler and Linker** of dialog ***Options***.

The compiler and the linker are configured via response and configuration files and called via a batch file. As a result, one can work with command lines longer than most operating systems approve. In addition, the paths and options need not be entered each time anew. The response files contain the paths necessary for compiler and linker while the configuration files summarize the options for the compiler and the linker.

DSH*plus* is prepared for use of the command line compiler BCC32 of Borland and of the Borland linker TLINK32 however, one can work fundamentally with every compiler and linker that is able to create DLL's.

The work files with attitudes for the Borland compiler and linker are in DSH*plus* available as file presentations. From these presentations and the attitudes of the register board **Compilers and Linker** of the input dialog **OPTIONS**, the batch file, the response and configuration file of the compiler and the response file of the linker are made. (The options for the linker are filed in the response file of the linker together with the paths.)

If for the work with DSH*plus* the Borland compiler is employed, only the correct paths of compiler and linker must be adjusted in the register board. It one works with another compiler, the user must make the templates for the work files with the contents necessary for the compiler and the corresponding syntax him/herself. Since however this depends on the respective compiler, at this point not further help can be given but be referred only to the manuals of the employed compiler.

In the following sections the work file templates provided by DSH*plus*

 © FLUIDON® GmbH 2008

and from which the work files are made for the Borland compiler are more closely introduced.

The following table shows from which templates the work files are made:

Denotation	File template	Work file
Batch file of compiler and linker	dglbat.dsh	dshdgl.bat
Response file of the compiler	comprsp.dsh	comprsp
Configuration file of the compiler	dglcfg32.dsh	dshdgl.cfg
Response file of the linker	lnkrsp32.dsh	lnkrsp

The template files contain keywords which are printed fat in the following source code. For the generation of the work files, these keywords are replaced.

From the template *dglbat.dsh* the batch file *dshdgl.bat* will be derivated, which calls the compiler and the linker:

```
/* Datei dglbat.dsh */

Compiler @%1\comprsp >  %1\result.txt
Linker   @%1\linkrsp >> %1\result.txt
@echo off
```

In this case, *compiler* becomes automatically replaced through the EXE file of the compiler and *linker* through the EXE file of the linker (both with its paths according to the information stored in **Compiler and Linker** in the **OPTIONS** dialog).

The file *comprsp.dsh* is the template for the response file of the compiler:

```
/* Datei comprsp.dsh */
```

```
+Configuration
-nPfad/Path
```

If the work file is made from this file, *Configuration* is replaced by the name of the Configurations file of the compiler with its path. The name of the file which should be compiled is registered in the case of pfad/path with path (source code of the model).

The compiler options are stored in the file *dglcfg32.dsh*:

```
/* Datei dglcfg32.dsh */

-c
-5
-v-
-G
-O
-Oc
-Jgd
-Vms
-Vp
-Vc
-Z
-k-
-V
-vi-
-w-bei
-w-lin
-w-lvc
-w-ret
-w-nci
-w-inl
-w-obi
-w-ofp
-w-ovl
-wpin
-wamb
-wamp
-w-par
-wasm
-wcln
-wdef
-wsig
-wnod
-wstv
-wucp
-wuse
-weas
-wpre
-
D_RTLDLL;_BIDSDLL;_OWLDLL;STRICT;_OWLPCH;_OWLALLPCH;WIN32BIT;
UNDER_WIN95
-P-.
-IInclude-Pfade/Include-Pathes
```

 © FLUIDON® GmbH 2008

In the work file made from the file *dglcfg32.dsh* is *include-pfade/include-path* replaced through the paths of the include files according to the information stored in **Compiler and Linker** in the **OPTIONS** dialog.

Since the linker must only be configured with few options, they are contained directly in the template for the response file of the linker *lnkrsp32.dsh*:

```
/* Datei lnkrsp32.dsh */

/Tpd/x/c/v /LLib-Pfade/Lib-Pathes +
c0d32 Objectfiles
DLL-Name,
cw32 import32 Libfiles
ModulDefinitionFile
```

Lib-Pfade/Lib-pathes is automatically replaced while making the work file through the directories of the **Libs** according to the settings stored among **Compiler and Linker** in the **OPTIONS** dialog. Among *Objectfiles* the objects of the model, made by the compiler, and if appropriate additional objects are registered. *DLL-Name* indicates the name of the DLL file which should be made by the linker. *Libfiles* is replaced by the file names of the additional Libs and instead of a *ModuleDefinitionFile*, the module header file is entered in the work file with its path.

The module header file *dshdgl32.def* supplies information to the linker about the contents and the system request of a Windows application:

```
/* Datei dshdgl3.def */

LIBRARY DSHDGL3 INITINSTANCE
DESCRIPTION 'Modell DSHplus 3 c 2001 FLUIDON GmbH'
CODE LOADONCALL EXECUTEREAD
DATA LOADONCALL MOVEABLE SINGLE NONSHARED

EXPORTS
  @Begrenzungen$qpg          @3 ; Begrenzungen(long double *)
  @Daten$qv                  @1 ; Daten()
  @Dgl$qgpgt2gt25FLAGS       @5 ; Dgl(long double,
                                        long double*,
                                        long double *,
                                        long double,
                                        long double *, FLAGS)
  @ExitFunktion$qv           @6 ; ExitFunktion()
  @Grenzwerte$qp9GRENZWERT   @4 ; Grenzwerte(GRENZWERT *)
```

```
@Initialisierung$qp4DATApc @2 ; Initialisierung(DATA *,
                                                char *)
```

```
@Output$qgpggt2t25FLAGS     @10 ; Output(long double,
                                        long double*,
                                        long double,
                                        long double *,
                                        long double *,
                                        FLAGS)
@getLocalKey$qv             @11 ; getLocalKey()
```

All files and paths necessary for the compiler and the linker must be registered in the register board **Compiler and Linker** of the dialog **OPTIONS**. The following table explains the input fields of the register board, in each case a example entry is given in brackets.

Input field	**Explanation**
Dateien	
Compiler	Compiler name with path (c:\program files\fluidon\dshplus\bc\bin\bcc32.exe)
Batch file	Template of the batch file (DglBat.dsh)
Compiler Response File	Template of the response file of the compiler (CompRsp.dsh)
Compiler Configuration File	Template of the configuration file of the compiler (dglcfg32.dsh)
Linker	Linker name with path (c:\program files\fluidon\dshplus\bc\bin\tlink32.exe)
Linker Response File	Template of the response file of the linker

 © FLUIDON® GmbH 2008

Input field	Explanation
	(with options) (LinkRsp32.dsh)
Module Definitions File	Standard is *dshdgl3.def* (DshDgl3.def)
Additional Libs	Additional Libraries (DSHTools3.lib)
Additional Includes	Additional Include Files
Additional Objects	Additional Object Files
Directories	
Include files	Directory of the include files (c:\program files\fluidon\dshplus\include; c:\program files\fluidon\dshplus\bc\include)
Libs	Directories of the lib files (c:\program files\fluidon\dshplus\lib; c:\program files\fluidon\dshplus\bc\lib)

Note:

Under additional libraries must the library of the mathematical one

- Under *include files* must both be specified the directory of the include files of the compilers and of DSH*plus*. In the same way the directories of the libraries of the compiler and of DSH*plus* must be specified under *Libraries*.

 © FLUIDON® GmbH 2008

Index

auxiliary parameter 148
compile 64
component 20, 146, 175
  align to grid 60
  auxiliary function 150
  auxiliary parameter 148
  connect 25
  connect several 53
  end 160
  exit function 151
  functions 152
  mark 55
  mathematical description 146
  move 56
  parameter 147
  place 23
  programming 175
  programming rules 178
  select 23
  source code 146
  variables 149
component connections
  connect 25
component library 20
  add component 175
component programming 175
  data format 183
  deftyp.h 175
  deftyp2.h 183
  dglthermotools.h 202
  external functions 227
  filter class 200
  friction 195
  general functions 195
  look-up table 196
  mathematical constants 184
  mathematical functions 186
  message function 205
  model source code 207
  pneumatic functions 204
  pneutools.h 204
  procedure 175
  ring memory 198
  specific definitions 180
  specific functions 180
  thermo tools 202
component-connections 25
design mode 51
differential equations 132
drawing area 18
exit 51
extras 120
  options 121
function line 17
functions 152
headline 17
help
  component library 127
  content 127
hydraulic state variables 133
hydro nodes 25
info about 127
initial conditions 132
input mode 20
installation 13
  uninstall 14
look-up table 82
  create 83
  delete 88
  import data 85
  load 86
  save 87
  save as 87
mechanical nodes 25

 © FLUIDON® GmbH 2008

menu line 17
model 20, 51
  align components to grid 60
  compile 64
  connect component 52
  connect several components 53
  copy 63
  copy component groups 56
  copy components 56
  create 64
  design 52
  designing 20
  drawing functions 52
  edit 52
  edit source code 66
  group components 59
  group name 56
  mark components 55
  move components 56
  paste 63
  paste component 52
  redo 64
  refresh 61, 63
  save 61
  select all 63
  select component 52
  source code modification 207
  stop edit 61
  undo 63
multi-user deployment 12
network usage 12
nodes 25
  hydro nodes 25
  mechanical nodes 25
  pneumatic nodes 25
  signal nodes 25
number pages 124
options 121
  compiler and linker 122
  directories 121
  graphic options 123
  page set-up 124
parameter 30, 67
  component parameter 67
  delete 75
  display 76
  load 70
  look-up table 82
  save 73
  save as 73
  save component parameter 74
  save parameter set 73
  setting 30
  simulation parameter 68
  standard values 160
  start values 68, 104
pneumatic nodes 25
pressure 133
print 50
print header 124
programming rules 178
project
  author 50
  close 40
  import 46
  info 50
  name 50
  new 44
  open 45
  remark 50
  save 49
  save as 49
  template 44
set print size of simulation graphics 124
signal nodes 25
simulation 90
  abort calculation 35

configure graphics 36
current values 39
final value as initial 104
graphics 36
parameter 91
progress 90
save results 101
simulation graphics 36
simulation progress 35
start calculation 35, 90
stop calculation 91
simulation graphics
adapt 97
copy to clipboard 99
set print size 124
simulation mode 29
simulation model 131
component definition 165
data 172
differential equation 168
exit function 167
initialization 166
limitations 171, 172
limits 167
model source code 162
source code 162
source code header 163
simulation parameter 32
solvers 133
source code 131
select editor 125
standard values 160
start values 68, 104, 132
status line 18
system requirements 12
uninstall 14
unit file 161
units 161
adjusting 30
user interface
drawing area 18
function line 17
headline 17
menu line 17
status line 18
variables 30
adjusting 30
standard values 160
volume flow 133
window 119
cascade 119
minimize 119
restore 119
standard size 119

 © FLUIDON® GmbH 2008