
OpenModelica User's Guide

Release v1.9.4-dev-455-ge96237f

Open Source Modelica Consortium

November 24, 2015

1	Introduction	3
1.1	System Overview	4
1.2	Interactive Session with Examples	5
1.3	Summary of Commands for the Interactive Session Handler	21
1.4	Running the compiler from command line	22
2	OMEdit – OpenModelica Connection Editor	25
2.1	Starting OMEdit	25
2.2	MainWindow & Browsers	26
2.3	Perspectives	31
2.4	Modeling a Model	31
2.5	Simulating a Model	36
2.6	Plotting the Simulation Results	37
2.7	Re-simulating a Model	38
2.8	How to Create User Defined Shapes – Icons	38
2.9	Settings	39
2.10	Debugger	42
3	2D Plotting	43
3.1	Example	43
3.2	Plotting Commands and their Options	44
4	Debugging	47
4.1	The Equation-based Debugger	47
4.2	The Algorithmic Debugger	49
5	OMNotebook with DrModelica and DrControl	55
5.1	Interactive Notebooks with Literate Programming	55
5.2	DrModelica Tutoring System – an Application of OMNotebook	55
5.3	DrControl Tutorial for Teaching Control Theory	61
5.4	OpenModelica Notebook Commands	68
5.5	References	76
6	Functional Mock-up Interface - FMI	77
6.1	FMI Export	77
6.2	FMI Import	78
7	Optimization with OpenModelica	81
7.1	Builtin Dynamic Optimization with OpenModelica and IpOpt	81
7.2	Compiling the Modelica code	81
7.3	An Example	81
7.4	Different Options for the Optimizer IPOPT	84
7.5	Dynamic Optimization with OpenModelica and CasADi	84
7.6	Parameter Sweep Optimization using OMOptim	86

8	MDT – The OpenModelica Development Tooling Eclipse Plugin	93
8.1	Introduction	93
8.2	Installation	93
8.3	Getting Started	94
9	MDT Debugger for Algorithmic Modelica	107
9.1	The Eclipse-based Debugger for Algorithmic Modelica	107
10	Modelica Performance Analyzer	115
10.1	Profiling information for ProfilingTest	116
10.2	Generated JSON for the Example	121
10.3	Using the Profiler from OMEdit	121
11	Modelica3D	125
11.1	Installing Modelica3D	125
11.2	Running Modelica3D	125
12	Simulation in Web Browser	127
13	Interoperability – C and Python	129
13.1	Calling External C functions	129
13.2	Calling external Python Code from a Modelica model	130
13.3	Calling OpenModelica from Python Code	131
14	OpenModelica Python Interface and PySimulator	135
14.1	OMPpython – OpenModelica Python Interface	135
14.2	PySimulator	137
15	Scripting API	139
15.1	OpenModelica Modelica Scripting Commands	139
15.2	OpenModelica API Calls	139
15.3	Examples	186
16	OpenModelica Compiler Flags	191
16.1	Options	191
16.2	Debug flags	201
16.3	Flags for Optimization Modules	205
17	Small Overview of Simulation Flags	207
17.1	OpenModelica (C-runtime) Simulation Flags	207
17.2	Integration Methods	211
18	Frequently Asked Questions (FAQ)	213
18.1	OpenModelica General	213
18.2	OMNotebook	213
18.3	OMDev - OpenModelica Development Environment	214
19	Major OpenModelica Releases	215
19.1	OpenModelica 1.9.3, September 2015	215
19.2	OpenModelica 1.9.2, February 2015	216
19.3	OpenModelica 1.9.1, October 2014	218
19.4	OpenModelica 1.9.0, October 2013	220
19.5	OpenModelica 1.8.1, March 2012	223
19.6	OpenModelica 1.8, November 2011	224
19.7	OpenModelica 1.7, April 2011	226
19.8	OpenModelica 1.6, November 2010	227
19.9	OpenModelica 1.5, July 2010	228
19.10	OpenModelica 1.4.5, January 2009	229
19.11	OpenModelica 1.4.4, Feb 2008	230
19.12	OpenModelica 1.4.3, June 2007	231

19.13	OpenModelica 1.4.2, October 2006	232
19.14	OpenModelica 1.4.1, June 2006	233
19.15	OpenModelica 1.4.0, May 2006	234
19.16	OpenModelica 1.3.1, November 2005	235
20	Contributors to OpenModelica	237
20.1	OpenModelica Contributors 2015	237
20.2	OpenModelica Contributors 2014	239
20.3	OpenModelica Contributors 2013	240
20.4	OpenModelica Contributors 2012	242
20.5	OpenModelica Contributors 2011	244
20.6	OpenModelica Contributors 2010	246
20.7	OpenModelica Contributors 2009	248
20.8	OpenModelica Contributors 2008	249
20.9	OpenModelica Contributors 2007	250
20.10	OpenModelica Contributors 2006	250
20.11	OpenModelica Contributors 2005	251
20.12	OpenModelica Contributors 2004	251
20.13	OpenModelica Contributors 2003	252
20.14	OpenModelica Contributors 2002	252
20.15	OpenModelica Contributors 2001	253
20.16	OpenModelica Contributors 2000	253
20.17	OpenModelica Contributors 1999	253
20.18	OpenModelica Contributors 1998	253
	Bibliography	255

Open Source Modelica Consortium

Copyright © 1998-*CurrentYear*, Open Source Modelica Consortium (OSMC), c/o Linköpings universitet, Department of Computer and Information Science, SE-58183 Linköping, Sweden

All rights reserved.

THIS PROGRAM IS PROVIDED UNDER THE TERMS OF GPL VERSION 3 LICENSE OR THIS OSMC PUBLIC LICENSE (OSMC-PL). ANY USE, REPRODUCTION OR DISTRIBUTION OF THIS PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THE OSMC PUBLIC LICENSE OR THE GPL VERSION 3, ACCORDING TO RECIPIENTS CHOICE.

The OpenModelica software and the OSMC (Open Source Modelica Consortium) Public License (OSMC-PL) are obtained from OSMC, either from the above address, from the URLs: <https://www.openmodelica.org> or <http://www.ida.liu.se/projects/OpenModelica>, and in the OpenModelica distribution. GNU version 3 is obtained from: <http://www.gnu.org/copyleft/gpl.html>.

This program is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE, EXCEPT AS EXPRESSLY SET FORTH IN THE BY RECIPIENT SELECTED SUBSIDIARY LICENSE CONDITIONS OF OSMC-PL.

See the full OSMC Public License conditions for more details.

This document is part of OpenModelica: <https://www.openmodelica.org>

Contact: OpenModelica@ida.liu.se

Modelica[®] is a registered trademark of the Modelica Association, <https://www.Modelica.org>

Mathematica[®] is a registered trademark of Wolfram Research Inc, <http://www.wolfram.com>

This users guide provides documentation and examples on how to use the OpenModelica system, both for the Modelica beginners and advanced users.

INTRODUCTION

The **OpenModelica** system described in this document has both short-term and long-term goals:

- **The short-term goal is to develop an efficient interactive** computational environment for the Modelica language, as well as a rather complete implementation of the language. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of both low level and high level numerical algorithms, e.g. for control system design, solving nonlinear equation systems, or to develop optimization algorithms that are applied to complex applications.
- **The longer-term goal is to have a complete reference implementation** of the Modelica language, including simulation of equation based models and additional facilities in the programming environment, as well as convenient facilities for research and experimentation in language design or other research activities. However, our goal is not to reach the level of performance and quality provided by current commercial Modelica environments that can handle large models requiring advanced analysis and optimization by the Modelica compiler.

The long-term *research* related goals and issues of the OpenModelica open source implementation of a Modelica environment include but are not limited to the following:

- **Development of a complete formal specification of Modelica**, including both static and dynamic semantics. Such a specification can be used to assist current and future Modelica implementers by providing a semantic reference, as a kind of reference implementation.
- **Language design, e.g. to further extend the scope of the** language, e.g. for use in diagnosis, structural analysis, system identification, etc., as well as modeling problems that require extensions such as partial differential equations, enlarged scope for discrete modeling and simulation, etc.
- **Language design to improve abstract properties such as** expressiveness, orthogonality, declarativity, reuse, configurability, architectural properties, etc.
- **Improved implementation techniques, e.g. to enhance the performance** of compiled Modelica code by generating code for parallel hardware.
- **Improved debugging support for equation based languages such as** Modelica, to make them even easier to use.
- **Easy-to-use specialized high-level (graphical) user interfaces** for certain application domains.
- **Visualization and animation techniques for interpretation and** presentation of results.
- **Application usage and model library development by researchers in** various application areas.

The OpenModelica environment provides a test bench for language design ideas that, if successful, can be submitted to the Modelica Association for consideration regarding possible inclusion in the official Modelica standard.

The current version of the OpenModelica environment allows most of the expression, algorithm, and function parts of Modelica to be executed interactively, as well as equation models and Modelica functions to be compiled into efficient C code. The generated C code is combined with a library of utility functions, a run-time library, and a numerical DAE solver.

1.1 System Overview

The OpenModelica environment consists of several interconnected subsystems, as depicted in Figure 1.1.

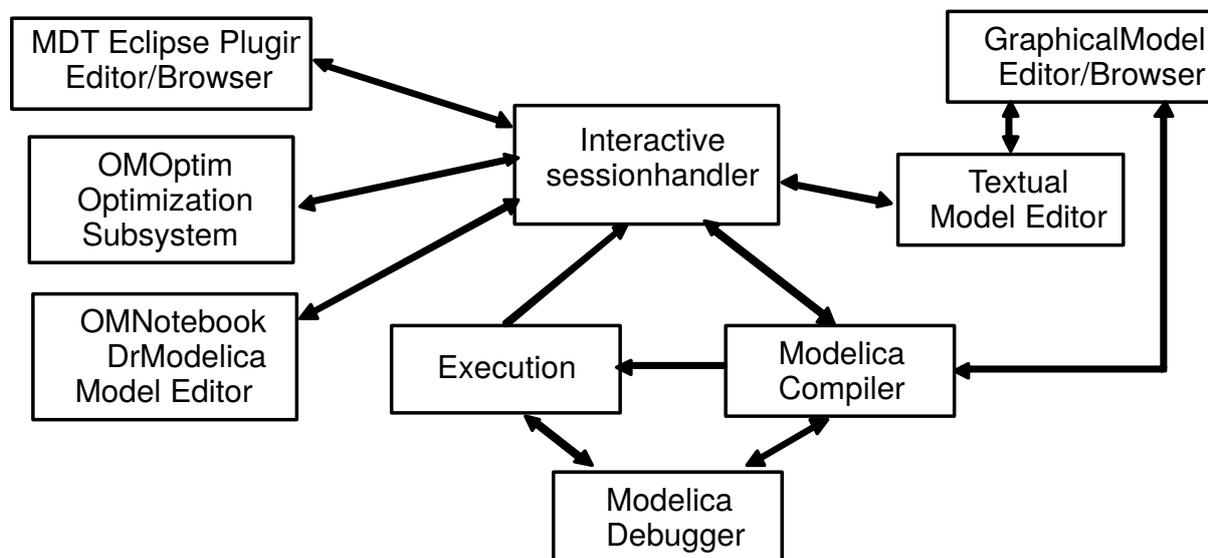


Figure 1.1: The architecture of the OpenModelica environment. Arrows denote data and control flow. The interactive session handler receives commands and shows results from evaluating commands and expressions that are translated and executed. Several subsystems provide different forms of browsing and textual editing of Modelica code. The debugger currently provides debugging of an extended algorithmic subset of Modelica.

The following subsystems are currently integrated in the OpenModelica environment:

- **An interactive session handler, that parses and interprets commands** and Modelica expressions for evaluation, simulation, plotting, etc. The session handler also contains simple history facilities, and completion of file names and certain identifiers in commands.
- **A Modelica compiler subsystem, translating Modelica to C code, with** a symbol table containing definitions of classes, functions, and variables. Such definitions can be predefined, user-defined, or obtained from libraries. The compiler also includes a Modelica interpreter for interactive usage and constant expression evaluation. The subsystem also includes facilities for building simulation executables linked with selected numerical ODE or DAE solvers.
- **An execution and run-time module. This module currently executes** compiled binary code from translated expressions and functions, as well as simulation code from equation based models, linked with numerical solvers. In the near future event handling facilities will be included for the discrete and hybrid parts of the Modelica language.
- **Eclipse plugin editor/browser. The Eclipse plugin called MDT** (Modelica Development Tooling) provides file and class hierarchy browsing and text editing capabilities, rather analogous to previously described Emacs editor/browser. Some syntax highlighting facilities are also included. The Eclipse framework has the advantage of making it easier to add future extensions such as refactoring and cross referencing support.
- **OMNotebook DrModelica model editor. This subsystem provides a** lightweight notebook editor, compared to the more advanced Mathematica notebooks available in MathModelica. This basic functionality still allows essentially the whole DrModelica tutorial to be handled. Hierarchical text documents with chapters and sections can be represented and edited, including basic formatting. Cells can contain ordinary text or Modelica models and expressions, which can be evaluated and simulated. However, no mathematical typesetting facilities are yet available in the cells of this notebook editor.
- **Graphical model editor/browser OMEdit. This is a graphical** connection editor, for component based model design by connecting instances of Modelica classes, and browsing Modelica model libraries

for reading and picking component models. The graphical model editor also includes a textual editor for editing model class definitions, and a window for interactive Modelica command evaluation.

- **Optimization subsystem *OMOptim*.** This is an optimization subsystem for OpenModelica, currently for design optimization choosing an optimal set of design parameters for a model. The current version has a graphical user interface, provides genetic optimization algorithms and Pareto front optimization, works integrated with the simulators and automatically accesses variables and design parameters from the Modelica model.
- **Dynamic Optimization subsystem.** This is dynamic optimization using collocation methods, for Modelica models extended with optimization specifications with goal functions and additional constraints. This subsystem is integrated with in the OpenModelica compiler.
- **Modelica equation model debugger.** The equation model debugger shows the location of an error in the model equation source code. It keeps track of the symbolic transformations done by the compiler on the way from equations to low-level generated C code, and also explains which transformations have been done.
- **Modelica algorithmic code debugger.** The algorithmic code Modelica debugger provides debugging for an extended algorithmic subset of Modelica, excluding equation-based models and some other features, but including some meta-programming and model transformation extensions to Modelica. This is a conventional full-feature debugger, using Eclipse for displaying the source code during stepping, setting breakpoints, etc. Various back-trace and inspection commands are available. The debugger also includes a data-view browser for browsing hierarchical data such as tree- or list structures in extended Modelica.

1.2 Interactive Session with Examples

The following is an interactive session using the interactive session handler in the OpenModelica environment, called OMShell – the OpenModelica Shell). Most of these examples are also available in the *OMNotebook with DrModelica and DrControl* UsersGuideExamples.onb as well as the testmodels in:

```
>>> getInstallationDirectoryPath() + "/share/doc/omc/testmodels/"
"«OPENMODELICAHOME»/share/doc/omc/testmodels/"
```

The following commands were run using OpenModelica version:

```
>>> getVersion()
"v1.9.4-dev.608+g7458c24"
```

1.2.1 Starting the Interactive Session

The Windows version which at installation is made available in the start menu as OpenModelica->OpenModelica Shell which responds with an interaction window:

We enter an assignment of a vector expression, created by the range construction expression 1:12, to be stored in the variable x. The value of the expression is returned.

```
>>> x := 1:12
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

1.2.2 Using the Interactive Mode

When running OMC in interactive mode (for instance using OMShell) one can make load classes and execute commands. Here we give a few example sessions.

Example Session 1

To get help on using OMSHELL and OpenModelica, type “help()” and press enter.

```
>>> model A Integer t = 1.5; end A; //The type is Integer but 1.5 is of Real Type
{A}
>>> instantiateModel(A)
""
"[<interactive>:1:9-1:23:writable] Error: Type mismatch in binding t = 1.5, expected subtype of Integer
Error: Error occurred while flattening model A
"
```

Example Session 2

To get help on using OMSHELL and OpenModelica, type “help()” and press enter.

```
model C
  Integer a;
  Real b;
equation
  der(a) = b;
  der(b) = 12.0;
end C;
```

```
>>> instantiateModel(C)
""
```

Error:

```
[<interactive>:5:3-5:13:writable] Error: Argument 'a' to der has illegal type Integer, must be a subtype of Real.
Error: Error occurred while flattening model C
```

1.2.3 Trying the Bubblesort Function

Load the function bubblesort, either by using the pull-down menu File->Load Model, or by explicitly giving the command:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/bubblesort.mo")
true
```

The function bubblesort is called below to sort the vector x in descending order. The sorted result is returned together with its type. Note that the result vector is of type Real[:], instantiated as Real[12], since this is the declared type of the function result. The input Integer vector was automatically converted to a Real vector according to the Modelica type coercion rules. The function is automatically compiled when called if this has not been done before.

```
>>> bubblesort(x)
{12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0}
```

Another call:

```
>>> bubblesort({4, 6, 2, 5, 8})
{8.0, 6.0, 5.0, 4.0, 2.0}
```

1.2.4 Trying the system and cd Commands

It is also possible to give operating system commands via the system utility function. A command is provided as a string argument. The example below shows the system utility applied to the UNIX command cat, which here outputs the contents of the file bubblesort.mo to the output stream when running omc from the command-line.

```
>>> system("cat '"+getInstallationDirectoryPath()+"/share/doc/omc/testmodels/bubblesort.mo' > bubblesort.mo")
0
```

```
function bubblesort
  input Real[:] x;
  output Real[size(x,1)] y;
protected
  Real t;
algorithm
  y := x;
  for i in 1:size(x,1) loop
    for j in 1:size(x,1) loop
      if y[i] > y[j] then
        t := y[i];
        y[i] := y[j];
        y[j] := t;
      end if;
    end for;
  end for;
end bubblesort;
```

Note: The output emitted into stdout by system commands is put into log-files when running the CORBA-based clients, not into the visible GUI windows. Thus the text emitted by the above cat command would not be returned, which is why it is redirected to another file.

A better way to read the content of files would be the readFile command:

```
>>> readFile("bubblesort.mo")
function bubblesort
  input Real[:] x;
  output Real[size(x,1)] y;
protected
  Real t;
algorithm
  y := x;
  for i in 1:size(x,1) loop
    for j in 1:size(x,1) loop
      if y[i] > y[j] then
        t := y[i];
        y[i] := y[j];
        y[j] := t;
      end if;
    end for;
  end for;
end bubblesort;
```

The system command only returns a success code (0 = success).

```
>>> system("dir")
0
>>> system("Non-existing command")
127
```

Another built-in command is cd, the *change current directory* command. The resulting current directory is returned as a string.

```
>>> dir:=cd()
"«DOCHOME»"
>>> cd("source")
"«DOCHOME»/source"
>>> cd(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/")
"«OPENMODELICAHOME»/share/doc/omc/testmodels"
>>> cd(dir)
"«DOCHOME»"
```

1.2.5 Modelica Library and DCMotor Model

We load a model, here the whole Modelica standard library, which also can be done through the File->Load Modelica Library menu item:

```
>>> loadModel(Modelica)
true
```

We also load a file containing the dcmotor model:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/dcmotor.mo")
true
```

It is simulated:

```
>>> simulate(dcmotor, startTime=0.0, stopTime=10.0)
record SimulationResult
  resultFile = "«DOCHOME»/dcmotor_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.291050725,
  timeBackend = 0.010876333,
  timeSimCode = 0.058119349,
  timeTemplates = 0.008925807000000001,
  timeCompile = 0.234732263,
  timeSimulation = 0.014280248,
  timeTotal = 0.618100886
end SimulationResult;
```

Warning:

[«OPENMODELICAHOME»/lib/omlibrary/Modelica 3.2.1/Electrical/Analog/Basic.mo:838:5-839:35:writable] Warning: Parameter emf1.k has no value, and is fixed during initialization (fixed=true), using available start value (start=1.0) as default value.

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

We list the source code of the model:

```
>>> list(dcmotor)
model dcmotor
  Modelica.Electrical.Analog.Basic.Resistor resistor1(R = 10);
  //Observe the difference between MSL 2.2 and 3.1 regarding the default values, in 3.1 there are
  Modelica.Electrical.Analog.Basic.Inductor inductor1(L = 0.2);
  Modelica.Electrical.Analog.Basic.Ground ground1;
  Modelica.Mechanics.Rotational.Components.Inertia load(J = 1);
  // Modelica version 3.1
  // Modelica.Mechanics.Rotational.Inertia load(J = 1); // Modelica version 2.2
  Modelica.Electrical.Analog.Basic.EMF emf1;
  Modelica.Blocks.Sources.Step step1;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1;
equation
  //connect(step1.outport, signalVoltage1.inPort);
  connect(step1.y, signalVoltage1.v);
  connect(signalVoltage1.p, resistor1.p);
  connect(resistor1.n, inductor1.p);
  connect(inductor1.n, emf1.p);
  // connect(emf1.flange_b, load.flange_a); //Modelica version 2.2
  connect(emf1.flange, load.flange_a);
  // Modelica version 3.1
  connect(signalVoltage1.n, ground1.p);
  connect(ground1.p, emf1.n);
end dcmotor;
```

We test code instantiation of the model to flat code:

```

>>> instantiateModel(dcmotor)
class dcmotor
  Real resistor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop between the two pins";
  Real resistor1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from pin p to pin n";
  Real resistor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real resistor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real resistor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real resistor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  parameter Boolean resistor1.useHeatPort = false "=true, if HeatPort is enabled";
  parameter Real resistor1.T(quantity = "ThermodynamicTemperature", unit = "K", displayUnit = "deg");
  Real resistor1.LossPower(quantity = "Power", unit = "W") "Loss power leaving component via HeatPort";
  Real resistor1.T_heatPort(quantity = "ThermodynamicTemperature", unit = "K", displayUnit = "deg");
  parameter Real resistor1.R(quantity = "Resistance", unit = "Ohm", start = 1.0) = 10.0 "Resistance";
  parameter Real resistor1.T_ref(quantity = "ThermodynamicTemperature", unit = "K", displayUnit = "deg");
  parameter Real resistor1.alpha(quantity = "LinearTemperatureCoefficient", unit = "1/K") = 0.0 "Linear temperature coefficient";
  Real resistor1.R_actual(quantity = "Resistance", unit = "Ohm") "Actual resistance = R*(1 + alpha*(T - T_ref))";
  Real inductor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop between the two pins";
  Real inductor1.i(quantity = "ElectricCurrent", unit = "A", start = 0.0) "Current flowing from pin p to pin n";
  Real inductor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real inductor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real inductor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real inductor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  parameter Real inductor1.L(quantity = "Inductance", unit = "H", start = 1.0) = 0.2 "Inductance";
  Real ground1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real ground1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real load.flange_a.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Absolute rotation of flange a";
  Real load.flange_a.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange";
  Real load.flange_b.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Absolute rotation of flange b";
  Real load.flange_b.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange";
  parameter Real load.J(quantity = "MomentOfInertia", unit = "kg.m2", min = 0.0, start = 1.0) = 1.0 "Moment of inertia";
  parameter enumeration (never, avoid, default, prefer, always) load.stateSelect = StateSelect.default;
  Real load.phi(quantity = "Angle", unit = "rad", displayUnit = "deg", stateSelect = StateSelect.default);
  Real load.w(quantity = "AngularVelocity", unit = "rad/s", stateSelect = StateSelect.default);
  Real load.a(quantity = "AngularAcceleration", unit = "rad/s2");
  parameter Boolean emf1.useSupport = false "= true, if support flange enabled, otherwise implicitly false";
  parameter Real emf1.k(quantity = "ElectricalTorqueConstant", unit = "N.m/A", start = 1.0) "Electrical torque constant";
  Real emf1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop between the two pins";
  Real emf1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from positive to negative terminal";
  Real emf1.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Angle of shaft flange with respect to support flange";
  Real emf1.w(quantity = "AngularVelocity", unit = "rad/s") "Angular velocity of flange relative to support flange";
  Real emf1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real emf1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real emf1.flange.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Absolute rotation of flange";
  Real emf1.flange.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange";
  protected Real emf1.internalSupport.tau(quantity = "Torque", unit = "N.m") = -emf1.flange.tau;
  protected Real emf1.internalSupport.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") = emf1.flange.phi;
  protected Real emf1.internalSupport.flange.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") = emf1.flange.phi;
  protected Real emf1.internalSupport.flange.tau(quantity = "Torque", unit = "N.m") = emf1.flange.tau;
  protected parameter Real emf1.fixed.phi0(quantity = "Angle", unit = "rad", displayUnit = "deg") = 0.0;
  protected Real emf1.fixed.flange.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") = emf1.flange.phi - emf1.fixed.phi0;
  protected Real emf1.fixed.flange.tau(quantity = "Torque", unit = "N.m") = emf1.flange.tau;
  Real step1.y "Connector of Real output signal";
  parameter Real step1.offset = 0.0 "Offset of output signal y";
  parameter Real step1.startTime(quantity = "Time", unit = "s") = 0.0 "Output y = offset for time >= startTime";
  parameter Real step1.height = 1.0 "Height of step";
  Real signalVoltage1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real signalVoltage1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real signalVoltage1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real signalVoltage1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
  Real signalVoltage1.v(unit = "V") "Voltage between pin p and n (= p.v - n.v) as input signal";
  Real signalVoltage1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from pin p to pin n";

```

equation

```

assert(1.0 + resistor1.alpha * (resistor1.T_heatPort - resistor1.T_ref) >= 1e-15, "Temperature c
resistor1.R_actual = resistor1.R * (1.0 + resistor1.alpha * (resistor1.T_heatPort - resistor1.T
resistor1.v = resistor1.R_actual * resistor1.i;
resistor1.LossPower = resistor1.v * resistor1.i;
resistor1.v = resistor1.p.v - resistor1.n.v;
0.0 = resistor1.p.i + resistor1.n.i;
resistor1.i = resistor1.p.i;
resistor1.T_heatPort = resistor1.T;
inductor1.L * der(inductor1.i) = inductor1.v;
inductor1.v = inductor1.p.v - inductor1.n.v;
0.0 = inductor1.p.i + inductor1.n.i;
inductor1.i = inductor1.p.i;
ground1.p.v = 0.0;
load.phi = load.flange_a.phi;
load.phi = load.flange_b.phi;
load.w = der(load.phi);
load.a = der(load.w);
load.J * load.a = load.flange_a.tau + load.flange_b.tau;
emf1.internalSupport.flange.tau = emf1.internalSupport.tau;
emf1.internalSupport.flange.phi = emf1.internalSupport.phi;
emf1.fixed.flange.phi = emf1.fixed.phi0;
emf1.v = emf1.p.v - emf1.n.v;
0.0 = emf1.p.i + emf1.n.i;
emf1.i = emf1.p.i;
emf1.phi = emf1.flange.phi - emf1.internalSupport.phi;
emf1.w = der(emf1.phi);
emf1.k * emf1.w = emf1.v;
emf1.flange.tau = (-emf1.k) * emf1.i;
step1.y = step1.offset + (if time < step1.startTime then 0.0 else step1.height);
signalVoltage1.v = signalVoltage1.p.v - signalVoltage1.n.v;
0.0 = signalVoltage1.p.i + signalVoltage1.n.i;
signalVoltage1.i = signalVoltage1.p.i;
signalVoltage1.p.i + resistor1.p.i = 0.0;
resistor1.n.i + inductor1.p.i = 0.0;
emf1.p.i + inductor1.n.i = 0.0;
signalVoltage1.n.i + ground1.p.i + emf1.n.i = 0.0;
load.flange_a.tau + emf1.flange.tau = 0.0;
load.flange_b.tau = 0.0;
emf1.internalSupport.flange.tau + emf1.fixed.flange.tau = 0.0;
emf1.fixed.flange.phi = emf1.internalSupport.flange.phi;
signalVoltage1.v = step1.y;
resistor1.p.v = signalVoltage1.p.v;
inductor1.p.v = resistor1.n.v;
emf1.p.v = inductor1.n.v;
emf1.flange.phi = load.flange_a.phi;
emf1.n.v = ground1.p.v;
emf1.n.v = signalVoltage1.n.v;
end dcmotor;

```

We plot part of the simulated result:

1.2.6 The val() function

The val(*variableName,time*) scripction function can be used to retrieve the interpolated value of a simulation result variable at a certain point in the simulation time, see usage in the BouncingBall simulation below.

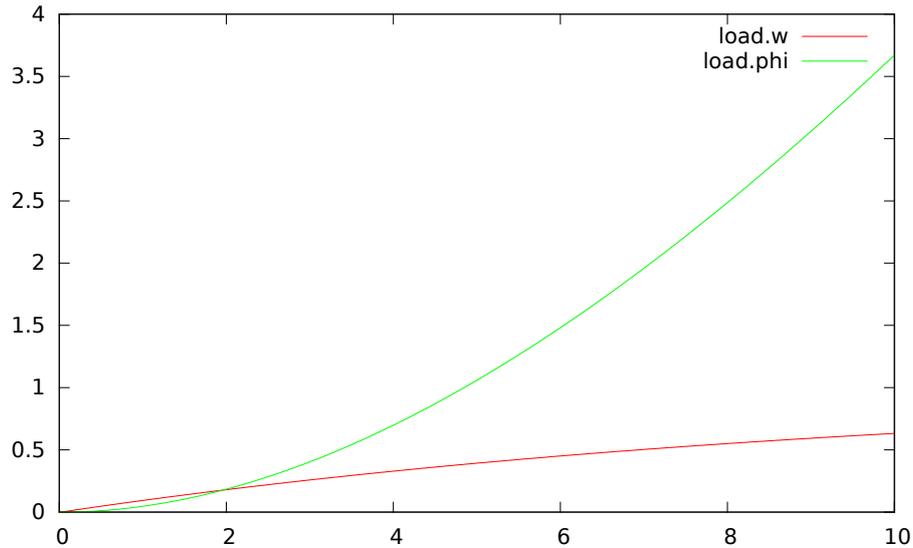


Figure 1.2: Rotation and rotational velocity of the DC motor

1.2.7 BouncingBall and Switch Models

We load and simulate the BouncingBall example containing when-equations and if-expressions (the Modelica keywords have been bold-faced by hand for better readability):

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.mo")
true
```

```
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(start = 1) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new;
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then -e * pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

Instead of just giving a simulate and plot command, we perform a runScript command on a .mos (Modelica script) file sim_BouncingBall.mos that contains these commands:

```
>>> writeFile("sim_BouncingBall.mos", "
  loadFile(getInstallationDirectoryPath() + \" /share/doc/omc/testmodels/BouncingBall.mo\");
  simulate(BouncingBall, stopTime=3.0);
  /* plot({h,flying}); */
")
true
>>> runScript("sim_BouncingBall.mos")
```

```
"true
record SimulationResult
  resultFile = "\"«DOCHOME»/BouncingBall_res.mat\"",
  simulationOptions = "\"startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500, tolerance = 1e-6\"",
  messages = "\"\"",
  timeFrontend = 0.006101118,
  timeBackend = 0.003530612,
  timeSimCode = 0.056958106,
  timeTemplates = 0.0055774790000000001,
  timeCompile = 0.234345326,
  timeSimulation = 0.01029387,
  timeTotal = 0.316891615
end SimulationResult;
"
```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

```
model Switch
  Real v;
  Real i;
  Real il;
  Real itot;
  Boolean open;
equation
  itot = i + il;
  if open then
    v = 0;
  else
    i = 0;
  end if;
  1 - il = 0;
  1 - v - i = 0;
  open = time >= 0.5;
end Switch;
```

```
>>> simulate(Switch, startTime=0, stopTime=1)
record SimulationResult
  resultFile = "«DOCHOME»/Switch_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.006350226,
  timeBackend = 0.0081955800000000001,
  timeSimCode = 0.048298599,
  timeTemplates = 0.004390284,
  timeCompile = 0.225927601,
  timeSimulation = 0.0080811090000000001,
  timeTotal = 0.301315199
end SimulationResult;
```

Retrieve the value of itot at time=0 using the val(variableName, time) function:

```
>>> val(itot,0)
1.0
```

Plot itot and open:

We note that the variable open switches from false (0) to true (1), causing itot to increase from 1.0 to 2.0.

1.2.8 Clear All Models

Now, first clear all loaded libraries and models:

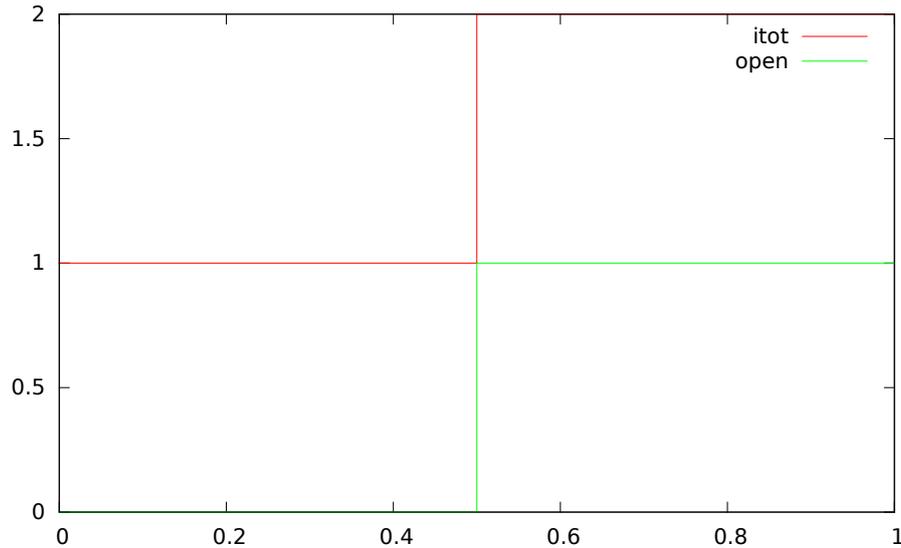


Figure 1.3: Plot when the switch opens

```
>>> clear()
true
```

List the loaded models – nothing left:

```
>>> list()
""
```

1.2.9 VanDerPol Model and Parametric Plot

We load another model, the VanDerPol model (or via the menu File->Load Model):

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/VanDerPol.mo")
true
```

It is simulated:

```
>>> simulate(VanDerPol, stopTime=80)
record SimulationResult
  resultFile = "«DOCHOME»/VanDerPol_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 80.0, numberOfIntervals = 500, tolerance = 1e-06",
  messages = "",
  timeFrontend = 0.009595224000000001,
  timeBackend = 0.001890452,
  timeSimCode = 0.050179687,
  timeTemplates = 0.00327367,
  timeCompile = 0.208703647,
  timeSimulation = 0.01010295,
  timeTotal = 0.283849972
end SimulationResult;
```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

It is plotted:

```
>>> plotParametric("x", "y")
```

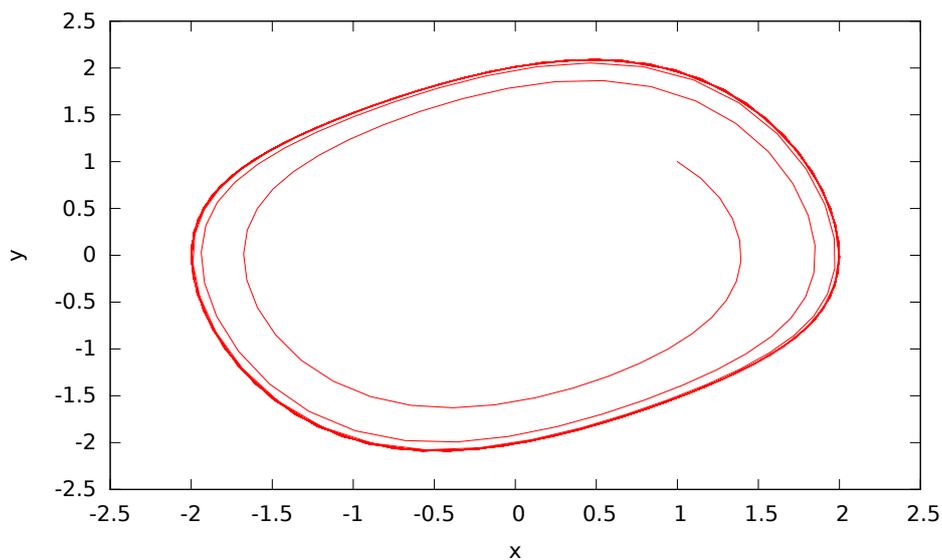


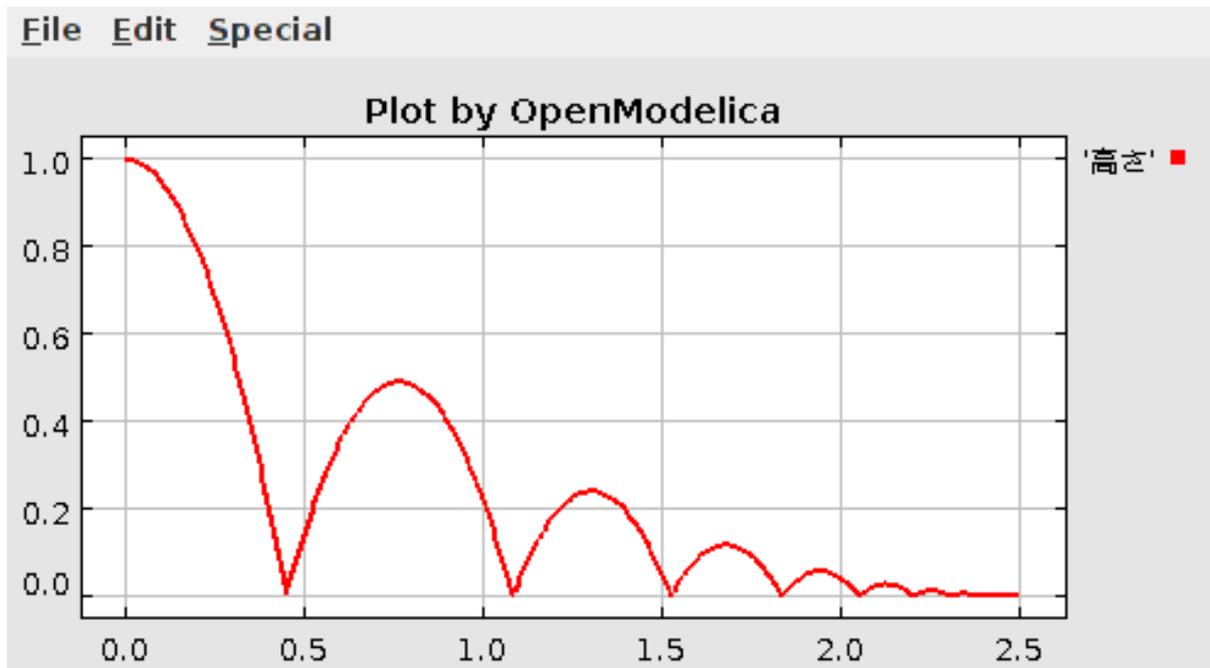
Figure 1.4: VanDerPol plotParametric(x,y)

Perform code instantiation to flat form of the VanDerPol model:

```
>>> instantiateModel(VanDerPol)
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1.0);
  Real y(start = 1.0);
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = lambda * (1.0 - x ^ 2.0) * y - x;
end VanDerPol;
```

1.2.10 Using Japanese or Chinese Characters

Japanese, Chinese, and other kinds of UniCode characters can be used within quoted (single quote) identifiers, see for example the variable name to the right in the plot below:



1.2.11 Scripting with For-Loops, While-Loops, and If-Statements

A simple summing integer loop (using multi-line input without evaluation at each line into OMSHELL requires copy-paste as one operation from another document):

```
>>> k := 0;
>>> for i in 1:1000 loop
  k := k + i;
end for;
>>> k
500500
```

A nested loop summing reals and integers:

```
>>> g := 0.0;
>>> h := 5;
>>> for i in {23.0, 77.12, 88.23} loop
  for j in i:0.5:(i+1) loop
    g := g + j;
    g := g + h / 2;
  end for;
  h := h + g;
end for;
```

By putting two (or more) variables or assignment statements separated by semicolon(s), ending with a variable, one can observe more than one variable value:

```
>>> h; g
1997.45
1479.09
```

A for-loop with vector traversal and concatenation of string elements:

```
>>> i:="";
>>> lst := {"Here ", "are ", "some ", "strings."};
>>> s := "";
>>> for i in lst loop
  s := s + i;
end for;
```

```
>>> s
"Here are some strings."
```

Normal while-loop with concatenation of 10 “abc ” strings:

```
>>> s:="";
>>> i:=1;
>>> while i<=10 loop
  s:="abc "+s;
  i:=i+1;
end while;
>>> s
"abc abc abc abc abc abc abc abc abc abc "
```

A simple if-statement. By putting the variable last, after the semicolon, its value is returned after evaluation:

```
>>> if 5>2 then a := 77; end if; a
77
```

An if-then-else statement with elseif:

```
>>> if false then
  a := 5;
elseif a > 50 then
  b:= "test"; a:= 100;
else
  a:=34;
end if;
```

Take a look at the variables a and b:

```
>>> a;b
100
"test"
```

1.2.12 Variables, Functions, and Types of Variables

Assign a vector to a variable:

```
>>> a:=1:5
{1,2,3,4,5}
```

Type in a function:

```
function mySqr
  input Real x;
  output Real y;
algorithm
  y:=x*x;
end mySqr;
```

Call the function:

```
>>> b:=mySqr(2)
4.0
```

Look at the value of variable a:

```
>>> a
{1,2,3,4,5}
```

Look at the type of a:

```
>>> typeof(a)
"Integer[5]"
```

Retrieve the type of b:

```
>>> typeof(b)
"Real"
```

What is the type of mySqr? Cannot currently be handled.

```
>>> typeof(mySqr)
```

List the available variables:

```
>>> listVariables()
{b, a, s, lst, i, h, g, k, currentSimulationResult}
```

Clear again:

```
>>> clear()
true
```

1.2.13 Getting Information about Error Cause

Call the function `getErrorString()` in order to get more information about the error cause after a simulation failure:

```
>>> getErrorString()
""
```

1.2.14 Alternative Simulation Output Formats

There are several output format possibilities, with `mat` being the default. `plt` and `mat` are the only formats that allow you to use the `val()` or `plot()` functions after a simulation. Compared to the speed of `plt`, `mat` is roughly 5 times faster for small files, and scales better for larger files due to being a binary format. The `csv` format is roughly twice as fast as `plt` on data-heavy simulations. The `plt` format allocates all output data in RAM during simulation, which means that simulations may fail due to applications only being able to address 4GB of memory on 32-bit platforms. `Empty` does no output at all and should be by far the fastest. The `csv` and `plt` formats are suitable when using external scripts or tools like `gnuplot` to generate plots or process data. The `mat` format can be post-processed in `MATLAB` or `Octave`.

```
>>> simulate(... , outputFormat="mat")
>>> simulate(... , outputFormat="csv")
>>> simulate(... , outputFormat="plt")
>>> simulate(... , outputFormat="empty")
```

It is also possible to specify which variables should be present in the result-file. This is done by using [POSIX Extended Regular Expressions](#). The given expression must match the full variable name (^ and \$ symbols are automatically added to the given regular expression).

```
// Default, match everything
```

```
>>> simulate(... , variableFilter=".*")
```

```
// match indices of variable myVar that only contain the numbers using combinations
```

```
// of the letters 1 through 3
```

```
>>> simulate(... , variableFilter="myVar\\[[1-3]*\\]")
```

```
// match x or y or z
```

```
>>> simulate(... , variableFilter="x|y|z")
```

1.2.15 Using External Functions

See Chapter *Interoperability – C and Python* for more information about calling functions in other programming languages.

1.2.16 Using Parallel Simulation via OpenMP Multi-Core Support

Faster simulations on multi-core computers can be obtained by using a new OpenModelica feature that automatically partitions the system of equations and schedules the parts for execution on different cores using shared-memory OpenMP based execution. The speedup obtained is dependent on the model structure, whether the system of equations can be partitioned well. This version in the current OpenModelica release is an experimental version without load balancing. The following command, not yet available from the OpenModelica GUI, will run a parallel simulation on a model:

```
>>> omc +d=openmp model.mo
```

1.2.17 Loading Specific Library Version

There exist many different versions of Modelica libraries which are not compatible. It is possible to keep multiple versions of the same library stored in the directory given by calling `getModelicaPath()`. By calling `loadModel(Modelica,{"3.2"})`, OpenModelica will search for a directory called "Modelica 3.2" or a file called "Modelica 3.2.mo". It is possible to give several library versions to search for, giving preference for a pre-release version of a library if it is installed. If the searched version is "default", the priority is: no version name (Modelica), main release version (Modelica 3.1), pre-release version (Modelica 3.1Beta 1) and unordered versions (Modelica Special Release).

The `loadModel` command will also look at the `uses` annotation of the top-level class after it has been loaded. Given the following package, `Complex 1.0` and `ModelicaServices 1.1` will also be loaded into the AST automatically.

```
package Modelica
  annotation (uses (Complex (version="1.0"),
    ModelicaServices (version="1.1")));
end Modelica;
```

```
>>> clear()
true
```

Packages will also be loaded if a model has a `uses`-annotation:

```
model M
  annotation (uses (Modelica (version="3.2.1")));
end M;
```

```
>>> instantiateModel (M)
class M
end M;
```

Note:

Notification: Automatically loaded package `Modelica 3.2.1` due to `uses` annotation.

Notification: Automatically loaded package `Complex 3.2.1` due to `uses` annotation.

Notification: Automatically loaded package `ModelicaServices 3.2.1` due to `uses` annotation.

Packages will also be loaded by looking at the first identifier in the path:

```
>>> instantiateModel(Modelica.Electrical.Analog.Basic.Ground)
class Modelica.Electrical.Analog.Basic.Ground "Ground node"
  Real p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
equation
  p.v = 0.0;
  p.i = 0.0;
end Modelica.Electrical.Analog.Basic.Ground;
```

Note:

Notification: Automatically loaded package Complex 3.2.1 due to uses annotation.

Notification: Automatically loaded package ModelicaServices 3.2.1 due to uses annotation.

Notification: Automatically loaded package Modelica default due to uses annotation.

1.2.18 Calling the Model Query and Manipulation API

In the OpenModelica System Documentation, an external API (application programming interface) is described which returns information about models and/or allows manipulation of models. Calls to these functions can be done interactively as below, but more typically by program clients to the OpenModelica Compiler (OMC) server. Current examples of such clients are the OpenModelica MDT Eclipse plugin, OMNotebook, the OMEdit graphic model editor, etc. This API is untyped for performance reasons, i.e., no type checking and minimal error checking is done on the calls. The results of a call is returned as a text string in Modelica syntax form, which the client has to parse. An example parser in C++ is available in the OMNotebook source code, whereas another example parser in Java is available in the MDT Eclipse plugin.

Below we show a few calls on the previously simulated BouncingBall model. The full documentation on this API is available in the system documentation. First we load and list the model again to show its structure:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.mo");
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(start = 1) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new;
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then -e * pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

Different kinds of calls with returned results:

```
>>> getClassRestriction(BouncingBall)
"model"
>>> getClassInformation(BouncingBall)
("model", "", false, false, false, "«OPENMODELICAHOME»/share/doc/omc/testmodels/BouncingBall.mo", false)
>>> isFunction(BouncingBall)
false
```

```

>>> existClass(BouncingBall)
true
>>> getComponents(BouncingBall)
{{Real,e,"coefficient of restitution", "public", false, false, false, false, "parameter", "none",
>>> getConnectionCount(BouncingBall)
0
>>> getInheritanceCount(BouncingBall)
0
>>> getComponentModifierValue(BouncingBall,e)
0.7
>>> getComponentModifierNames(BouncingBall,e)

>>> getClassRestriction(BouncingBall)
"model"
>>> getVersion() // Version of the currently running OMC
"v1.9.4-dev.608+g7458c24"

```

Error:

```
[<interactive>:1:1-1:0:writable] Error: Class getComponentModifierNames not found in scope <global scope>
(looking for a function or record).
```

1.2.19 Quit OpenModelica

Leave and quit OpenModelica:

```
>>> quit()
```

1.2.20 Dump XML Representation

The command `dumpXMLDAE` dumps an XML representation of a model, according to several optional parameters.

```
dumpXMLDAE(modelname[,asInSimulationCode=<Boolean>] [filePrefix=<String>] [storeInTemp=<Boolean>] [,addMathMLCode =<Boolean>])
```

This command dumps the mathematical representation of a model using an XML representation, with optional parameters. In particular, `asInSimulationCode` defines where to stop in the translation process (before dumping the model), the other options are relative to the file storage: `filePrefix` for specifying a different name and `storeInTemp` to use the temporary directory. The optional parameter `addMathMLCode` gives the possibility to don't print the MathML code within the xml file, to make it more readable. Usage is trivial, just: `addMathMLCode=true/false` (default value is false).

1.2.21 Dump Matlab Representation

The command `exportDAEtoMatlab` dumps an XML representation of a model, according to several optional parameters.

```
exportDAEtoMatlab(modelname);
```

This command dumps the mathematical representation of a model using a Matlab representation. Example:

```

>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.mo")
true
>>> exportDAEtoMatlab(BouncingBall)
"The equation system was dumped to Matlab file:BouncingBall_imatrix.m"

```

```

% Incidence Matrix
% =====
% number of rows: 6

```

```

IM={{3,6},{1,{'if', 'true', '==' {3},{},}},{{'if', 'true', '==' {4},{},}},{5},{2,{'if', 'edge(impact
VL = {'foo','v_new','impact','flying','v','h'};

EqStr = {'impact = h <= 0.0;', 'foo = if impact then 1 else 2;', 'der(v) = if flying then -g else 0

OldEqStr={'class BouncingBall', ' parameter Real e = 0.7 "coefficient of restitution";', ' paramete

```

1.3 Summary of Commands for the Interactive Session Handler

The following is the complete list of commands currently available in the interactive session handler.

`simulate(modelname)` Translate a model named *modelname* and simulate it.

```

simulate(modelname[,startTime=<Real>][,stopTime=<Real>][,numberOfIntervals
=<Integer>][,outputInterval=<Real>][,method=<String>]
[,tolerance=<Real>][,fixedStepSize=<Real>]

```

[,outputFormat=<String>]) Translate and simulate a model, with optional start time, stop time, and optional number of simulation intervals or steps for which the simulation results will be computed. More intervals will give higher time resolution, but occupy more space and take longer to compute. The default number of intervals is 500. It is possible to choose solving method, default is “dassl”, “euler” and “rungekutta” are also available. Output format “mat” is default. “plt” and “mat” (MATLAB) are the only ones that work with the `val()` command, “csv” (comma separated values) and “empty” (no output) are also available (see section *Alternative Simulation Output Formats*).

`plot(vars)` Plot the variables given as a vector or a scalar, e.g. `plot({x1,x2})` or `plot(x1)`.

`plotParametric(var1, var2)` Plot *var2* relative to *var1* from the most recently simulated model, e.g. `plotParametric(x,y)`.

`cd()` Return the current directory.

`cd(dir)` Change directory to the directory given as string.

`clear()` Clear all loaded definitions.

`clearVariables()` Clear all defined variables.

`dumpXMLDAE(modelname, ...)` Dumps an XML representation of a model, according to several optional parameters.

`exportDAEtoMatlab(name)` Dumps a Matlab representation of a model.

`instantiateModel(modelname)` Performs code instantiation of a model/class and return a string containing the flat class definition.

`list()` Return a string containing all loaded class definitions.

`list(modelname)` Return a string containing the class definition of the named class.

`listVariables()` Return a vector of the names of the currently defined variables.

`loadModel(classname)` Load model or package of name *classname* from the path indicated by the environment variable `OPENMODELICALIBRARY`.

`loadFile(str)` Load Modelica file (.mo) with name given as string argument *str*.

`readFile(str)` Load file given as string *str* and return a string containing the file content.

`runScript(str)` Execute script file with file name given as string argument *str*.

`system(str)` Execute *str* as a system(shell) command in the operating system; return integer success value. Output into stdout from a shell command is put into the console window.

`timing(expr)` Evaluate expression *expr* and return the number of seconds (elapsed time) the evaluation took.

`typeof(variable)` Return the type of the *variable* as a string.

`saveModel(str,modelname)` Save the model/class with name *modelname* in the file given by the string argument *str*.

`val(variable,timePoint)` Return the (interpolated) value of the *variable* at time *timePoint*.

`help()` Print this helptext (returned as a string).

`quit()` Leave and quit the OpenModelica environment

1.4 Running the compiler from command line

The OpenModelica compiler can also be used from command line, in Windows cmd.exe.

Example Session 1 – obtaining information about command line parameters

```
C:\dev> C:\OpenModelica1.9.2\bin\omc -h
OpenModelica Compiler 1.9.2
Copyright © 2015 Open Source Modelica Consortium (OSMC)
Distributed under OSMC-PL and GPL, see https://www.openmodelica.org/
Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]
...
```

Example Session 2 - create an TestModel.mo file and run omc on it

```
C:\dev> echo model TestModel parameter Real x = 1; end TestModel; > TestModel.mo
C:\dev> C:\OpenModelica1.9.2\bin\omc TestModel.mo
class TestModel
    parameter Real x = 1.0;
end TestModel;
C:\dev>
```

Example Session 3 - create an script.mos file and run omc on it

Create a file script.mos using your editor containing these commands:

```
// start script.mos
loadModel(Modelica); getErrorString();
simulate(Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum); getErrorString();
// end script.mos
C:\dev> notepad script.mos
C:\dev> C:\OpenModelica1.9.2\bin\omc script.mos
true
...
record SimulationResult
    resultFile = "C:/dev/Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum_res.mat",
```

```
simulationOptions = "startTime = 0.0, stopTime = 5.0, numberOfIntervals = 500, tolerance = 1e-006,
method = 'dassl', fileNamePrefix = 'Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum',
options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = "",
messages = "",
timeFrontend = 1.245787339209033,
timeBackend = 20.51007138993843,
timeSimCode = 0.1510248469321959,
timeTemplates = 0.5052317333954395,
timeCompile = 5.128213942691722,
timeSimulation = 0.4049189573103951,
timeTotal = 27.9458487395605
end SimulationResult;
""
```

In order to obtain more information from the compiler one can use the command line options **+showErrorMessages +d=failtrace** when running the compiler:

```
C:\dev> C:\OpenModelica1.9.2\bin\omc +showErrorMessages +d=failtrace script.mos
```


OMEDIT – OPENMODELICA CONNECTION EDITOR

OMEdit – OpenModelica Connection Editor is the new Graphical User Interface for graphical model editing in OpenModelica. It is implemented in C++ using the Qt 4.8 graphical user interface library and supports the Modelica Standard Library version 3.1 that is included in the latest OpenModelica installation. This chapter gives a brief introduction to OMEdit and also demonstrates how to create a DCMotor model using the editor.

OMEdit provides several user friendly features for creating, browsing, editing, and simulating models:

- *Modeling* – Easy model creation for Modelica models.
- *Pre-defined models* – **Browsing the Modelica Standard library to** access the provided models.
- *User defined models* – **Users can create their own models for** immediate usage and later reuse.
- *Component interfaces* – **Smart connection editing for drawing and** editing connections between model interfaces.
- *Simulation* – **Subsystem for running simulations and specifying** simulation parameters start and stop time, etc.
- *Plotting* – Interface to plot variables from simulated models.

2.1 Starting OMEdit

A splash screen similar to the one shown in [Figure 2.1](#) will appear indicating that it is starting OMEdit. The executable is found in different places depending on the platform (see below).

2.1.1 Microsoft Windows

OMEdit can be launched using the executable placed in `OpenModelicaInstallationDirectory/bin/OMEdit/OMEdit.exe`. Alternately, choose `OpenModelica > OpenModelica Connection Editor` from the start menu in Windows.

2.1.2 Linux

Start OMEdit by either selecting the corresponding menu application item or typing “**OMEdit**” at the shell or command prompt.

2.1.3 Mac OS X

The default installation is `/Application/MacPorts/OMEdit.app`.

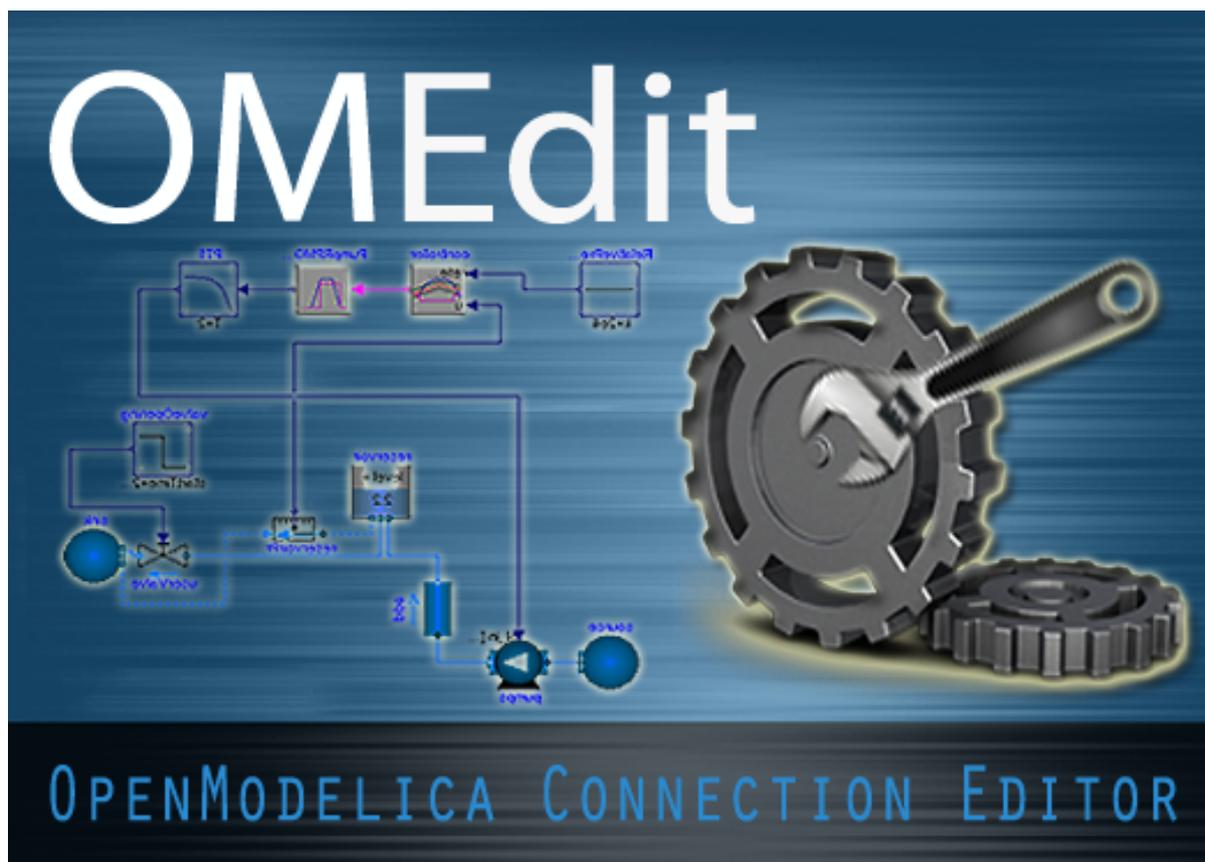


Figure 2.1: OMEdit Splash Screen.

2.2 MainWindow & Browsers

The MainWindow contains several dockable browsers,

- Search Browser
- Libraries Browser
- Documentation Browser
- Variables Browser
- Messages Browser

Figure 2.2 shows the MainWindow and browsers.

The default location of the browsers are shown in Figure 2.2. All browsers except for Message Browser can be docked into left or right column. The Messages Browser can be docked into left, right or bottom areas. If you want OMEdit to remember the new docked position of the browsers then you must enable Preserve User's GUI Customizations option, see section *General*.

2.2.1 Search Browser

To view the Search Browser click Edit->Search Browser or press keyboard shortcut Ctrl+Shift+F. The loaded Modelica classes can be searched by typing any part of the class name. It is also possible to search the Modelica class if one knows the text string that is used within it but Within Modelica text checkbox should be checked for this feature to work.

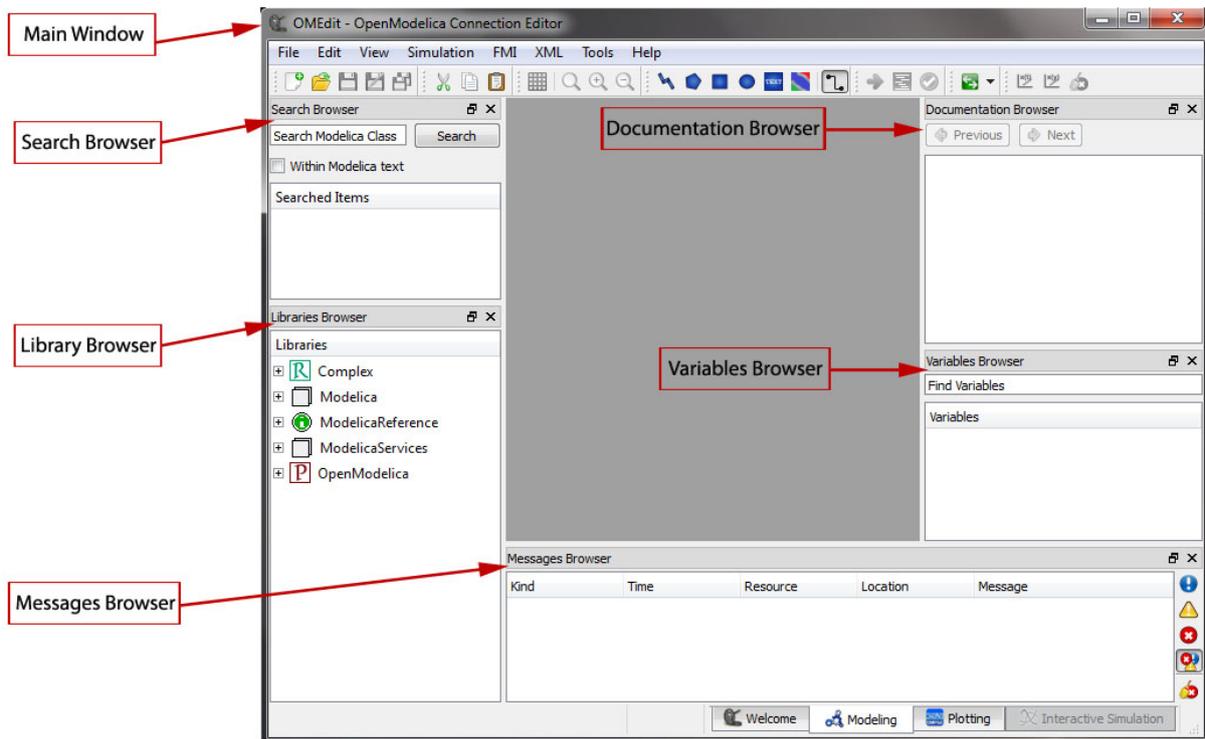


Figure 2.2: OMEdit MainWindow and Browsers.

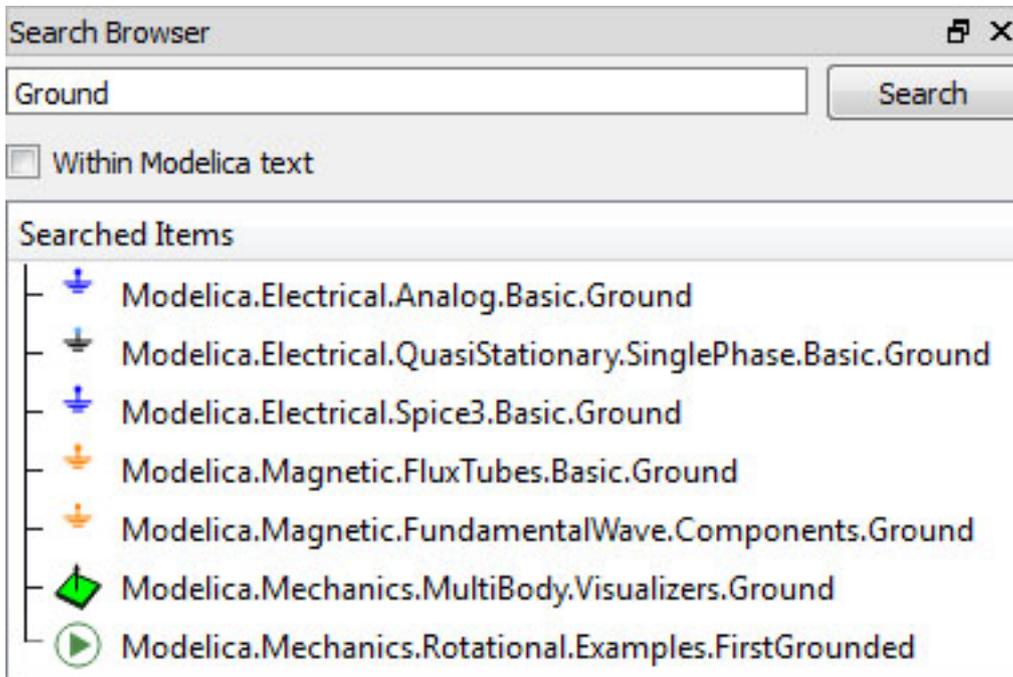


Figure 2.3: Search Browser.

2.2.2 Libraries Browser

To view the Libraries Browser click View->Windows->Libraries Browser. Shows the list of loaded Modelica classes. Each item of the Libraries Browser has right click menu for easy manipulation and usage of the class. The classes are shown in a tree structure with name and icon. The protected classes are not shown by default. If you want to see the protected classes then you must enable the Show Protected Classes option, see section *General*.

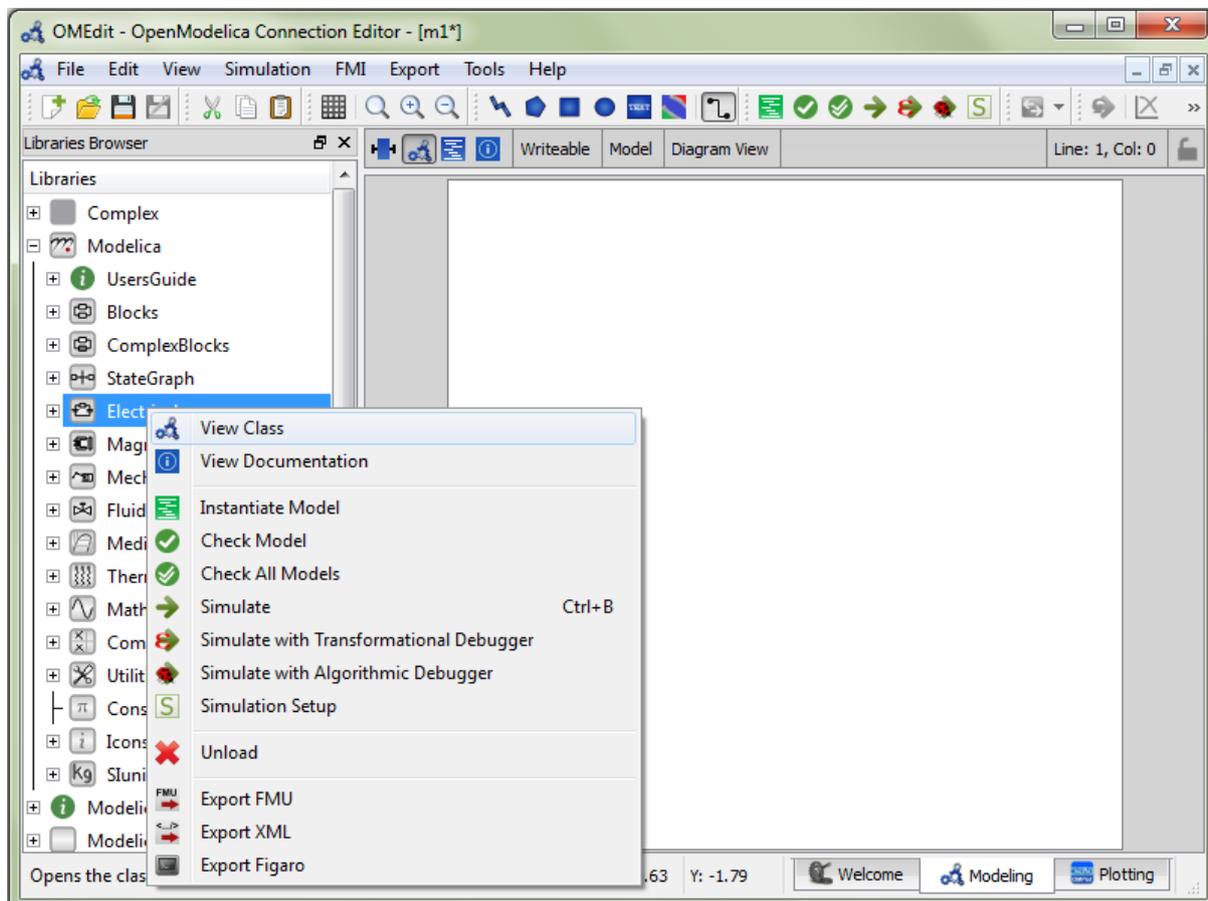


Figure 2.4: Libraries Browser.

2.2.3 Documentation Browser

Displays the HTML documentation of Modelica classes. It contains the navigation buttons for moving forward and backward. To see documentation of any class, right click the Modelica class in Libraries Browser and choose View Documentation.

2.2.4 Variables Browser

The class variables are structured in the form of the tree and are displayed in the Variables Browser. Each variable has a checkbox. Ticking the checkbox will plot the variable values. There is a find box on the top for filtering the variable in the tree. The filtering can be done using Regular Expression, Wildcard and Fixed String. The complete Variables Browser can be collapsed and expanded using the Collapse All and Expand All buttons.

The browser allows manipulation of changeable parameters for *Re-simulating a Model*. It also displays the unit and description of the variable.

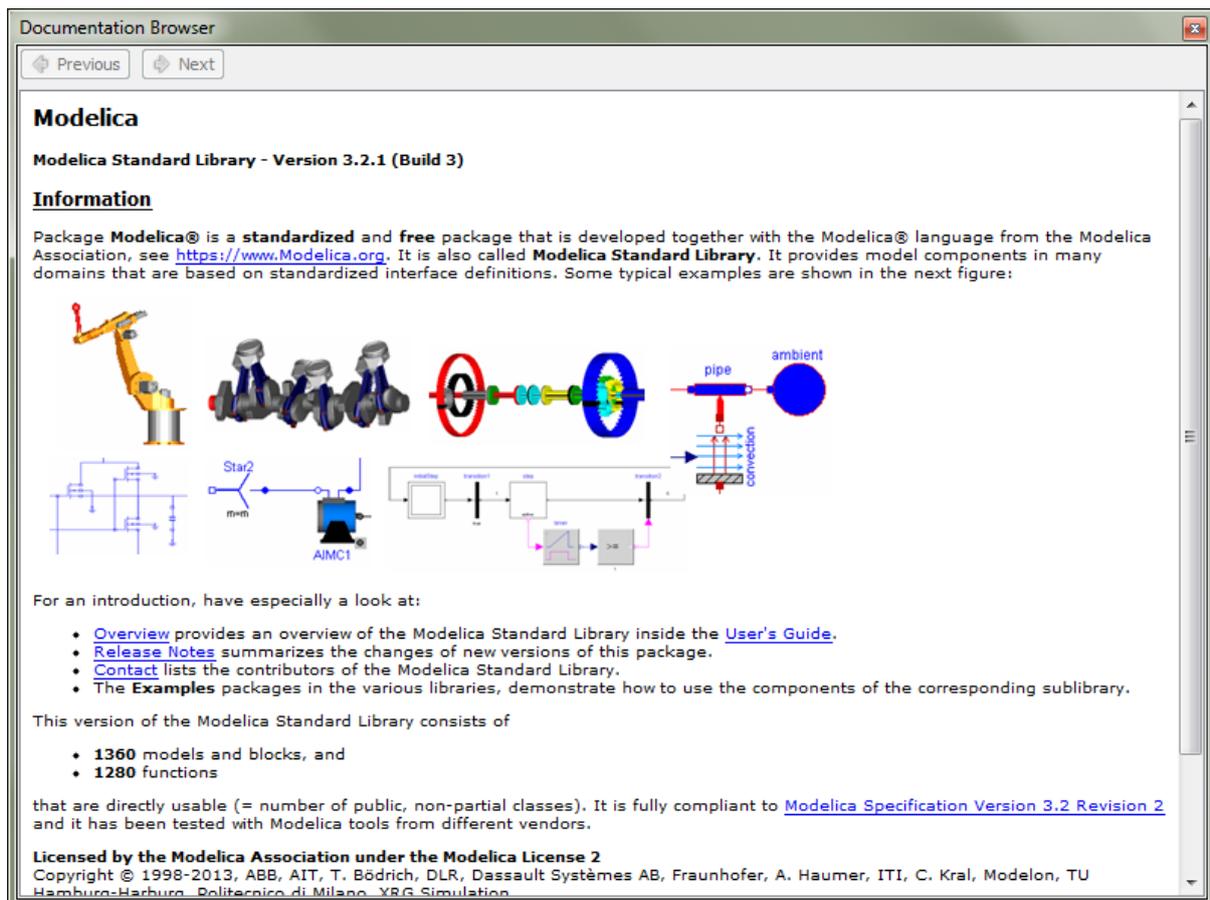


Figure 2.5: Documentation Browser.

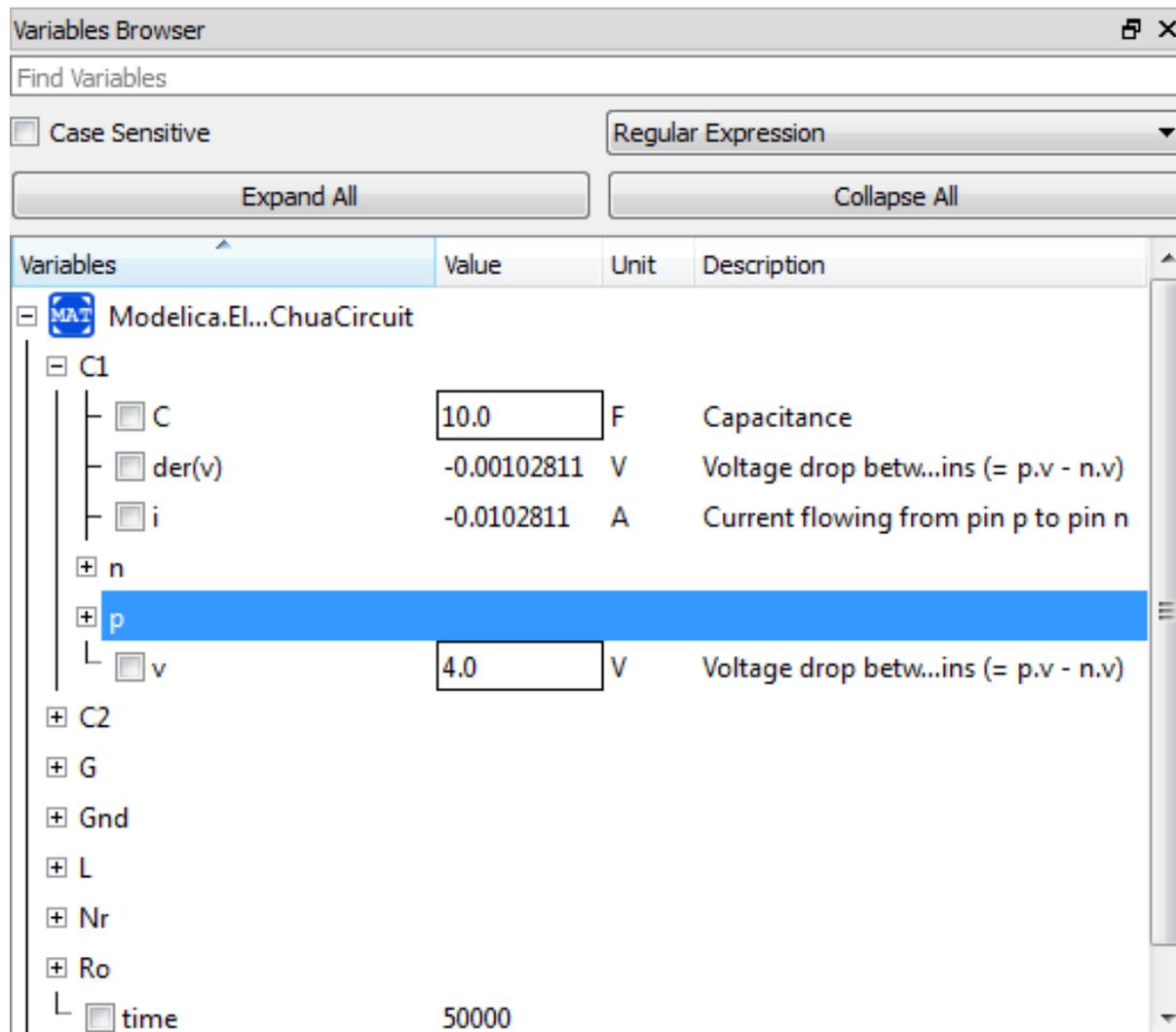


Figure 2.6: Variables Browser.

2.2.5 Messages Browser

Shows the list of errors. Following kinds of error can occur,

- Syntax
- Grammar
- Translation
- Symbolic
- Simulation
- Scripting

See section *Messages* for Messages Browser options.

2.3 Perspectives

The perspective tabs are located at the bottom right of the MainWindow:

- Welcome Perspective
- Modeling Perspective
- Plotting Perspective

2.3.1 Welcome Perspective

The Welcome Perspective shows the list of recent files and the list of latest news from <https://www.openmodelica.org/>. See [Figure 2.7](#). The orientation of recent files and latest news can be horizontal or vertical. User is allowed to show/hide the latest news. See section *General*.

2.3.2 Modeling Perspective

The Modeling Perspective provides the interface where user can create and design their models. See [Figure 2.8](#).

The Modeling Perspective interface can be viewed in two different modes, the tabbed view and subwindow view, see section *General*.

2.3.3 Plotting Perspective

The Plotting Perspective shows the simulation results of the models. Plotting Perspective will automatically become active when the simulation of the model is finished successfully. It will also become active when user opens any of the OpenModelica's supported result file. Similar to Modeling Perspective this perspective can also be viewed in two different modes, the tabbed view and subwindow view, see section *General*.

2.4 Modeling a Model

2.4.1 Creating a New Modelica class

Creating a new Modelica class in OMEdit is rather straightforward. Choose any of the following methods,

- Select File > New Modelica Class from the menu.
- Click on New Modelica Class toolbar button.
- **Click on the Create New Modelica Class button available at the left** bottom of Welcome Perspective.

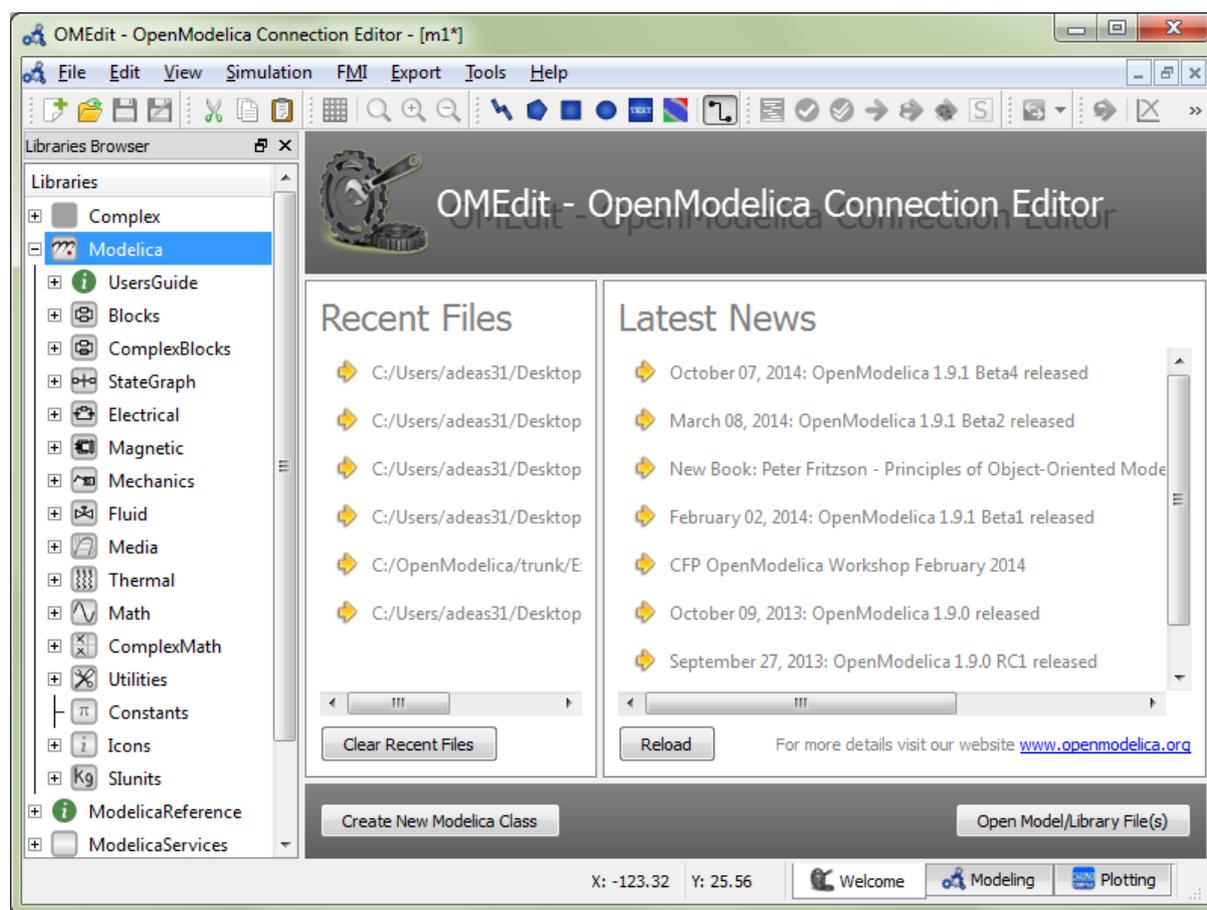


Figure 2.7: OMEdit Welcome Perspective.

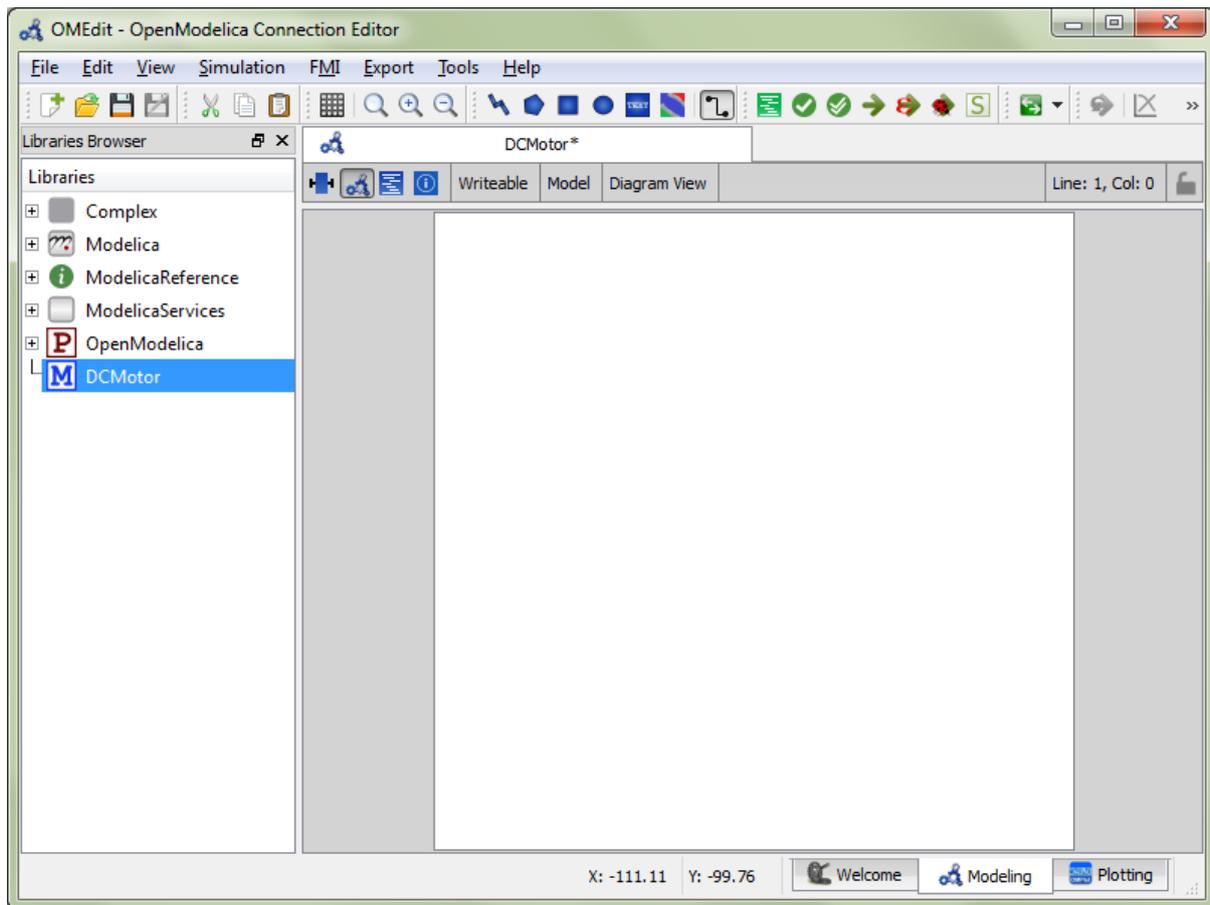


Figure 2.8: OMEdit Modeling Perspective.

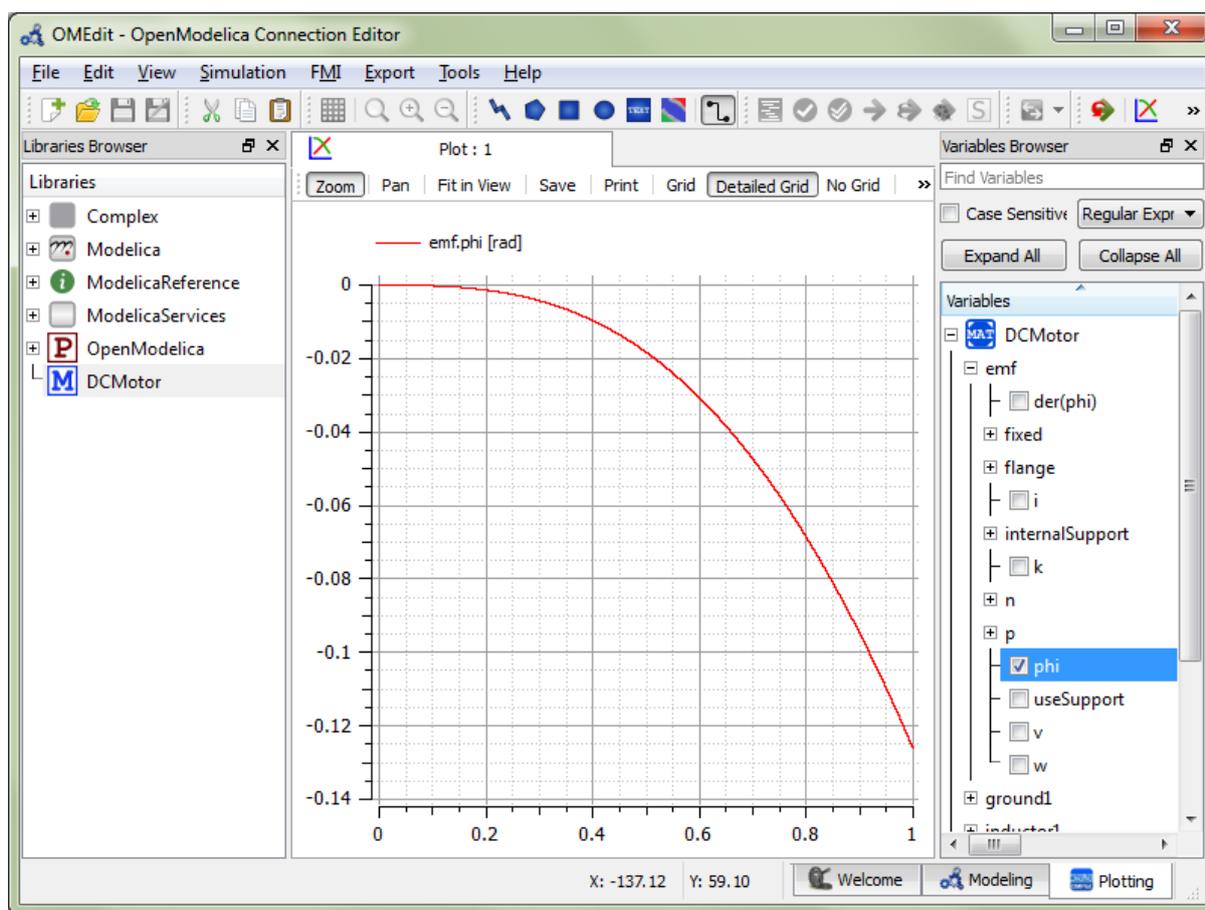


Figure 2.9: OMEdit Plotting Perspective.

- Press Ctrl+N.

2.4.2 Opening a Modelica File

Choose any of the following methods to open a Modelica file,

- Select File > Open Model/Library File(s) from the menu.
- Click on Open Model/Library File(s) toolbar button.
- **Click on the Open Model/Library File(s) button available at the right** bottom of Welcome Perspective.
- Press Ctrl+O.

2.4.3 Opening a Modelica File with Encoding

Select File > Open/Convert Modelica File(s) With Encoding from the menu. It is also possible to convert files to UTF-8.

2.4.4 Model Widget

For each Modelica class one Model Widget is created. It has a statusbar and a view area. The statusbar contains buttons for navigation between the views and labels for information. The view area is used to display the icon, diagram and text layers of Modelica class. See [Figure 2.10](#).

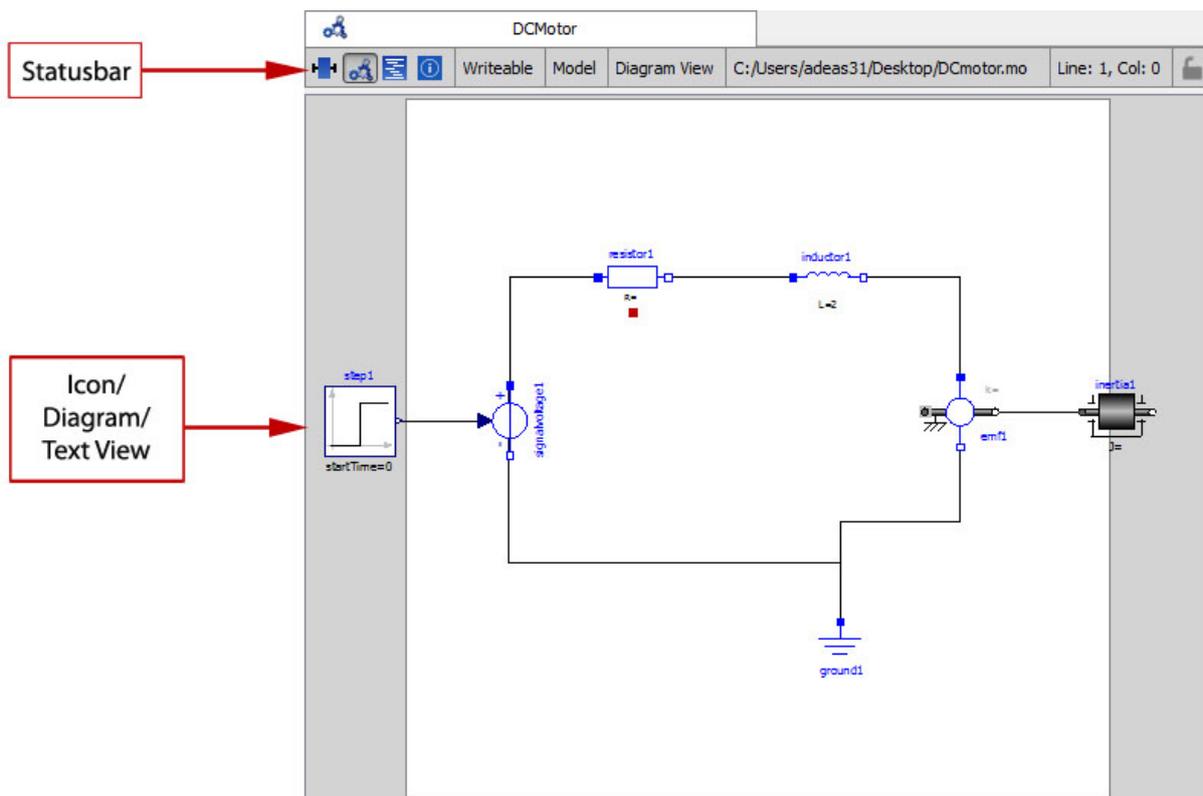


Figure 2.10: Model Widget showing the Diagram View.

2.4.5 Adding Component Models

Drag the models from the Libraries Browser and drop them on either Diagram or Icon View of Model Widget.

2.4.6 Making Connections

In order to connect one component model to another the user first needs to enable the connect mode () from the toolbar.

2.5 Simulating a Model

The OMEdit Simulation Dialog can be launched by,

- **Selecting Simulation > Simulation Setup from the menu.** (requires a model to be active in ModelWidget)
- **Clicking on the Simulation Setup toolbar button.** (requires a model to be active in ModelWidget)
- **Right clicking the model from the Libraries Browser and choosing Simulation Setup.**

2.5.1 General Tab

- *Start Time* – the simulation start time.
- *Stop Time* – the simulation stop time.
- *Method* – the simulation solver. See section *Integration Methods* for solver details.
- *Tolerance* – the simulation tolerance.
- *Compiler Flags (Optional)* – the optional C compiler flags.
- *Number of Processors* – the number of processors used to build the simulation.
- *Launch Transformational Debugger* – launches the transformational debugger.
- *Launch Algorithmic Debugger* – launches the algorithmic debugger.

2.5.2 Output Tab

- *Number of Intervals* – the simulation number of intervals.
- *Output Format* – the simulation result file output format.
- *File Name (Optional)* – the simulation result file name.
- *Variable Filter (Optional)*.
- *Protected Variables* – adds the protected variables in result file.
- *Store Variables at Events* – adds the variables at time events.
- *Show Generated File* – displays the generated files in a dialog box.

2.5.3 Simulation Flags Tab

- *Model Setup File (Optional)* – specifies a new setup XML file to the generated simulation code.
- *Initialization Method (Optional)* – specifies the initialization method.
- *Equation System Initialization File (Optional)* – specifies an external file for the initialization of the model.
- *Equation System Initialization Time (Optional)* – specifies a time for the initialization of the model.
- *Clock (Optional)* – the type of clock to use.
- *Linear Solver (Optional)* – specifies the linear solver method.

- *Non Linear Solver (Optional)* – specifies the nonlinear solver.
- *Linearization Time (Optional)* – specifies a time where the linearization of the model should be performed.
- *Output Variables (Optional)* – outputs the variables **a, b and c** at the end of the simulation to the standard output.
- *Profiling* – creates a profiling HTML file.
- *CPU Time* – dumps the cpu-time into the result file.
- *Enable All Warnings* – outputs all warnings.
- *Logging (Optional)*
- *DASSL Solver Information* – prints additional information about dassl solver.
- *Debug* – prints additional debug information.
- *Dynamic State Selection Information* – outputs information about dynamic state selection.
- *Jacobians Dynamic State Selection Information* – outputs jacobain of the dynamic state selection.
- *Event Iteration* – additional information during event iteration.
- *Verbose Event System* – verbose logging of event system.
- *Initialization* – prints additional information during initialization.
- *Jacobians Matrix* – outputs the jacobian matrix used by dassl.
- *Non Linear Systems* – logging for nonlinear systems.
- *Verbose Non Linear Systems* – verbose logging of nonlinear systems.
- *Jacobians Non Linear Systems* – outputs the jacobian of nonlinear systems.
- *Initialization Residuals* – outputs residuals of the initialization.
- *Simulation Process* – additional information about simulation process.
- *Solver Process* – additional information about solver process.
- *Final Initialization Solution* – final solution of the initialization.
- *Timer/Event/Solver Statistics* – additional statistics about timer/events/solver.
- *Util.*
- *Zero Crossings* – additional information about the zerocrossings.
- *Additional Simulation Flags (Optional)* – specify any other simulation flag.

2.6 Plotting the Simulation Results

Successful simulation of model produces the result file which contains the instance variables that are candidate for plotting. Variables Browser will show the list of such instance variables. Each variable has a checkbox, checking it will plot the variable. See [Figure 2.9](#).

2.6.1 Types of Plotting

The plotting type depends on the active Plot Window. By default the plotting type is Time Plot.

Time Plot

Plots the variable over the simulation time. You can have multiple Time Plot windows by clicking on New Plot Window toolbar button .

Plot Parametric

Draws a two-dimensional parametric diagram, between variables x and y , with y as a function of x . You can have multiple Plot Parametric windows by clicking on the New Plot Parametric toolbar button .

2.7 Re-simulating a Model

The *Variables Browser* allows manipulation of changeable parameters for re-simulation. After changing the parameter values user can click on the re-simulate toolbar button , or right click the model in Variables Browser and choose re-simulate from the menu.

2.8 How to Create User Defined Shapes – Icons

Users can create shapes of their own by using the shape creation tools available in OMEdit.

- **Line Tool – Draws a line. A line is created with a minimum of two** points. In order to create a line, the user first selects the line tool from the toolbar and then click on the Icon/Diagram View; this will start creating a line. If a user clicks again on the Icon/Diagram View a new line point is created. In order to finish the line creation, user has to double click on the Icon/Diagram View.
- **Polygon Tool – Draws a polygon. A polygon is created in a similar** fashion as a line is created. The only difference between a line and a polygon is that, if a polygon contains two points it will look like a line and if a polygon contains more than two points it will become a closed polygon shape.
- **Rectangle Tool – Draws a rectangle. The rectangle only contains two** points where first point indicates the starting point and the second point indicates the ending the point. In order to create rectangle, the user has to select the rectangle tool from the toolbar and then click on the Icon/Diagram View, this click will become the first point of rectangle. In order to finish the rectangle creation, the user has to click again on the Icon/Diagram View where he/she wants to finish the rectangle. The second click will become the second point of rectangle.
- **Ellipse Tool – Draws an ellipse. The ellipse is created in a** similar way as a rectangle is created.
- **Text Tool** – Draws a text label.
- **Bitmap Tool** – Draws a bitmap container.

The shape tools are located in the toolbar. See [Figure 2.11](#).

The user can select any of the shape tools and start drawing on the Icon/Diagram View. The shapes created on the Diagram View of Model Widget are part of the diagram and the shapes created on the Icon View will become the icon representation of the model.

For example, if a user creates a model with name testModel and add a rectangle using the rectangle tool and a polygon using the polygon tool, in the Icon View of the model. The model's Modelica Text will appear as follows:

```
model testModel
  annotation (Icon (graphics = {Rectangle (rotation = 0, lineColor = {0,0,255}, fillColor = {0,0,255}
end testModel;
```

In the above code snippet of testModel, the rectangle and a polygon are added to the icon annotation of the model. Similarly, any user defined shape drawn on a Diagram View of the model will be added to the diagram annotation of the model.

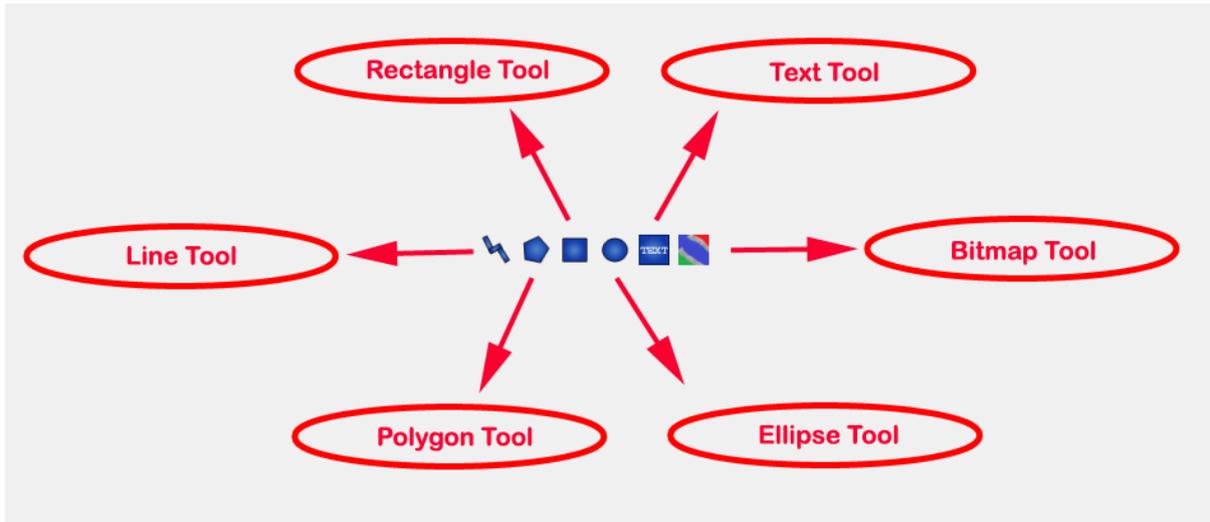


Figure 2.11: User defined shapes.

2.9 Settings

OMEdit allows users to save several settings which will be remembered across different sessions of OMEdit. The Options Dialog can be used for reading and writing the settings.

2.9.1 General

- General
- *Language* – Sets the application language.
- *Working Directory* – Sets the application working directory.
- *Toolbar Icon Size* – Sets the size for toolbar icons.
- *Preserve User's GUI Customizations* – If true then OMEdit will remember its windows and toolbars positions and sizes.
- Libraries Browser
- *Library Icon Size* – Sets the size for library icons.
- *Show Protected Classes* – Sets the application language.
- Modeling View Mode
- *Tabbed View/SubWindow View* – Sets the view mode for modeling.
- Plotting View Mode
- *Tabbed View/SubWindow View* – Sets the view mode for plotting.
- Default View
- *Icon View/DiagramView/Modelica Text View/Documentation View* – If no preferredView annotation is defined then this setting is used to show the respective view when user double clicks on the class in the Libraries Browser.
- Enable Auto Save
- *Auto Save interval* – Sets the auto save interval value. The minimum possible interval value is 60 seconds.
- *Enable Auto Save for single classes* – Enables the auto save for one class saved in one file.
- *Enable Auto Save for one file packages* – Enables the auto save for packages saved in one file.

- Welcome Page
- *Horizontal View/Vertical View* – Sets the view mode for welcome page.
- *Show Latest News* – if true then displays the latest news.

2.9.2 Libraries

- *System Libraries* – **The list of system libraries that should be** loaded every time OMEdit starts.
- *Force loading of Modelica Standard Library* – **If true then Modelica** and `ModelicaReference` will always load even if user has removed them from the list of system libraries.
- *User Libraries* – **The list of user libraries/files that should be** loaded every time OMEdit starts.

2.9.3 Modelica Text Editor

- General
- *Enable Syntax Highlighting* – Enable/Disable the syntax highlighting for the Modelica Text Widget.
- *Enable Line Wrapping* – Enable/Disable the line wrapping for the Modelica Text Widget.
- Fonts and Colors
- *Font Family* – Contains the names list of available fonts.
- *Font Size* – Sets the font size.
- *Items* – List of categories used of syntax highlighting the code.
- *Item Color* – Sets the color for the selected item.
- *Preview* – Shows the demo of the syntax highlighting.

2.9.4 Graphical Views

- Extent
- *Left* – Defines the left extent point for the view.
- *Bottom* – Defines the bottom extent point for the view.
- *Right* – Defines the right extent point for the view.
- *Top* – Defines the top extent point for the view.
- Grid
- *Horizontal* – Defines the horizontal size of the view grid.
- *Vertical* – Defines the vertical size of the view grid.
- Component
- *Scale factor* – Defines the initial scale factor for the component dragged on the view.
- *Preserve aspect ratio* – If true then the component's aspect ratio is preserved while scaling.

2.9.5 Simulation

- Simulation
- *Matching Algorithm* – sets the matching algorithm for simulation.
- *Index Reduction Method* – sets the index reduction method for simulation.

- *OMC Flags* – sets the omc flags for simulation.
- *Save class before simulation* – if true then always saves the class before running the simulation.
- *Output*
- *Structured* – Shows the simulation output in the form of tree structure.
- *Formatted Text* – Shows the simulation output in the form of formatted text.

2.9.6 Messages

- *General*
- *Output Size* - Specifies the maximum number of rows the Messages Browser may have. If there are more rows then the rows are removed from the beginning.
- *Reset messages number before simulation* – Resets the messages counter before starting the simulation.
- *Font and Colors*
- *Font Family* – Sets the font for the messages.
- *Font Size* – Sets the font size for the messages.
- *Notification Color* – Sets the text color for notification messages.
- *Warning Color* – Sets the text color for warning messages.
- *Error Color* – Sets the text color for error messages.

2.9.7 Notifications

- *Notifications*
- *Always quit without prompt* – If true then OMEdit will quit without prompting the user.
- *Show item dropped on itself message* – If true then a message will pop-up when a class is dragged and dropped on itself.
- *Show model is defined as partial and component will be added as replaceable message* – If true then a message will pop-up when a partial class is added to another class.
- *Show component is declared as inner message* – If true then a message will pop-up when an inner component is added to another class.
- *Show save model for bitmap insertion message* – If true then a message will pop-up when user tries to insert a bitmap from a local directory to an unsaved class.

2.9.8 Line Style

- *Line Style*
- *Color* – Sets the line color.
- *Pattern* – Sets the line pattern.
- *Thickness* – Sets the line thickness.
- *Start Arrow* – Sets the line start arrow.
- *End Arrow* – Sets the line end arrow.
- *Arrow Size* – Sets the start and end arrow size.
- *Smooth* – If true then the line is drawn as a Bezier curve.

2.9.9 Fill Style

- Fill Style
- *Color* – Sets the fill color.
- *Pattern* – Sets the fill pattern.

2.9.10 Curve Style

- Curve Style
- *Pattern* – Sets the curve pattern.
- *Thickness* – Sets the curve thickness.

2.9.11 Figaro

- Figaro
- *Figaro Database File* – the Figaro database file path.
- *Figaro Mode* –
- *Figaro Options File* – the Figaro options file path.
- *Figaro Process* – the Figaro process location.

2.9.12 Debugger

- Algorithmic Debugger
- *GDB Path* – the gnu debugger path
- *GDB Command Timeout* – timeout for gdb commands.
- *Display C frames* – if true then shows the C stack frames.
- *Display unknown frames* – if true then shows the unknown stack frames. Unknown stack frames means frames whose file path is unknown.
- *Clear old output on a new run* – if true then clears the output window on new run.
- *Clear old log on new run* – if true then clears the log window on new run.
- Transformational Debugger
- *Always show Transformational Debugger after compilation* – if true then always open the Transformational Debugger window after model compilation.
- *Generate operations in the info xml* – if true then adds the operations information in the info xml file.

2.9.13 FMI

- Export
- *Version 1.0* – Sets the FMI export version to 1.0
- *Version 2.0* – Sets the FMI export version to 2.0

2.10 Debugger

For debugging capability, see [‘debugging’_](#).

2D PLOTTING

This chapter covers the 2D plotting available in OpenModelica via OMNotebook, OMShell and command line script. The plotting is based on OMPlot application.

3.1 Example

```
class HelloWorld
  Real x(start = 1, fixed = true);
  parameter Real a = 1;
equation
  der(x) = - a * x;
end HelloWorld;
```

To create a simple time plot the above model HelloWorld is simulated. To reduce the amount of simulation data in this example the number of intervals is limited with the argument numberOfIntervals=5. The simulation is started with the command below.

```
>>> simulate(HelloWorld, outputFormat="csv", startTime=0, stopTime=4, numberOfIntervals=5)
record SimulationResult
  resultFile = "«DOCHOME»/HelloWorld_res.csv",
  simulationOptions = "startTime = 0.0, stopTime = 4.0, numberOfIntervals = 5, tolerance = 1e-06",
  messages = "",
  timeFrontend = 0.005910406,
  timeBackend = 0.003416457,
  timeSimCode = 0.050474886,
  timeTemplates = 0.003695134,
  timeCompile = 0.211992532,
  timeSimulation = 0.006793014,
  timeTotal = 0.282396426
end SimulationResult;
```

When the simulation is finished the file *HelloWorld_res.csv* contains the simulation data:

Listing 3.1: HelloWorld_res.csv

```
"time", "x", "der(x) "
0, 1, -1
0.8, 0.4493289092712475, -0.4493289092712475
1.6, 0.2018973974273906, -0.2018973974273906
2.4, 0.09071896372718975, -0.09071896372718975
3.2, 0.04076293845066793, -0.04076293845066793
4, 0.01831609502171534, -0.01831609502171534
4, 0.01831609502171534, -0.01831609502171534
```

Diagrams are now created with the new OMPlot program by using the following plot command:

By re-simulating and saving results at many more points, for example using the default 500 intervals, a much smoother plot can be obtained. Note that the default solver method dassl has more internal points than the output points in the initial plot. The results are identical, except the detailed plot has a smoother curve.

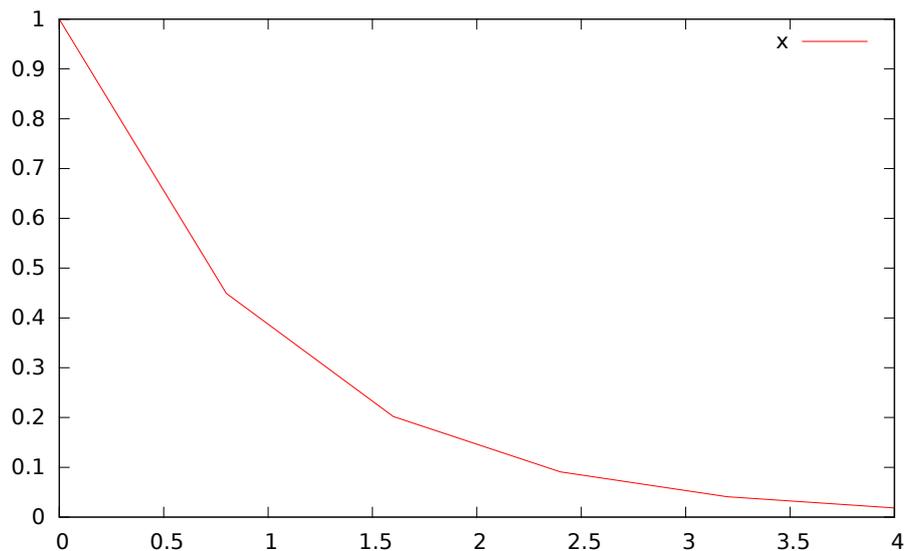


Figure 3.1: Simple 2D plot of the HelloWorld example.

```
>>> 0==system("./HelloWorld -override stepSize=0.008")
true
>>> res:=strtok(readFile("HelloWorld_res.csv"), "\n");
>>> res[end]
"4,0.01831609502171534,-0.01831609502171534"
```

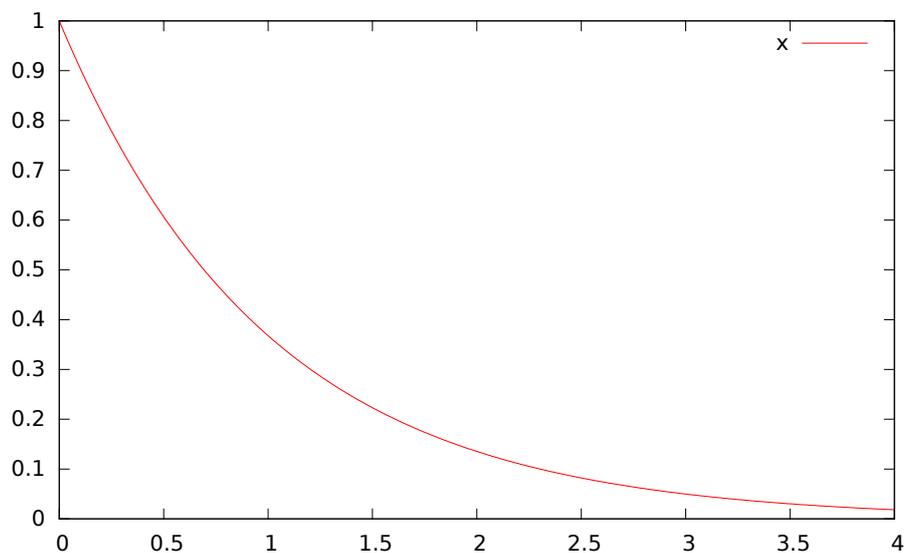


Figure 3.2: Simple 2D plot of the HelloWorld example with a larger number of output points.

3.2 Plotting Commands and their Options

All of these commands can have any number of optional arguments to further customize the the resulting diagram. The available options and their allowed values are listed below.

Option	Default value	Description
fileName	The result of the last simulation	The name of the result-file containing the variables to plot
grid	true	Determines whether or not a grid is shown in the diagram.
title	""	This text will be used as the diagram title.
logX	false	Determines whether or not the horizontal axis is logarithmically scaled.
logY	false	Determines whether or not the vertical axis is logarithmically scaled.
xLabel	"time"	This text will be used as the horizontal label in the diagram.
yLabel	""	This text will be used as the vertical label in the diagram.
xRange	{0, 0}	Determines the horizontal interval that is visible in the diagram. {0, 0} will select a suitable range.
yRange	{0, 0}	Determines the vertical interval that is visible in the diagram. {0, 0} will select a suitable range.
curveWidth	1.0	Defines the width of the curve.
curveStyle	1	Defines the style of the curve. SolidLine=1, DashLine=2, DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.
legend-Position	"top"	Defines the position of the legend in the diagram. Possible values are left, right, top, bottom and none.
external-Window	false	Opens a new OMPlot window if set to true otherwise update the current opened window.

DEBUGGING

There are two main ways to debug Modelica code, the transformations browser, which shows the transformations OpenModelica performs on the equations. There is also a debugger for *debugging of algorithm sections and functions*.

4.1 The Equation-based Debugger

This section gives a short description how to get started using the equation-based debugger in OMEdit.

4.1.1 Enable Tracing Symbolic Transformations

This enables tracing symbolic transformations of equations. It is optional but strongly recommended in order to fully use the debugger. The compilation time overhead from having this tracing on is less than 1%, however, in addition to that, some time is needed for the system to write the xml file containing the transformation tracing information.

Enable `+d=infoXmlOperations` in Tools->Options->Simulation (see section *Simulation*) OR alternatively click on the checkbox *Generate operations in the info xml* in Tools->Options->Debugger (see section *Debugger*) which performs the same thing.

This adds all the transformations performed by OpenModelica on the equations and variables stored in the `model_info.xml` file. This is necessary for the debugger to be able to show the whole path from the source equation(s) to the position of the bug.

4.1.2 Load a Model to Debug

Load an interesting model. We will use the package `Debugging.mo` since it contains suitable, broken models to demonstrate common errors.

Todo

Include `Debugging.mo` in the test models in the distribution?

4.1.3 Simulate and Start the Debugger

Select and simulate the model as usual. For example, if using the `Debugging` package, select the model `Debugging.Chattering.ChatteringEvents1`. If there is an error, you will get a clickable link that starts the debugger. If the user interface is unresponsive or the running simulation uses too much processing power, click cancel simulation first.

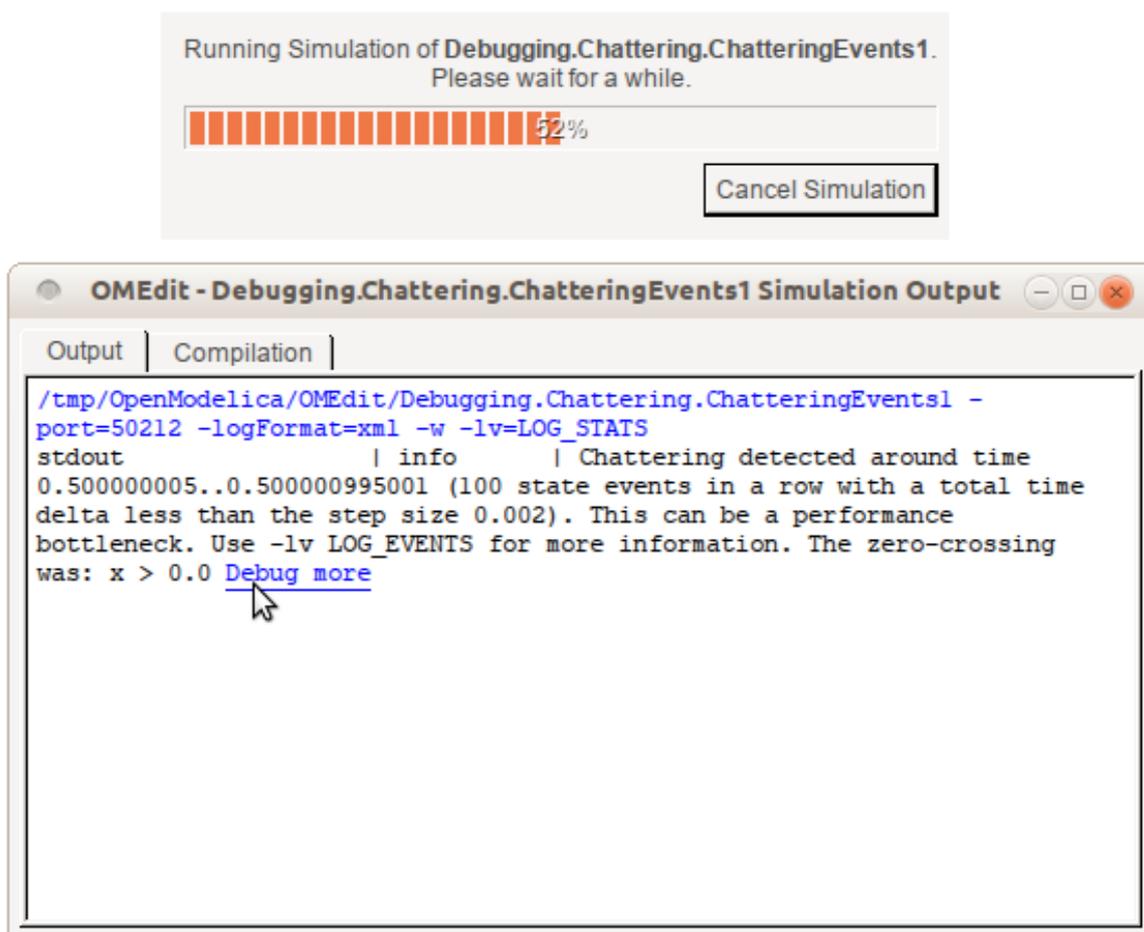


Figure 4.1: Simulating the model.

4.1.4 Use the Transformation Debugger for Browsing

Use the transformation debugger. It opens on the equation where the error was found. You can browse through the dependencies (variables that are defined by the equation, or the equation is dependent on), and similar for variables. The equations and variables form a bipartite graph that you can walk.

If the `+d=infoXmlOperations` was used or you clicked the “generate operations” button, the operations performed on the equations and variables can be viewed. In the example package, there are not a lot of operations because the models are small.

Try some larger models, e.g. in the MultiBody library or some other library, to see more operations with several transformation steps between different versions of the relevant equation(s). If you do not trigger any errors in a model, you can still open the debugger, using File->Open Transformations File (model_info.json).

The screenshot shows the OMEdit - Transformational Debugger interface. The main window is titled "/tmp/OpenModelica_marsj/OMEdit/Debugging.Chattering.ChatteringEvents1_info.xml". The interface is divided into several panes:

- Variables Browser:** A table with columns 'Variables', 'Comment', 'Line', and 'Location'. It lists variables x, y, and z.
- Defined In Equations:** A table with columns 'Inc', 'Type', and 'Equation'. It shows equations 2 and 5 defining variables.
- Used In Equations:** A table with columns 'Inc', 'Type', and 'Equation'. It shows equations 3 and 6 using variables.
- Equations Browser:** A table with columns 'Inc', 'Type', and 'Equation'. It lists equations 1 through 7, with equation 5 highlighted in orange.
- Variable Operations:** A section for viewing operations on variables.
- Equation Operations:** A section for viewing operations on equations, showing a flattened version of equation 5.
- Source Browser:** A code editor showing the source code of the model, with the equation for z highlighted in blue.

Figure 4.2: Transformations Browser.

4.2 The Algorithmic Debugger

This section gives a short description how to get started using the algorithmic debugger in OMEdit. See section [Simulation](#) for further details of debugger options/settings. The Algorithmic Debugger window can be launched from Tools->Windows->Algorithmic Debugger.

4.2.1 Adding Breakpoints

There are two ways to add the breakpoints,

- Click directly on the line number in Text View, a red circle is created indicating a breakpoint as shown in Figure 4.3.
- Open the Algorithmic Debugger window and add a breakpoint using the right click menu of Breakpoints Browser window.

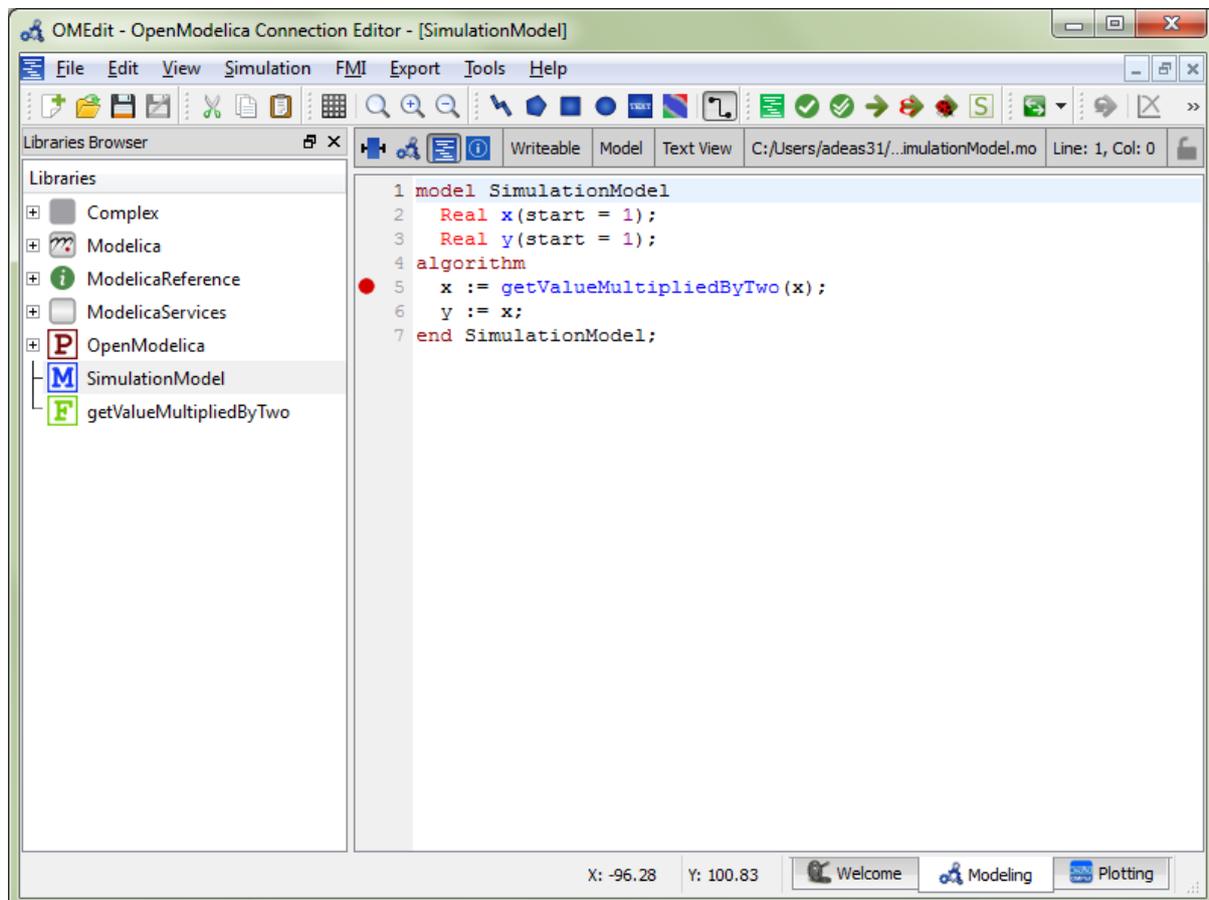


Figure 4.3: Adding breakpoint in Text View.

4.2.2 Start the Algorithmic Debugger

You should add breakpoints before starting the debugger because sometimes the simulation finishes quickly and you won't get any chance to add the breakpoints.

There are four ways to start the debugger,

- Open the Simulation Setup and click on **Launch Algorithmic Debugger** before pressing Simulate.
- Right click the model in Libraries Browser and select **Simulate with** Algorithmic Debugger.
- Open the Algorithmic Debugger window and from menu select **Debug-> Debug Configurations**.
- Open the Algorithmic Debugger window and from menu select **Debug-> Attach to Running Process**.

4.2.3 Debug Configurations

If you already have a simulation executable with debugging symbols outside of OMEdit then you can use the **Debug->Debug Configurations** option to load it.

The debugger also supports MetaModelica data structures so one can debug omc executable. Select omc executable as program and write the name of the mos script file in Arguments.

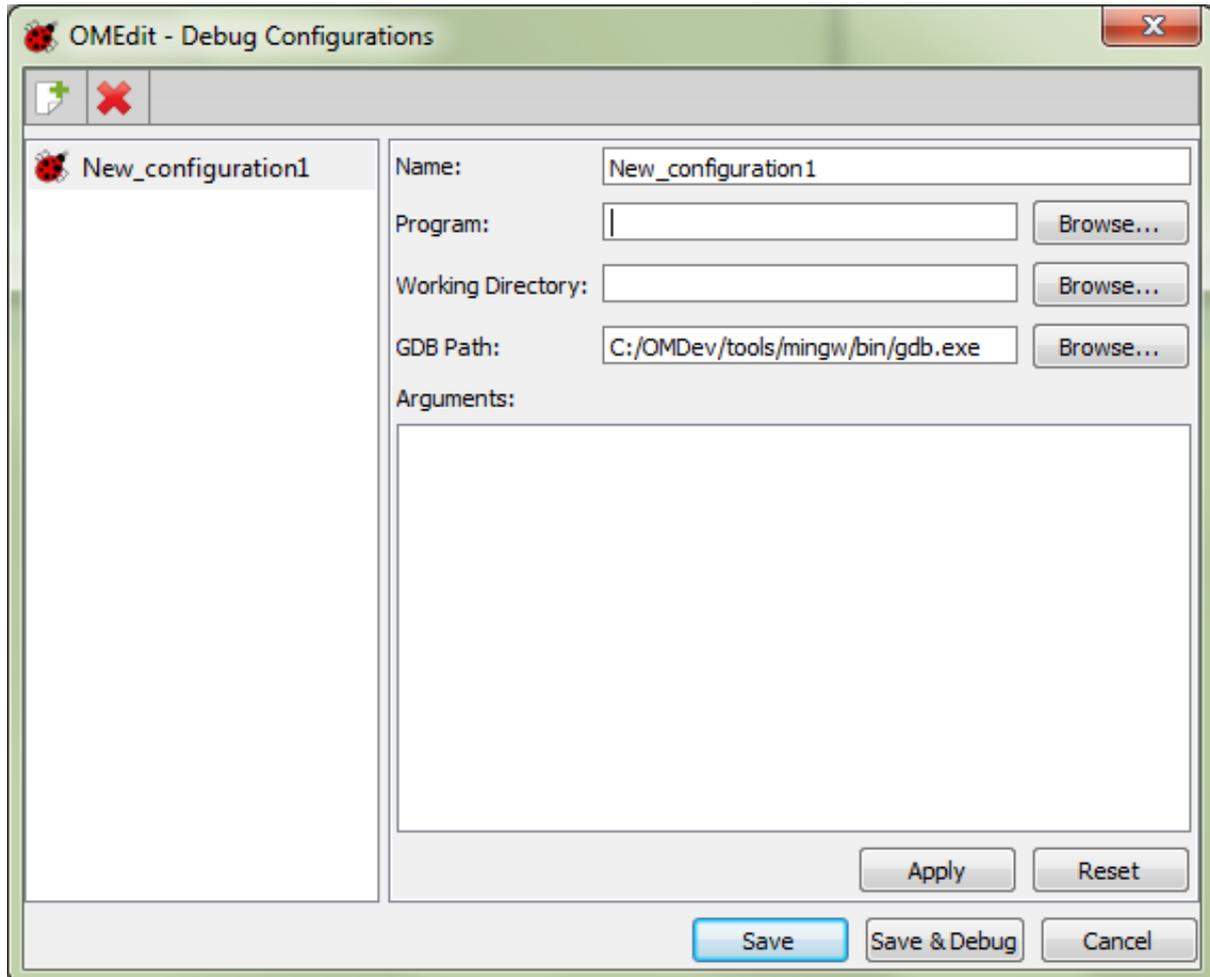


Figure 4.4: Debug Configurations.

4.2.4 Attach to Running Process

If you already have a running simulation executable with debugging symbols outside of OMEdit then you can use the Debug->Attach to Running Process option to attach the debugger with it. Figure 4.5 shows the Attach to Running Process dialog. The dialog shows the list of processes running on the machine. The user selects the program that he/she wish to debug. OMEdit debugger attaches to the process.

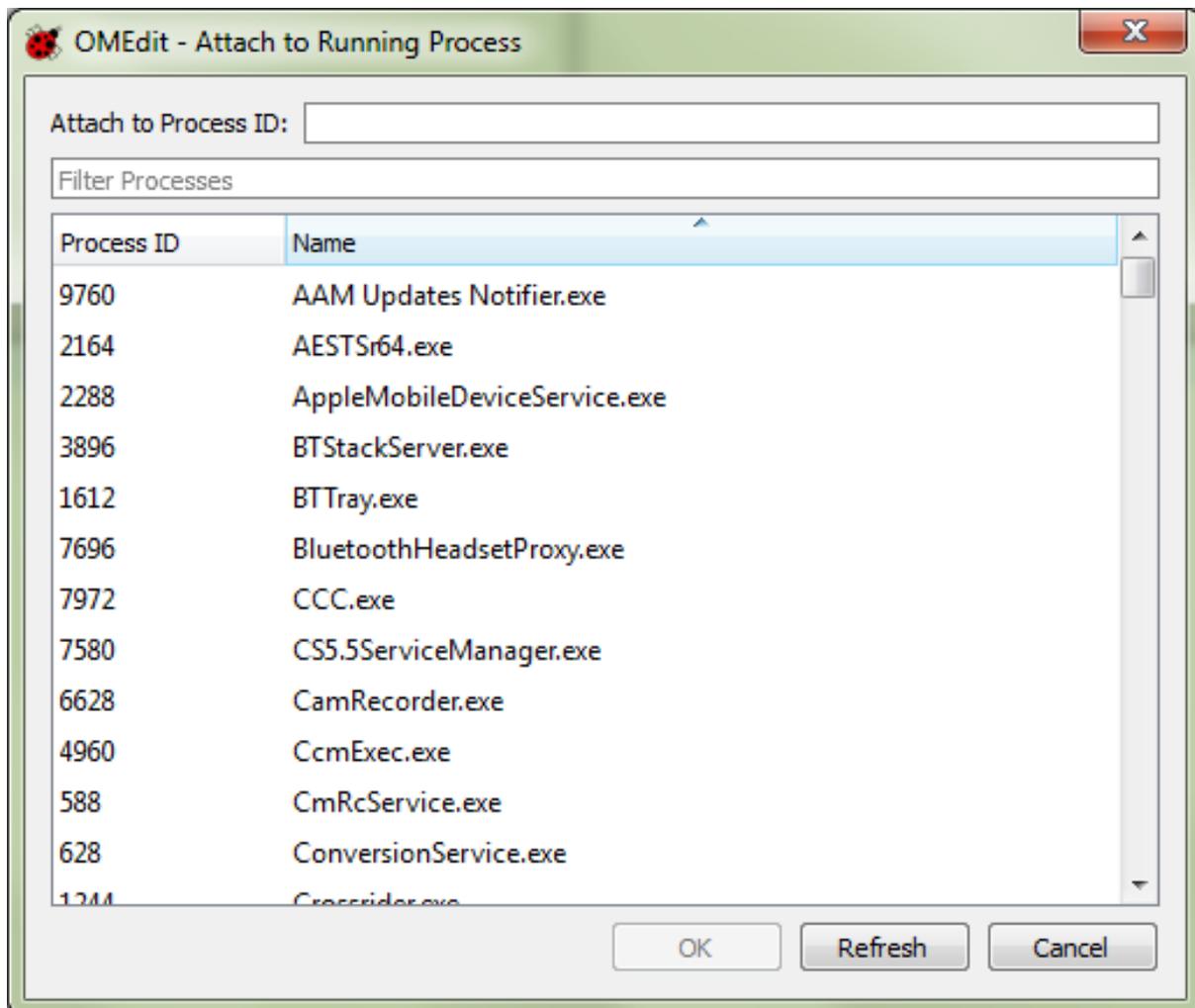


Figure 4.5: Attach to Running Process.

4.2.5 Using the Algorithmic Debugger Window

Figure 4.6 shows the Algorithmic Debugger window. The window contains the following browsers,

- **Stack Frames Browser** – shows the list of frames. It contains the program context buttons like resume, interrupt, exit, step over, step in, step return. It also contains a threads drop down which allows switching between different threads.
- **BreakPoints Browser** – shows the list of breakpoints. Allows adding/editing/removing breakpoints.
- **Locals Browser** – Shows the list of local variables with values. Select the variable and the value will be shown in the bottom right window. This is just for convenience because some variables might have long values.
- **Debugger CLI** – shows the commands sent to gdb and their responses. This is for advanced users who want to have more control of the debugger. It allows sending commands to gdb.

- *Output Browser* – shows the output of the debugged executable.

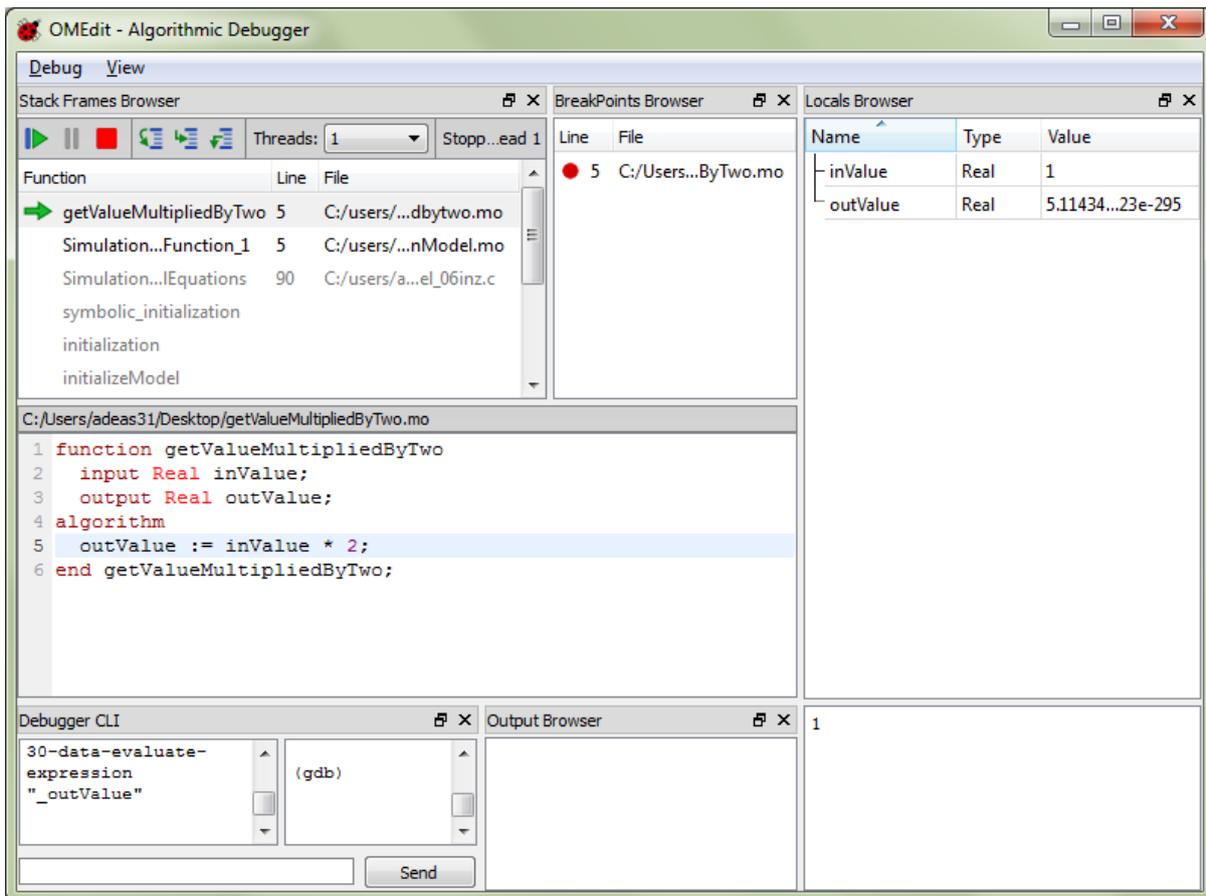


Figure 4.6: Algorithmic Debugger.

OMNOTEBOOK WITH DRMODELICA AND DRCONTROL

This chapter covers the OpenModelica electronic notebook subsystem, called OMNotebook, together with the DrModelica tutoring system for teaching Modelica, and DrControl for teaching control together with Modelica. Both are using such notebooks.

5.1 Interactive Notebooks with Literate Programming

Interactive Electronic Notebooks are active documents that may contain technical computations and text, as well as graphics. Hence, these documents are suitable to be used for teaching and experimentation, simulation scripting, model documentation and storage, etc.

5.1.1 Mathematica Notebooks

Literate Programming [Knu84] is a form of programming where programs are integrated with documentation in the same document. Mathematica notebooks [Wol96] is one of the first WYSIWYG (What-You-See-Is-What-You-Get) systems that support Literate Programming. Such notebooks are used, e.g., in the MathModelica modeling and simulation environment, see e.g. Figure 5.1 below and Chapter 19 in [Fri04].

5.1.2 OMNotebook

The OMNotebook software [Axe05][Fernstrom06] is a new open source free software that gives an interactive WYSIWYG realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document.

The OMNotebook facility is actually an interactive WYSIWYG realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document. OMNotebook is a simple open-source software tool for an electronic notebook supporting Modelica.

A more advanced electronic notebook tool, also supporting mathematical typesetting and many other facilities, is provided by Mathematica notebooks in the MathModelica environment, see Figure 5.1.

Traditional documents, e.g. books and reports, essentially always have a hierarchical structure. They are divided into sections, subsections, paragraphs, etc. Both the document itself and its sections usually have headings as labels for easier navigation. This kind of structure is also reflected in electronic notebooks. Every notebook corresponds to one document (one file) and contains a tree structure of cells. A cell can have different kinds of contents, and can even contain other cells. The notebook hierarchy of cells thus reflects the hierarchy of sections and subsections in a traditional document such as a book.

5.2 DrModelica Tutoring System – an Application of OMNotebook

Understanding programs is hard, especially code written by someone else. For educational purposes it is essential to be able to show the source code and to give an explanation of it at the same time.

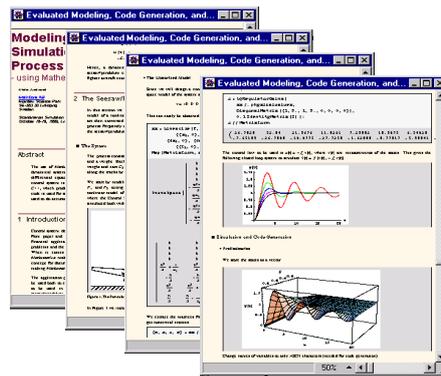


Figure 5.1: Examples of Mathematica notebooks in the MathModelica modeling and simulation environment.

Moreover, it is important to show the result of the source code's execution. In modeling and simulation it is also important to have the source code, the documentation about the source code, the execution results of the simulation model, and the documentation of the simulation results in the same document. The reason is that the problem solving process in computational simulation is an iterative process that often requires a modification of the original mathematical model and its software implementation after the interpretation and validation of the computed results corresponding to an initial model.

Most of the environments associated with equation-based modeling languages focus more on providing efficient numerical algorithms rather than giving attention to the aspects that should facilitate the learning and teaching of the language. There is a need for an environment facilitating the learning and understanding of Modelica. These are the reasons for developing the DrModelica teaching material for Modelica and for teaching modeling and simulation.

An earlier version of DrModelica was developed using the MathModelica (now Wolfram SystemModeler) environment. The rest of this chapter is concerned with the OMNotebook version of DrModelica and on the OMNotebook tool itself.

DrModelica has a hierarchical structure represented as notebooks. The front-page notebook is similar to a table of contents that holds all other notebooks together by providing links to them. This particular notebook is the first page the user will see (Figure 5.2).

In each chapter of DrModelica the user is presented a short summary of the corresponding chapter of the Modelica book [Fri04]. The summary introduces some *keywords*, being hyperlinks that will lead the user to other notebooks describing the keywords in detail.

Now, let us consider that the link “HelloWorld” in DrModelica Section is clicked by the user. The new HelloWorld notebook (see Figure 5.3), to which the user is being linked, is not only a textual description but also contains one or more examples explaining the specific keyword. In this class, HelloWorld, a differential equation is specified.

No information in a notebook is fixed, which implies that the user can add, change, or remove anything in a notebook. Alternatively, the user can create an entirely new notebook in order to write his/her own programs or copy examples from other notebooks. This new notebook can be linked from existing notebooks.

When a class has been successfully evaluated the user can simulate and plot the result, as previously depicted in Figure 5.3 for the simple HelloWorld example model.

After reading a chapter in DrModelica the user can immediately practice the newly acquired information by doing the exercises that concern the specific chapter. Exercises have been written in order to elucidate language constructs step by step based on the pedagogical assumption that a student learns better “*using the strategy of learning by doing*”. The exercises consist of either theoretical questions or practical programming assignments. All exercises provide answers in order to give the user immediate feedback.

Figure 5.4 shows part of Chapter 9 of the DrModelica teaching material. Here the user can read about language constructs, like algorithm sections, when-statements, and reinit equations, and then practice these constructs by solving the exercises corresponding to the recently studied section.

Exercise 1 from Chapter 9 is shown in Figure 5.5. In this exercise the user has the opportunity to practice different language constructs and then compare the solution to the answer for the exercise. Notice that the answer is not

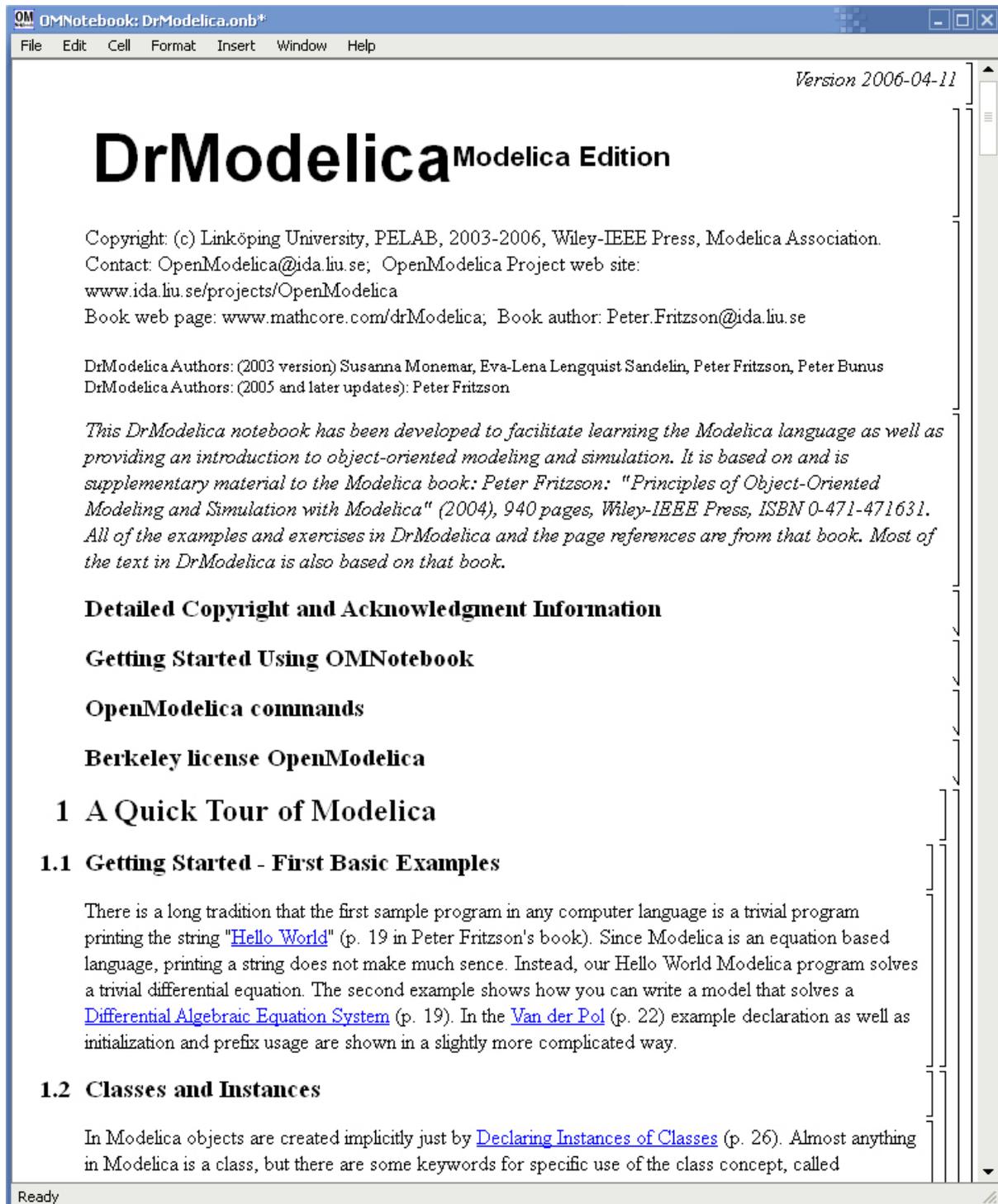


Figure 5.2: The front-page notebook of the OMNotebook version of the DrModelica tutoring system.

The screenshot shows the OMNotebook interface with the following content:

First Basic Class

1 HelloWorld

The program contains a declaration of a class called `HelloWorld` with two fields and one equation. The first field is the variable `x` which is initialized to a start value `2` at the time when the simulation starts. The second field is the variable `a`, which is a constant that is initialized to `2` at the beginning of the simulation. Such a constant is prefixed by the keyword parameter in order to indicate that it is constant during simulation but is a model parameter that can be changed between simulations.

The Modelica program solves a trivial differential equation: $x' = -a * x$. The variable `x` is a state variable that can change value over time. The x' is the time derivative of `x`.

```
class HelloWorld
  Real x(start = 1, fixed=true);
  parameter Real a = 1;
equation
  der(x) = - a * x;
end HelloWorld;
```

{HelloWorld}

2 Simulation of HelloWorld

```
simulate( HelloWorld, startTime=0, stopTime=3 )

record SimulationResult
  resultFile = "HelloWorld_res.mat",
  messages = ""
end SimulationResult;
```

Plot the results.

```
plot( x )
```

[done]

Zoom Pan Auto Scale Fit in View Save Print Grid Detailed Grid No Grid Log X Log Y Setup

— x

Plot by OpenModelica

time	x
0.0	1.00
0.5	0.61
1.0	0.37
1.5	0.22
2.0	0.14
2.5	0.08
3.0	0.05
3.5	0.03
4.0	0.02

Ready

Figure 5.3: The HelloWorld class simulated and plotted using the OMNotebook version of DrModelica.

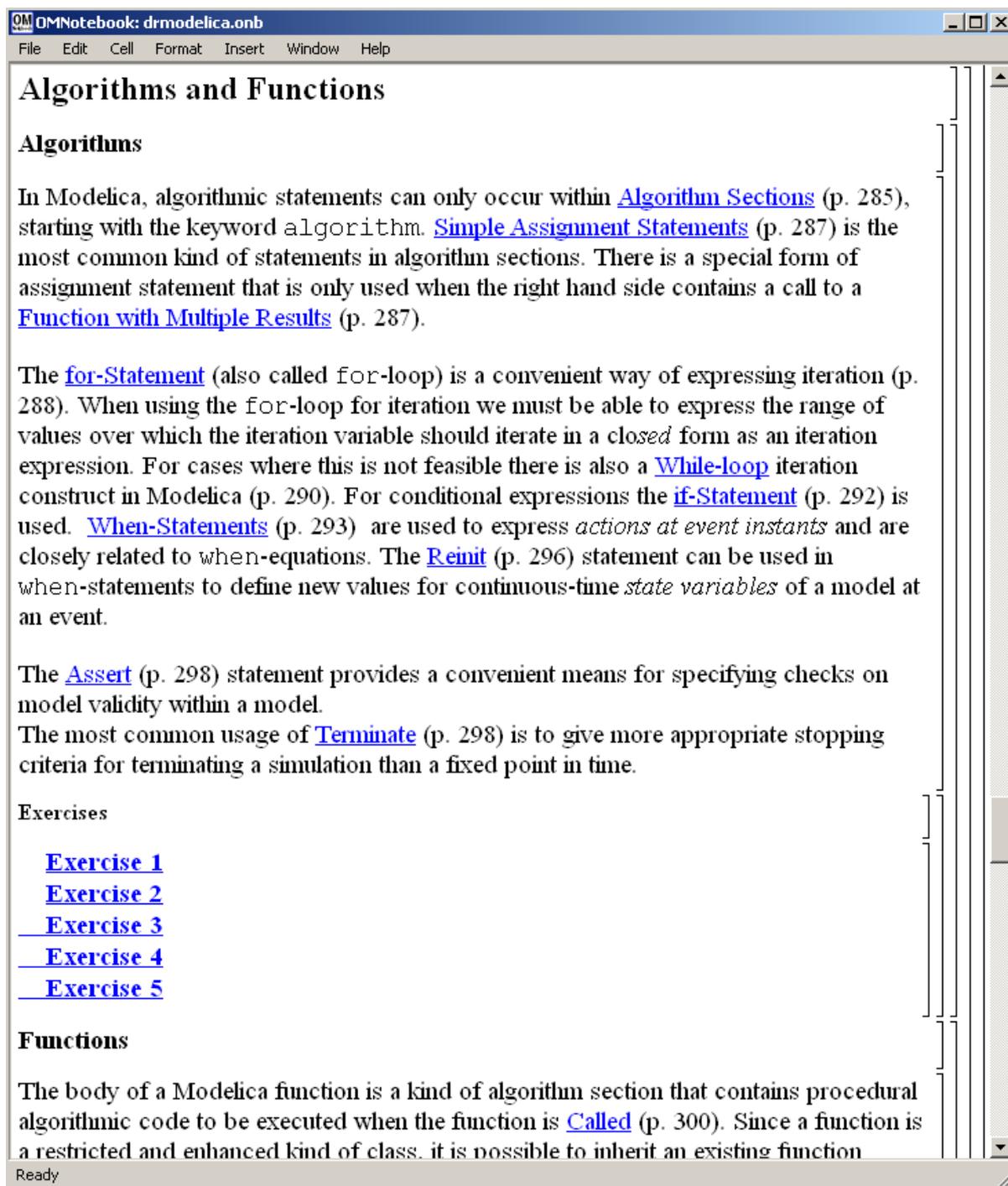


Figure 5.4: DrModelica Chapter on Algorithms and Functions in the main page of the OMNotebook version of DrModelica.

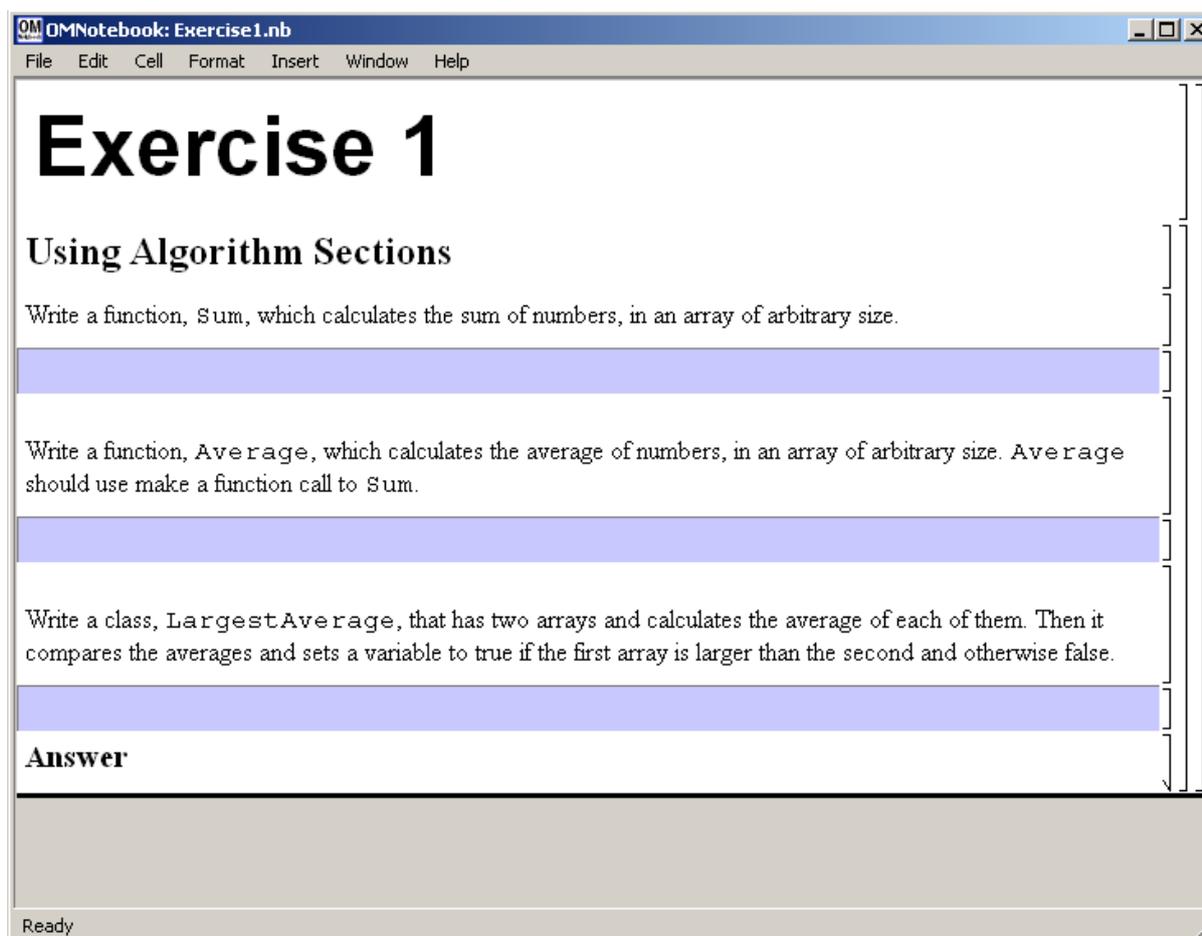


Figure 5.5: Exercise 1 in Chapter 9 of DrModelica.

visible until the *Answer* section is expanded. The answer is shown in [Figure 5.6](#).

5.3 DrControl Tutorial for Teaching Control Theory

DrControl is an interactive OMNotebook document aimed at teaching control theory. It is included in the OpenModelica distribution and appears under the directory:

```
>>> getInstallationDirectoryPath() + "/share/omnotebook/drcontrol"
"«OPENMODELICAHOME»/share/omnotebook/drcontrol"
```

The front-page of DrControl resembles a linked table of content that can be used as a navigation center. The content list contains topics like:

- Getting started
- The control problem in ordinary life
- Feedback loop
- Mathematical modeling
- Transfer function
- Stability
- Example of controlling a DC-motor
- Feedforward compensation
- State-space form
- State observation
- Closed loop control system.
- Reconstructed system
- Linear quadratic optimization
- Linearization

Each entry in this list leads to a new notebook page where either the theory is explained with Modelica examples or an exercise with a solution is provided to illustrate the background theory. Below we show a few sections of DrControl.

5.3.1 Feedback Loop

One of the basic concepts of control theory is using feedback loops either for neutralizing the disturbances from the surroundings or a desire for a smoother output.

In [Figure 5.7](#), control of a simple car model is illustrated where the car velocity on a road is controlled, first with an open loop control, and then compared to a closed loop system with a feedback loop. The car has a mass m , velocity y , and aerodynamic coefficient α . The θ is the road slope, which in this case can be regarded as noise.

Lets look at the Modelica model for the open loop controlled car:

$$m\dot{y} = u - \alpha y - mg * \sin(\theta)$$

```
model noFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y; // output signal without noise, theta = 0 -> v(t) =
  SI.Velocity yNoise; // output signal with noise, theta <> 0 -> v(t)
  parameter SI.Mass m = 1500;
  parameter Real alpha = 200;
```

Answer

Sum

```
function Sum
  input Real[:] x;
  output Real sum;
algorithm
  for i in 1:size(x,1) loop
    sum := sum + x[i];
  end for;
end Sum;
```

Average

```
function Average
  input Real[:] x;
  output Real average;
protected
  Real sum;
algorithm
  average := Sum(x) / size(x,1);
end Average;
```

LargestAverage

```
class LargestAverage
  parameter Integer[:] A1 = {1, 2, 3, 4, 5};
  parameter Integer[:] A2 = {7, 8, 9};
  Real averageA1, averageA2;
  Boolean A1Largest(start = false);
algorithm
  averageA1 := Average(A1);
  averageA2 := Average(A2);
  if averageA1 > averageA2 then
    A1Largest := true;
  else
    A1Largest := false;
  end if;
end LargestAverage;
```

Simulation of LargestAverage

```
simulate( LargestAverage );
```

When we look at the values in the variables we see that A2 has the largest average (8) and therefore the variable A1Largest is false (= 0).

Ready

Figure 5.6: The answer section to Exercise 1 in Chapter 9 of DrModelica.

File Edit Cell Format Insert Window Help

Feedback

One important method in designing control system is a feedback loop. It can be used to eliminate the influence of noise or to decrease the output error.

1 Example

Assume that we want to control the speed of a car on the road. The car has a mass m , velocity y , and aerodynamic coefficient α . The θ is the road slope, which in this case can be regarded as noise.

$$m\dot{y} = u - \alpha y - mg\sin(\theta)$$

If we want a reference speed of 20 m/s for a car with $m=1500$ kg, $\alpha=250$ Ns/m, $\theta=0$ rad, how high should the amplification factor be in the regulator?
Try with $u = 250*r$.

1.1 Open Loop

```

loadModel(Modelica)
true
model noFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y; // output signal without
  noise, theta = 0 -> v(t) = 0 // output signal with noise
  SI.Velocity vNoise;

```

Figure 5.7: Feedback loop.

```

parameter SI.Angle theta = 5*3.141592/180;
parameter SI.Acceleration g = 9.82;
SI.Force u;
SI.Velocity r=20;
equation
m*der(y)=u-alpha*y; // signal without noise
m*der(yNoise)=u-alpha*yNoise-m*g*sin(theta); // with noise
u = 250*r;
end noFeedback;

```

By applying a road slope angle different from zero the car velocity is influenced which can be regarded as noise in this model. The output signal in Figure 5.8 is stable but an overshoot can be observed compared to the reference signal. Naturally the overshoot is not desired and the student will in the next exercise learn how to get rid of this undesired behavior of the system.

```

>>> loadModel(Modelica)
true
>>> simulate(noFeedback, stopTime=100)
record SimulationResult
  resultFile = "«DOCHOME»/noFeedback_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 100.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.319539773,
  timeBackend = 0.005813808,
  timeSimCode = 0.051529470999999994,
  timeTemplates = 0.0044300760000000001,
  timeCompile = 0.224850526,
  timeSimulation = 0.0075723230000000001,
  timeTotal = 0.613846662
end SimulationResult;

```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

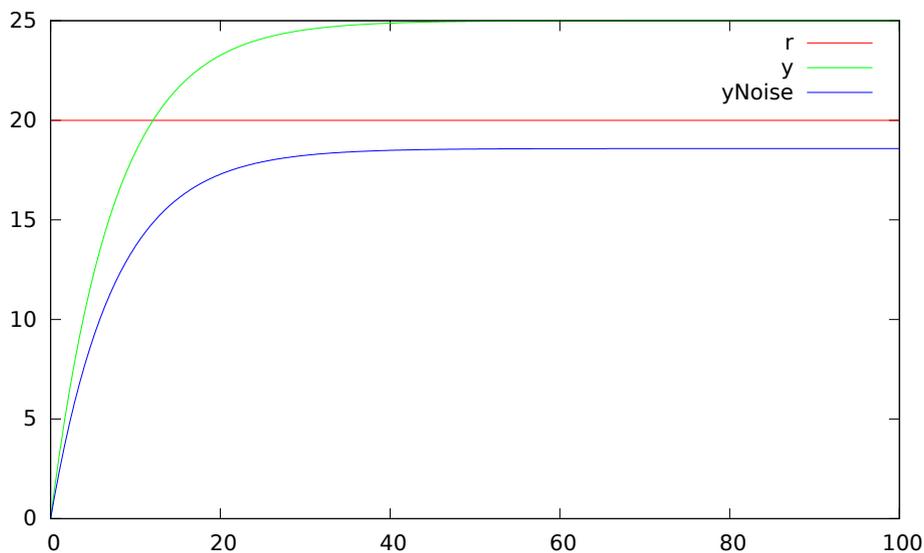


Figure 5.8: Open loop control example.

The closed car model with a proportional regulator is shown below:

$$u = K * (r - y)$$

```

model withFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y; // output signal with feedback link and without noise
  SI.Velocity yNoise; // output signal with feedback link and noise,
  parameter SI.Mass m = 1500;
  parameter Real alpha = 250;
  parameter SI.Angle theta = 5*3.141592/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Force uNoise;
  SI.Velocity r=20;
equation
  m*der(y)=u-alpha*y;
  m*der(yNoise)=uNoise-alpha*yNoise-m*g*sin(theta);
  u = 5000*(r-y);
  uNoise = 5000*(r-yNoise);
end withFeedback;

```

By using the information about the current level of the output signal and re-tune the regulator the output quantity can be controlled towards the reference signal smoothly and without an overshoot, as shown in [Figure 5.9](#).

In the above simple example the flat modeling approach was adopted since it was the fastest one to quickly obtain a working model. However, one could use the object oriented approach and encapsulate the car and regulator models in separate classes with the Modelica connector mechanism in between.

```

>>> loadModel(Modelica)
true
>>> simulate(withFeedback, stopTime=10)
record SimulationResult
  resultFile = "«DOCHOME»/withFeedback_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.234737719,
  timeBackend = 0.05831245,
  timeSimCode = 0.046095904,
  timeTemplates = 0.0069769440000000001,
  timeCompile = 0.22905171,
  timeSimulation = 0.0075438840000000001,
  timeTotal = 0.582805286
end SimulationResult;

```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

5.3.2 Mathematical Modeling with Characteristic Equations

In most systems the relation between the inputs and outputs can be described by a linear differential equation. Tearing apart the solution of the differential equation into homogenous and particular parts is an important technique taught to the students in engineering courses, also illustrated in [Figure 5.10](#).

$$\frac{\partial^n y}{\partial t^n} + a_1 \frac{\partial^{n-1} y}{\partial t^{n-1}} + \dots + a_n y = b_0 \frac{\partial^m u}{\partial t^m} + \dots + b_{m-1} \frac{\partial u}{\partial t} + b_m u$$

Now let us examine a second order system:

$$\ddot{y} + a_1 \dot{y} + a_2 y = 1$$

```

model NegRoots
  Real y;

```

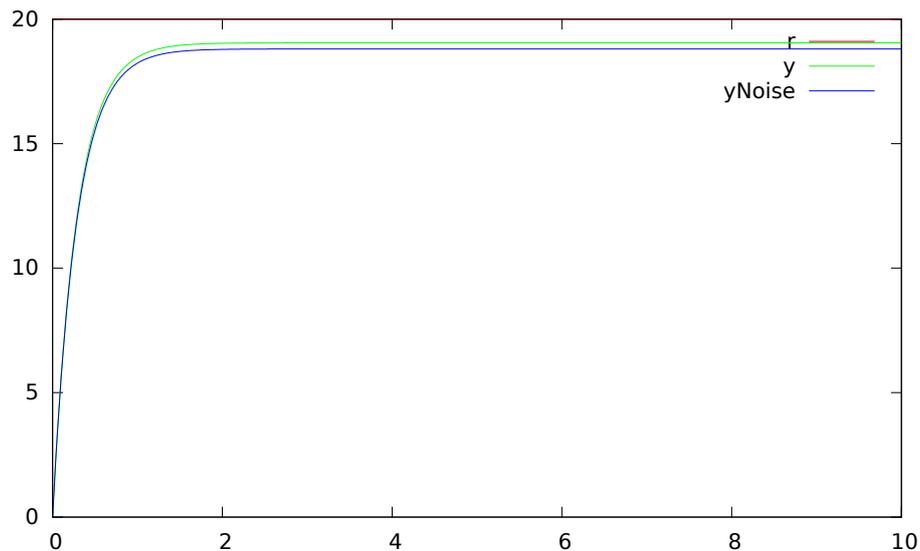


Figure 5.9: Closed loop control example.

```

Real der_y;
parameter Real a1 = 3;
parameter Real a2 = 2;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;

```

Choosing different values for a_1 and a_2 leads to different behavior as shown in [Figure 5.11](#) and [Figure 5.12](#).

In the first example the values of a_1 and a_2 are chosen in such way that the characteristic equation has negative real roots and thereby a stable output response, see [Figure 5.11](#).

```

>>> simulate(NegRoots, stopTime=10)
record SimulationResult
  resultFile = "«DOCHOME»/NegRoots_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.21539976399999999,
  timeBackend = 0.002103542,
  timeSimCode = 0.056900064,
  timeTemplates = 0.003597831,
  timeCompile = 0.217988514,
  timeSimulation = 0.007672385,
  timeTotal = 0.503746721
end SimulationResult;

```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

The importance of the sign of the roots in the characteristic equation is illustrated in [Figure 5.11](#) and [Figure 5.12](#), e.g., a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

```

model ImgPosRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;

```

File Edit Cell Format Insert Window Help

Mathematical Modeling

In most systems the relation between the inputs and outputs can be approximated by a linear differential equation.

$$\frac{d^n}{dt^n}y(t) + a_1 \frac{d^{n-1}}{dt^{n-1}}y(t) + \dots + a_n y(t) = b_0 \frac{d^m}{dt^m}u(t) + \dots + b_{m-1} \frac{d}{dt}u(t) + b_m u(t)$$

where the coefficients a_i and b_i are constants. The above differential equation has a homogeneous and a particular solution:

$$y = y_h + y_p$$

The homogeneous solution where u is set to zero has the form:

$$y_h = C_1 e^{\lambda_1 t} + \dots + C_n e^{\lambda_n t}$$

where

$$\lambda^n + a_1 \lambda^{n-1} + \dots + a_{n-1} \lambda + a_n = 0$$

1 Example

Consider the following model.

$$\frac{d^2}{dt^2}y(t) + a_1 \frac{d^1}{dt^1}y(t) + a_2 y(t) = 1$$

Examine the behavior of the system for different values on a_1 and a_2 .

1.1 Characteristic Equation with Negative Real Roots, $\lambda=-1,-2$

```

model negRoots
  Real y;
  Real der_y;
  parameter Real a1 = 3;
  parameter Real a2 = 2;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end negRoots;
{negRoots}
simulate(negRoots.stopTime=10)

```

Figure 5.10: Mathematical modeling with characteristic equation.

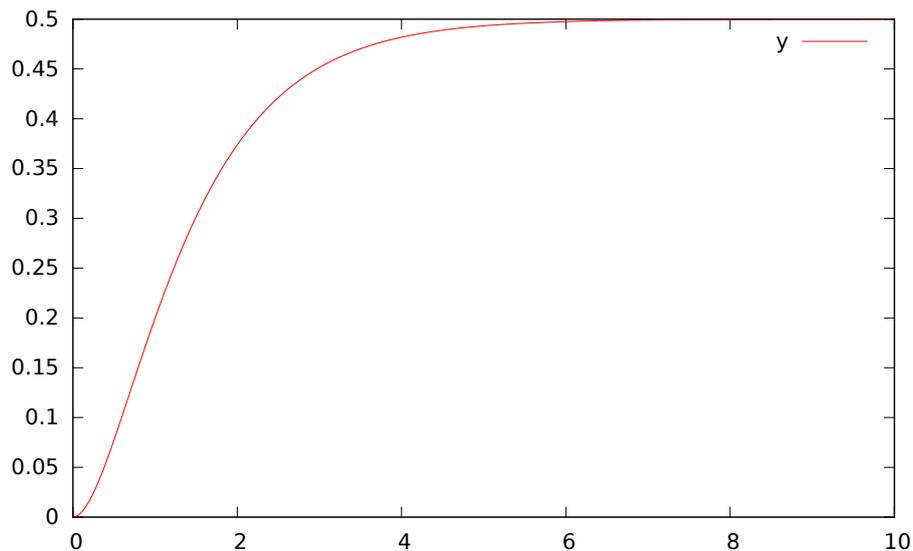


Figure 5.11: Characteristic equation with real negative roots.

```
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end ImgPosRoots;
```

```
>>> simulate(ImgPosRoots, stopTime=10)
record SimulationResult
  resultFile = "«DOCHOME»/ImgPosRoots_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.294388469,
  timeBackend = 0.002134831,
  timeSimCode = 0.049531459,
  timeTemplates = 0.00761279,
  timeCompile = 0.23101288199999999,
  timeSimulation = 0.0072636140000000001,
  timeTotal = 0.59201742299999999
end SimulationResult;
```

Warning:

Warning: The initial conditions are not fully specified. Use +d=initialization for more information.

The theory and application of Kalman filters is also explained in the interactive course material.

In reality noise is present in almost every physical system under study and therefore the concept of noise is also introduced in the course material, which is purely Modelica based.

5.4 OpenModelica Notebook Commands

OMNotebook currently supports the commands and concepts that are described in this section.

5.4.1 Cells

Everything inside an OMNotebook document is made out of cells. A cell basically contains a chunk of data. That data can be text, images, or other cells. OMNotebook has four types of cells: headercell, textcell, inputcell, and

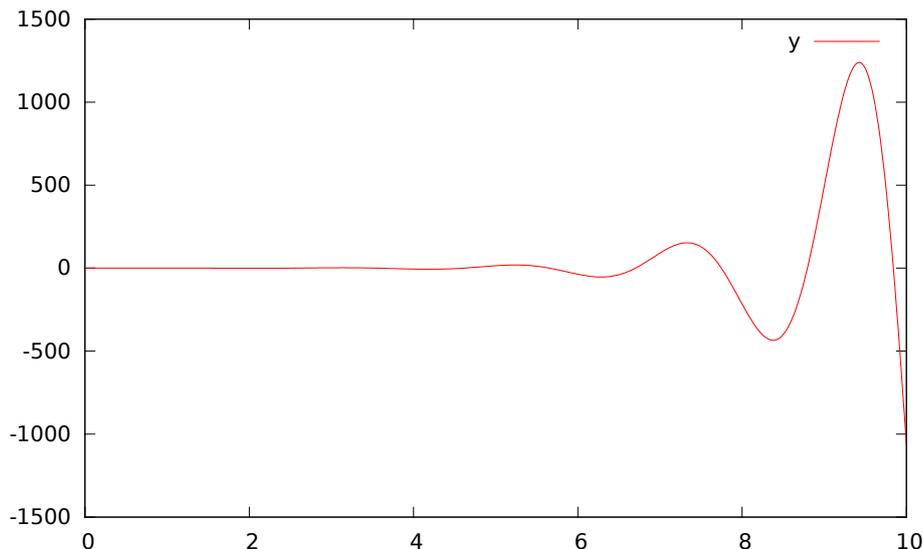


Figure 5.12: Characteristic equation with imaginary roots with positive real part.

groupcell. Cells are ordered in a tree structure, where one cell can be a parent to one or more additional cells. A tree view is available close to the right border in the notebook window to display the relation between the cells.

- **Textcell** – This cell type is used to display ordinary text and images. Each textcell has a style that specifies how text is displayed. The cell's style can be changed in the menu Format->Styles, example of different styles are: Text, Title, and Subtitle. The Textcell type also has support for following links to other notebook documents.
- **Inputcell** – This cell type has support for syntax highlighting and evaluation. It is intended to be used for writing program code, e.g. Modelica code. Evaluation is done by pressing the key combination Shift+Return or Shift+Enter. All the text in the cell is sent to OMC (OpenModelica Compiler/interpreter), where the text is evaluated and the result is displayed below the inputcell. By double-clicking on the cell marker in the tree view, the inputcell can be collapsed causing the result to be hidden.
- **Latexcell** – This cell type has support for evaluation of latex scripts. It is intended to be mainly used for writing mathematical equations and formulas for advanced documentation in OMNotebook. Each Latexcell supports a maximum of one page document output. To evaluate this cell, latex must be installed in your system. The users can copy and paste the latex scripts and start the evaluation. Evaluation is done by pressing the key combination Shift+Return or Shift+Enter or the green color eval button present in the toolbar. The script in the cell is sent to latex compiler, where it is evaluated and the output is displayed hiding the latex source. By double-clicking on the cell marker in the tree view, the latex source is displayed for further modification.
- **Groupcell** – This cell type is used to group together other cell. A groupcell can be opened or closed. When a groupcell is opened all the cells inside the groupcell are visible, but when the groupcell is closed only the first cell inside the groupcell is visible. The state of the groupcell is changed by the user double-clicking on the cell marker in the tree view. When the groupcell is closed the marker is changed and the marker has an arrow at the bottom.

5.4.2 Cursors

An OMNotebook document contains cells which in turn contain text. Thus, two kinds of cursors are needed for positioning, text cursor and cell cursor:

- **Textcursor** – A cursor between characters in a cell, appearing as a small vertical line. Position the cursor by clicking on the text or using the arrow buttons.

File Edit Cell Format Insert Window Help

loadModel(Modelica.Blocks)

model Tank
 Modelica.Blocks.Continuous.TransferFunction G(b={1/A},
 a={1,1/T},y_start(fixed=true)=1/A);
 Modelica.Blocks.Continuous.TransferFunction GStep(b={1/A}, a={1,1/T});
 parameter Real T = 15;
 parameter Real A = 5;
 Real u = if (time > 0 or time<0) then 0 else Modelica.Constants.inf;
 Real uStep = if (time > 0 or time<0) then 1 else 0;
equation
 G.u = if time > 0 then 0 else 1e10;
 GStep.u = uStep;
end Tank;

{Tank}

simulate(Tank,startTime=-1e-10,numberOfIntervals=500,stopTime=10);

plot({G.y,GStep.y})

true

Plot by OpenModelica

Ready Ln 8, Col 1

Figure 5.13: Step and pulse (weight function) response.

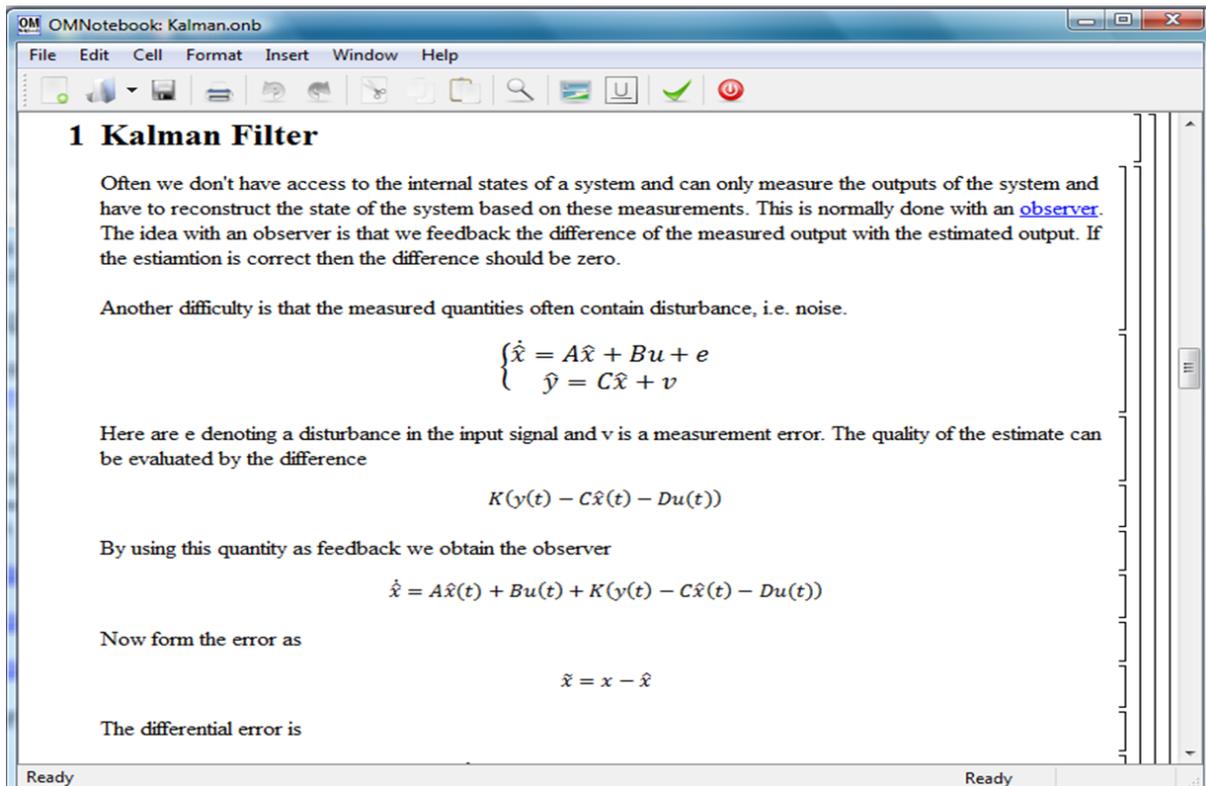


Figure 5.14: Theory background about Kalman filter.

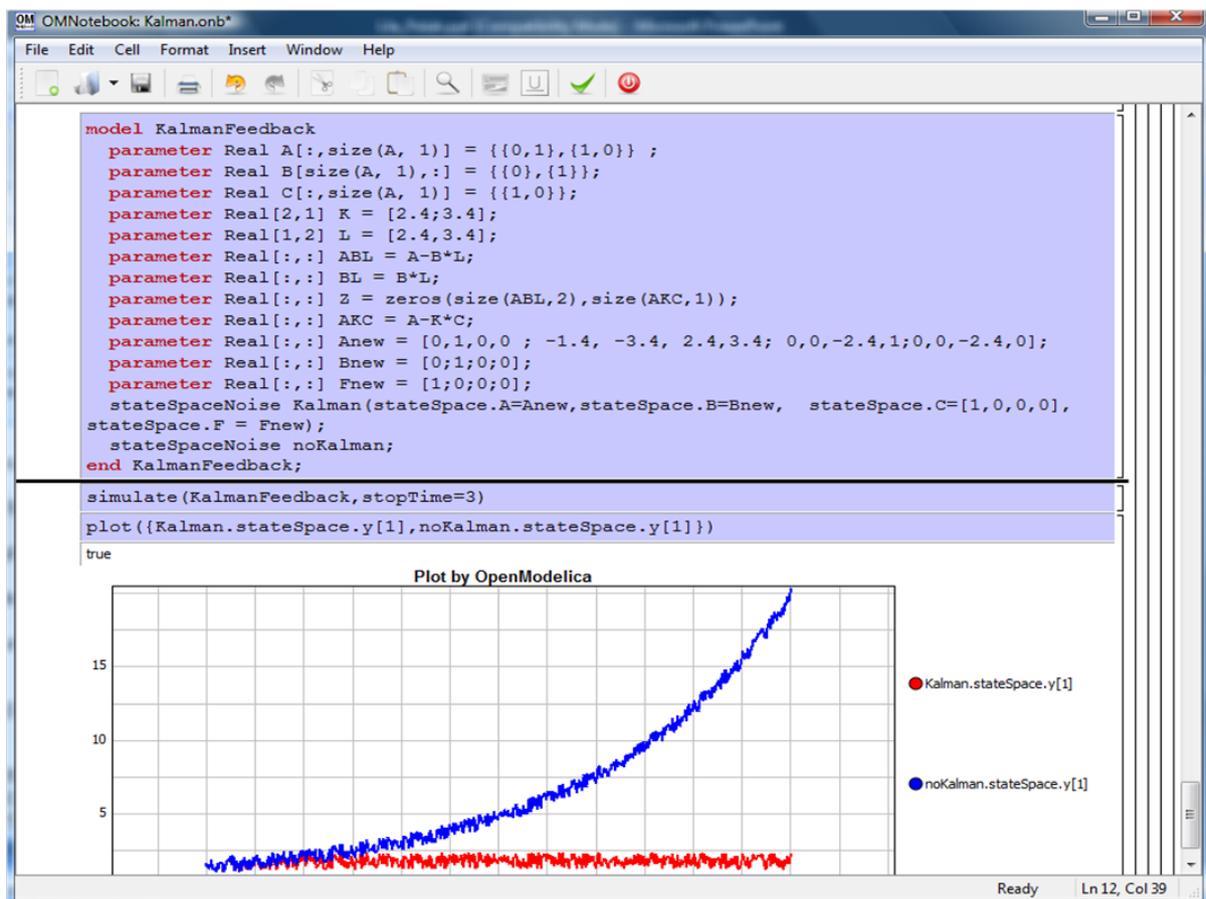


Figure 5.15: Comparison of a noisy system with feedback link in DrControl.

- **Cellcursor** – This cursor shows which cell currently has the input focus. It consists of two parts. The main cellcursor is basically just a thin black horizontal line below the cell with input focus. The cellcursor is positioned by clicking on a cell, clicking between cells, or using the menu item Cell->Next Cell or Cell->Previous Cell. The cursor can also be moved with the key combination Ctrl+Up or Ctrl+Down. The dynamic cellcursor is a short blinking horizontal line. To make this visible, you must click once more on the main cellcursor (the long horizontal line). NOTE: In order to paste cells at the cellcursor, the *dynamic cellcursor must be made active* by clicking on the main cellcursor (the horizontal line).

5.4.3 Selection of Text or Cells

To perform operations on text or cells we often need to select a range of characters or cells.

- **Select characters** – There are several ways of selecting characters, e.g. double-clicking on a word, clicking and dragging the mouse, or click followed by a shift-click at an adjacent position selects the text between the previous click and the position of the most recent shift-click.
- **Select cells** – Cells can be selected by clicking on them. Holding down Ctrl and clicking on the cell markers in the tree view allows several cells to be selected, one at a time. Several cells can be selected at once in the tree view by holding down the Shift key. Holding down Shift selects all cells between last selected cell and the cell clicked on. This only works if both cells belong to the same groupcell.

5.4.4 File Menu

The following file related operations are available in the file menu:

- **Create a new notebook** – A new notebook can be created using the menu File->New or the key combination Ctrl+N. A new document window will then open, with a new document inside.
- **Open a notebook** – To open a notebook use File->Open in the menu or the key combination Ctrl+O. Only files of the type .onb or .nb can be opened. If a file does not follow the OMNotebook format or the FullForm Mathematica Notebook format, a message box is displayed telling the user what is wrong. Mathematica Notebooks must be converted to fullform before they can be opened in OMNotebook.
- **Save a notebook** – To save a notebook use the menu item File->Save or File->Save As. If the notebook has not been saved before the save as dialog is shown and a filename can be selected. OMNotebook can only save in xml format and the saved file is not compatible with Mathematica. Key combination for save is Ctrl+S and for save as Ctrl+Shift+S. The saved file by default obtains the file extension .onb.
- **Print** – Printing a document to a printer is done by pressing the key combination Ctrl+P or using the menu item File->Print. A normal print dialog is displayed where the usually properties can be changed.
- **Import old document** – Old documents, saved with the old version of OMNotebook where a different file format was used, can be opened using the menu item File->Import->Old OMNotebook file. Old documents have the extension .xml.
- **Export text** – The text inside a document can be exported to a text document. The text is exported to this document without almost any structure saved. The only structure that is saved is the cell structure. Each paragraph in the text document will contain text from one cell. To use the export function, use menu item File->Export->Pure Text.
- **Close a notebook window** – A notebook window can be closed using the menu item File->Close or the key combination Ctrl+F4. Any unsaved changes in the document are lost when the notebook window is closed.
- **Quitting OMNotebook** – To quit OMNotebook, use menu item File->Quit or the key combination Ctrl+Q. This closes all notebook windows; users will have the option of closing OMC also. OMC will not automatically shutdown because other programs may still use it. Evaluating the command quit() has the same result as exiting OMNotebook.

5.4.5 Edit Menu

- **Editing cell text** – Cells have a set of of basic editing functions. The key combination for these are: Undo (Ctrl+Z), Redo (Ctrl+Y), Cut (Ctrl+X), Copy (Ctrl+C) and Paste (Ctrl+V). These functions can also be accessed from the edit menu; Undo (Edit->Undo), Redo (Edit->Redo), Cut (Edit->Cut), Copy (Edit->Copy) and Paste (Edit->Paste). Selection of text is done in the usual way by double-clicking, triple-clicking (select a paragraph), dragging the mouse, or using (Ctrl+A) to select all text within the cell.
- **Cut cell** – Cells can be cut from a document with the menu item Edit->Cut or the key combination Ctrl+X. The cut function will always cut cells if cells have been selected in the tree view, otherwise the cut function cuts text.
- **Copy cell** – Cells can be copied from a document with the menu item Edit->Copy or the key combination Ctrl+C. The copy function will always copy cells if cells have been selected in the tree view, otherwise the copy function copy text.
- **Paste cell** – To paste copied or cut cells the cell cursor must be selected in the location where the cells should be pasted. This is done by clicking on the cell cursor. Pasting cells is done from the menu Edit->Paste or the key combination Ctrl+V. If the cell cursor is selected the paste function will always paste cells. OMNotebook share the same application-wide clipboard. Therefore cells that have been copied from one document can be pasted into another document. Only pointers to the copied or cut cells are added to the clipboard, thus the cell that should be pasted must still exist. Consequently a cell can not be pasted from a document that has been closed.
- **Find** – Find text string in the current notebook, with the options match full word, match cell, search within closed cells. Short command Ctrl+F.
- **Replace** – Find and replace text string in the current notebook, with the options match full word, match cell, search+replace within closed cells. Short command Ctrl+H.
- **View expression** – Text in a cell is stored internally as a subset of HTML code and the menu item Edit->View Expression let the user switch between viewing the text or the internal HTML representation. Changes made to the HTML code will affect how the text is displayed.

5.4.6 Cell Menu

- **Add textcell** – A new textcell is added with the menu item Cell->Add Cell (previous cell style) or the key combination Alt+Enter. The new textcell gets the same style as the previous selected cell had.
- **Add inputcell** – A new inputcell is added with the menu item Cell->Add Inputcell or the key combination Ctrl+Shift+I.
- **Add latexcell** – A new latexcell is added with the menu item Cell->Add Latexcell or the key combination Ctrl+Shift+E.
- **Add groupcell** – A new groupcell is inserted with the menu item Cell->Groupcell or the key combination Ctrl+Shift+G. The selected cell will then become the first cell inside the groupcell.
- **Ungroup groupcell** – A groupcell can be ungrouped by selecting it in the tree view and using the menu item Cell->Ungroup Groupcell or by using the key combination Ctrl+Shift+U. Only one groupcell at a time can be ungrouped.
- **Split cell** – Splitting a cell is done with the menu item Cell->Split cell or the key combination Ctrl+Shift+P. The cell is splitted at the position of the text cursor.
- **Delete cell** – The menu item Cell->Delete Cell will delete all cells that have been selected in the tree view. If no cell is selected this action will delete the cell that have been selected by the cellcursor. This action can also be called with the key combination Ctrl+Shift+D or the key Del (only works when cells have been selected in the tree view).
- **Cellcursor** – This cell type is a special type that shows which cell that currently has the focus. The cell is basically just a thin black line. The cellcursor is moved by clicking on a cell or using the menu

item Cell->Next Cell or Cell->Previous Cell. The cursor can also be moved with the key combination Ctrl+Up or Ctrl+Down.

5.4.7 Format Menu

- **Textcell** – This cell type is used to display ordinary text and images. Each textcell has a style that specifies how text is displayed. The cells style can be changed in the menu Format->Styles, examples of different styles are: Text, Title, and Subtitle. The Textcell type also have support for following links to other notebook documents.
- **Text manipulation** – There are a number of different text manipulations that can be done to change the appearance of the text. These manipulations include operations like: changing font, changing color and make text bold, but also operations like: changing the alignment of the text and the margin inside the cell. All text manipulations inside a cell can be done on single letters, words or the entire text. Text settings are found in the Format menu. The following text manipulations are available in OMNotebook:

- > Font family
- > Font face (Plain, Bold, Italic, Underline)
- > Font size
- > Font stretch
- > Font color
- > Text horizontal alignment
- > Text vertical alignment
- > Border thickness
- > Margin (outside the border)
- > Padding (inside the border)

5.4.8 Insert Menu

- **Insert image** – Images are added to a document with the menu item Insert->Image or the key combination Ctrl+Shift+M. After an image has been selected a dialog appears, where the size of the image can be chosen. The images actual size is the default value of the image. OMNotebook stretches the image accordantly to the selected size. All images are saved in the same file as the rest of the document.
- **Insert link** – A document can contain links to other OMNotebook file or Mathematica notebook and to add a new link a piece of text must first be selected. The selected text make up the part of the link that the user can click on. Inserting a link is done from the menu Insert->Link or with the key combination Ctrl+Shift+L. A dialog window, much like the one used to open documents, allows the user to choose the file that the link refers to. All links are saved in the document with a relative file path so documents that belong together easily can be moved from one place to another without the links failing.

5.4.9 Window Menu

- **Change window** – Each opened document has its own document window. To switch between those use the Window menu. The window menu lists all titles of the open documents, in the same order as they were opened. To switch to another document, simple click on the title of that document.

5.4.10 Help Menu

- **About OMNotebook** – Accessing the about message box for OMNotebook is done from the menu Help->About OMNotebook.
- **About Qt** – To access the message box for Qt, use the menu Help->About Qt.
- **Help Text** – Opening the help text (document OMNotebookHelp.onb) for OMNotebook can be done in the same way as any OMNotebook document is opened or with the menu Help->Help Text. The menu item can also be triggered with the key F1.

5.4.11 Additional Features

- **Links** – By clicking on a link, OMNotebook will open the document that is referred to in the link.
- **Update link** – All links are stored with relative file path. Therefore OMNotebook has functions that automatically updating links if a document is resaved in another folder. Every time a document is saved, OMNotebook checks if the document is saved in the same folder as last time. If the folder has changed, the links are updated.
- **Evaluate whole Notebook** – All the cells present in the Notebook can be evaluated in one step by pressing the red color evalall button in the toolbar. The cells are evaluated in the same order as they are in the Notebook. However the latex cells cannot be evaluated by this feature.
- **Evaluate several cells** – Several input cells can be evaluated at the same time by selecting them in the treeview and then pressing the key combination Shift+Enter or Shift+Return. The cells are evaluated in the same order as they have been selected. If a group cell is selected all input cells in that group cell are evaluated, in the order they are located in the group cell.
- **Moving and Reordering cells in a Notebook** – It is possible to shift cells to a new position and change the hierarchical order of the document. This can be done by clicking the cell and press the Up and Down arrow button in the tool bar to move either Up or Down. The cells are moved one cell above or below. It is also possible to move a cell directly to a new position with one single click by pressing the red color bidirectional UpDown arrow button in the toolbar. To do this the user has to place the cell cursor to a position where the selected cells must be moved. After selecting the cell cursor position, select the cells you want to shift and press the bidirectional UpDown arrow button. The cells are shifted in the same order as they are selected. This is especially very useful when shifting a group cell.
- **Command completion** – Input cells have command completion support, which checks if the user is typing a command (or any keyword defined in the file commands.xml) and finish the command. If the user types the first two or three letters in a command, the command completion function fills in the rest. To use command completion, press the key combination Ctrl+Space or Shift+Tab. The first command that matches the letters written will then appear. Holding down Shift and pressing Tab (alternative holding down Ctrl and pressing Space) again will display the second command that matches. Repeated request to use command completion will loop through all commands that match the letters written. When a command is displayed by the command completion functionality any field inside the command that should be edited by the user is automatically selected. Some commands can have several of these fields and by pressing the key combination Ctrl+Tab, the next field will be selected inside the command. > Active Command completion: Ctrl+Space / Shift+Tab > Next command: Ctrl+Space / Shift+Tab > Next field in command: Ctrl+Tab'
- **Generated plot** – When plotting a simulation result, OMC uses the program Ptpplot to create a plot. From Ptpplot OMNotebook gets an image of the plot and automatically adds that image to the output part of an input cell. Like all other images in a document, the plot is saved in the document file when the document is saved.
- **Stylesheet** – OMNotebook follows the style settings defined in stylesheet.xml and the correct style is applied to a cell when the cell is created.
- **Automatic Chapter Numbering** – OMNotebook automatically numbers different chapter, subchapter, section and other styles. The user can specify which styles should have chapter numbers and which

level the style should have. This is done in the stylesheet.xml file. Every style can have a <chapter-Level> tag that specifies the chapter level. Level 0 or no tag at all, means that the style should not have any chapter numbering.

- **Scrollarea** – **Scrolling through a document can be done by using the** mouse wheel. A document can also be scrolled by moving the cell cursor up or down.
- **Syntax highlighter** – **The syntax highlighter runs in a separated** thread which speeds up the loading of large document that contains many Modelica code cells. The syntax highlighter only highlights when letters are added, not when they are removed. The color settings for the different types of keywords are stored in the file modelicacolors.xml. Besides defining the text color and background color of keywords, whether or not the keywords should be bold or/and italic can be defined.
- **Change indicator** – **A star (*) will appear behind the filename in** the title of notebook window if the document has been changed and needs saving. When the user closes a document that has some unsaved change, OMNotebook asks the user if he/she wants to save the document before closing. If the document never has been saved before, the save-as dialog appears so that a filename can be chosen for the new document.
- **Update menus** – **All menus are constantly updated so that only menu** items that are linked to actions that can be performed on the currently selected cell is enabled. All other menu items will be disabled. When a textcell is selected the Format menu is updated so that it indicates the text settings for the text, in the current cursor position.

5.5 References

Todo

Add these into extrarefs.bib and cite them somewhere

Eric Allen, Robert Cartwright, Brian Stoler. DrJava: A lightweight pedagogic environment for Java. In Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002) (Northern Kentucky – The Southern Side of Cincinnati, USA, February 27 – March 3, 2002).

Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done?. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006.

Eva-Lena Lengquist-Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. DrModelica – A Web-Based Teaching Environment for Modelica. In Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS'2003), available at www.scan-sims.org. Västerås, Sweden. September 18-19, 2003.

FUNCTIONAL MOCK-UP INTERFACE - FMI

The new standard for model exchange and co-simulation with Functional Mockup Interface (FMI) allows export of pre-compiled models, i.e., C-code or binary code, from a tool for import in another tool, and vice versa. The FMI standard is Modelica independent. Import and export works both between different Modelica tools, or between certain non-Modelica tools. OpenModelica supports FMI 1.0 & 2.0,

- Model Exchange
- Co-Simulation (under development)

6.1 FMI Export

To export the FMU use the OpenModelica command `translateModelFMU(ModelName)` from command line interface, OMSshell, OMNotebook or MDT. The export FMU command is also integrated with OMEdit. Select FMI > Export FMU the FMU package is generated in the current directory of omc. You can use the `cd()` command to see the current location. You can set which version of FMI to export through OMEdit settings, see section [FMI](#).

To export the bouncing ball example to an FMU, use the following commands:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.mo")
true
>>> translateModelFMU(BouncingBall)
"SimCode: The model BouncingBall has been translated to FMU"
>>> system("unzip -l BouncingBall.fmu | egrep -v 'sources|files' | tail -n+3 | grep -o '[A-Za-z._-]' | fold -w 80 -n 1 | tr -d '\n'")
0
```

Warning:

Warning: The initial conditions are not fully specified. Use `+d=initialization` for more information.

After the command execution is complete you will see that a file `BouncingBall.fmu` has been created. Its contents varies depending on the current platform. On the machine generating this documentation, the contents in [Listing 6.1](#) are generated (along with the C source code).

Listing 6.1: BouncingBall FMU contents

```
binaries/
binaries/linux64/
binaries/linux64/config.log
binaries/linux64/BouncingBall_FMU.libs
binaries/linux64/BouncingBall.so
modelDescription.xml
```

A log file for FMU creation is also generated named `ModelName_FMU.log`. If there are some errors while creating FMU they will be shown in the command line window and logged in this log file as well.

6.2 FMI Import

To import the FMU package use the OpenModelica command `importFMU`,

```
>>> list(OpenModelica.Scripting.importFMU, interfaceOnly=true)
function importFMU
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files. <default> will
  input Integer loglevel = 3 "loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;loglevel_warni
  input Boolean fullPath = false "When true the full output path is returned otherwise only the f
  input Boolean debugLogging = false "When true the FMU's debug output is printed.";
  input Boolean generateInputConnectors = true "When true creates the input connector pins.";
  input Boolean generateOutputConnectors = true "When true creates the output connector pins.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMU;
```

The command could be used from command line interface, OMShell, OMNotebook or MDT. The `importFMU` command is also integrated with OMEdit. Select FMI > Import FMU the FMU package is extracted in the directory specified by `workdir`, since the `workdir` parameter is optional so if its not specified then the current directory of `omc` is used. You can use the `cd()` command to see the current location.

The implementation supports FMI for Model Exchange 1.0 & 2.0 and FMI for Co-Simulation 1.0 stand-alone. The support for FMI Co-Simulation is still under development.

The FMI Import is currently a prototype. The prototype has been tested in OpenModelica with several examples. It has also been tested with example FMUs from FMUSDK and Dymola. A more fullfleged version for FMI Import will be released in the near future.

When importing the model into OMEdit, roughly the following commands will be executed:

```
>>> imported_fmu_mo_file:=importFMU("BouncingBall.fmu")
"BouncingBall_me_FMU.mo"
>>> loadFile(imported_fmu_mo_file)
true
```

The imported FMU can then be simulated like any normal model:

```
>>> simulate(BouncingBall_me_FMU, stopTime=3.0)
record SimulationResult
  resultFile = "«DOCHOME»/BouncingBall_me_FMU_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500, tolerance = 1e
  messages = "",
  timeFrontend = 0.028360559,
  timeBackend = 0.008139257,
  timeSimCode = 0.05437575200000001,
  timeTemplates = 0.019645804,
  timeCompile = 0.238398044,
  timeSimulation = 0.037323255,
  timeTotal = 0.386338554
end SimulationResult;
```

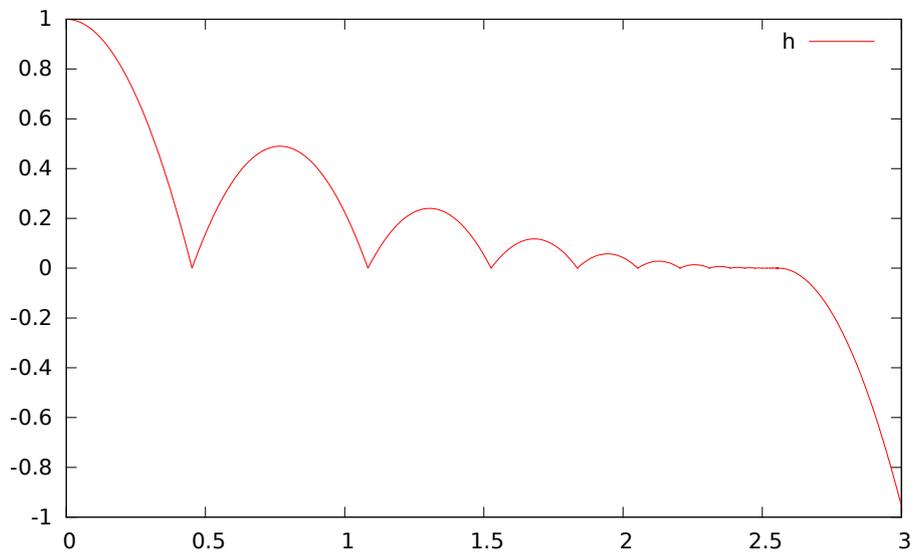


Figure 6.1: Height of the bouncing ball, simulated through an FMU.

OPTIMIZATION WITH OPENMODELICA

The following facilities for model-based optimization are provided with OpenModelica:

- **Builtin Dynamic Optimization with OpenModelica and IpOpt using** dynamic optimization is the recommended way of performing dynamic optimization with OpenModelica.
- **Dynamic Optimization with OpenModelica and CasADi.** Use this if you want to employ the CasADi tool for dynamic optimization.
- **Classical Parameter Sweep Optimization using OMOptim.** Use this if you have a static optimization problem.

7.1 Builtin Dynamic Optimization with OpenModelica and IpOpt

Note: this is a very short preliminary description which soon will be considerably improved.

OpenModelica provides builtin dynamic optimization of models by using the powerful symbolic machinery of the OpenModelica compiler for more efficient and automatic solution of dynamic optimization problems.

The builtin dynamic optimization allows users to define optimal control problems (OCP) using the Modelica language for the model and the optimization language extension called Optimica (currently partially supported) for the optimization part of the problem. This is used to solve the underlying dynamic optimization model formulation using collocation methods, using a single execution instead of multiple simulations as in the parameter-sweep optimization described in section *Parameter Sweep Optimization using OMOptim*.

For more detailed information regarding background and methods, see [BOR+12][RBB+14]

7.2 Compiling the Modelica code

Before starting the optimization the model should be symbolically instantiated by the compiler in order to get a single flat system of equations. The model variables should also be scalarized. The compiler frontend performs this, including syntax checking, semantics and type checking, simplification and constant evaluation etc. are applied. Then the complete flattened model can be used for initialization, simulation and last but not least for model-based dynamic optimization.

The OpenModelica command `optimize(ModelName)` from OMShell, OMNotebook or MDT runs immediately the optimization. The generated result file can be read in and visualized with OMEdit or within OMNotebook.

7.3 An Example

In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by a numerical algorithm. It includes several constructs including a new specialized class `Optimization`, a constraint section, `startTime`, `finalTime` etc. See the optimal control problem for batch reactor model below.

Create a new file named *BatchReactor.mo* and save it in your working directory. Notice that this model contains both the dynamic system to be optimized and the optimization specification.

Once we have formulated the underlying optimal control problems, we can run the optimization by using OMSHELL, OMNotebook, MDT, OMEdit using command line terminals similar to the options described below:

```
>>> setCommandLineOptions("+g=Optimica");
```

Listing 7.1: BatchReactor.mo

```
optimization BatchReactor(objective=-x2(finalTime), startTime = 0, finalTime =1)
  Real x1(start =1, fixed=true, min=0, max=1);
  Real x2(start =0, fixed=true, min=0, max=1);
  input Real u(min=0, max=5);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;
```

```
optimization nmpcBatchReactor(objective=-x2)
  extends BatchReactor;
end nmpcBatchReactor;
```

```
>>> optimize(nmpcBatchReactor, numberOfIntervals=16, stopTime=1, tolerance=1e-8)
record SimulationResult
  resultFile = "«DOCHOME»/nmpcBatchReactor_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 16, tolerance = 1e-8",
  messages = "
Optimizer Variables
=====
State[0]:x1(start = 1, nominal = 1, min = 0, max = 1, init = 1)
State[1]:x2(start = 0, nominal = 1, min = 0, max = 1, init = 0)
Input[2]:u(start = 0, nominal = 5, min = 0, max = 5)
-----
number of nonlinear constraints: 0
=====

*****
This program contains Ipopt, a library for large-scale nonlinear optimization.
Ipopt is released as open source code under the Eclipse Public License (EPL).
For more information visit http://projects.coin-or.org/Ipopt
*****

",
  timeFrontend = 0.07241109000000001,
  timeBackend = 0.009763674,
  timeSimCode = 0.055686138,
  timeTemplates = 0.005327275,
  timeCompile = 0.22989095399999999,
  timeSimulation = 0.028461984,
  timeTotal = 0.401639444
end SimulationResult;
```

The control and state trajectories of the optimization results:

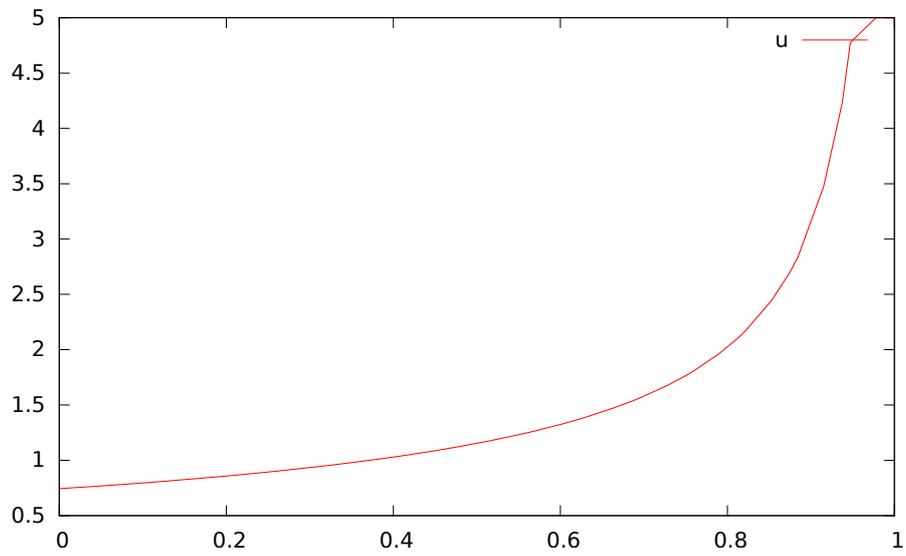


Figure 7.1: Optimization results for Batch Reactor model – input variables.

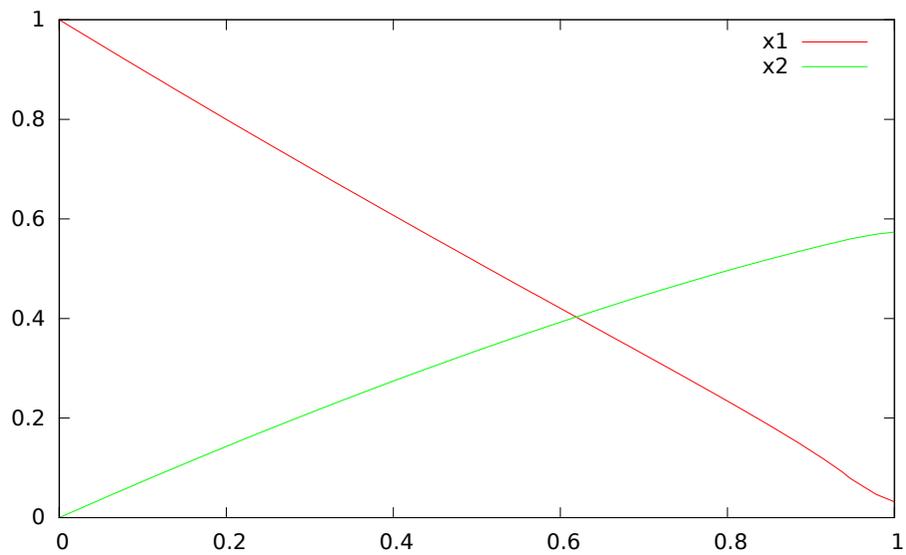


Figure 7.2: Optimization results for Batch Reactor model – state variables.

7.4 Different Options for the Optimizer IPOPT

Table 7.1: New meanings of the usual simulation options for Ipopt.

numberOfIntervals		collocation intervals
startTime, stopTime		time horizon
tolerance = 1e-8	e.g. 1e-8	solver tolerance
simflags	all run/debug options	

Table 7.2: New simulation options for Ipopt.

-lv	LOG_IPOPT	console output
-ipopt_hesse	CONST,BFGS,NUM	hessian approximation
-ipopt_max_iter	number e.g. 10	maximal number of iteration for ipopt
externalInput.csv		input guess

7.5 Dynamic Optimization with OpenModelica and CasADi

OpenModelica coupling with CasADi supports dynamic optimization of models by OpenModelica exporting the optimization problem to CasADi which performs the optimization. In order to convey the dynamic system model information between Modelica and CasADi, we use an XML-based model exchange format for differential-algebraic equations (DAE). OpenModelica supports export of models written in Modelica and the Optimization language extension using this XML format, while CasADi supports import of models represented in this format. This allows users to define optimal control problems (OCP) using Modelica and Optimization language specifications, and solve the underlying model formulation using a range of optimization methods, including direct collocation and direct multiple shooting.

7.5.1 Compiling the Modelica code

Before exporting a model to XML, the model should be symbolically instantiated by the compiler in order to get a single flat system of equations. The model variables should also be scalarized. The compiler frontend performs this, including syntax checking, semantics and type checking, simplification and constant evaluation etc. are applied. Then the complete flattened model is exported to XML code. The exported XML document can then be imported to CasADi for model-based dynamic optimization.

The OpenModelica command `translateModelXML(ModelName)` from OMSHELL, OMNotebook or MDT exports the XML. The export XML command is also integrated with OMEdit. Select XML > Export XML the XML document is generated in the current directory of omc. You can use the `cd()` command to see the current location. After the command execution is complete you will see that a file `ModelName.xml` has been exported.

Assuming that the model is defined in the `modelName.mo`, the model can also be exported to an XML code using the following steps from the terminal window:

- Go to the path where your model file found
- Run command `omc +g=Optimica +simCodeTarget=XML Model.mo`

7.5.2 An example

In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by a numerical algorithm. It

includes several constructs including a new specialized class optimization, a constraint section, startTime, final-Time etc. See the optimal control problem for batch reactor model below.

Create a new file named *BatchReactor.mo* and save it in your working directory. Notice that this model contains both the dynamic system to be optimized and the optimization specification.

```
>>> list (BatchReactor)
optimization BatchReactor
  Real x1(start = 1, fixed = true, min = 0, max = 1);
  Real x2(start = 0, fixed = true, min = 0, max = 1);
  input Real u(min = 0, max = 5);
equation
  der(x1) = -(u + u ^ 2 / 2) * x1;
  der(x2) = u * x1;
end BatchReactor;
```

Once we have formulated the underlying optimal control problems, we can export the XML by using OMSHELL, OMNotebook, MDT, OMEdit or command line terminals which are described in Section *XML Import to CasADi via OpenModelica Python Script*.

To export XML, we set the simulation target to XML:

```
>>> translateModelXML (BatchReactor)
```

Error:

Error: Internal error - BackendDAE.incidenceRow failed for equation: \$OMC\$objectMayerTerm := -x2(finalTime)

Error: Internal error BackendDAEUtil.incidenceMatrix failed.

Error: pre-optimization module removeEqualFunctionCalls (simulation) failed.

Error: Internal error SimCode: The model BatchReactor could not be translated to XML

Error: Internal error - BackendDAE.incidenceRow failed for equation: \$OMC\$objectMayerTerm := -x2(finalTime)

Error: Internal error BackendDAEUtil.incidenceMatrix failed.

Error: pre-optimization module removeEqualFunctionCalls (simulation) failed.

Error: Internal error SimCode: The model BatchReactor could not be translated to XML

This will generate an XML file named *batchreactorxml (batchreactorxml)* that contains a symbolic representation of the optimal control problem and can be inspected in a standard XML editor.

7.5.3 XML Import to CasADi via OpenModelica Python Script

The symbolic optimal control problem representation (or just model description) contained in *BatchReactor.xml* can be imported into CasADi in the form of the *SymbolicOCP* class via *OpenModelica* python script.

The *SymbolicOCP* class contains symbolic representation of the optimal control problem designed to be general and allow manipulation. For a more detailed description of this class and its functionalities, we refer to the API documentation of CasADi.

The following step compiles the model to an XML format, imports to CasADi and solves an optimization problem in windows PowerShell:

1. Create a new file named *BatchReactor.mo* and save it in your working directory.

E.g. C:\OpenModelica1.9.2\share\casadi\testmodel

1. Perform compilation and generate the XML file

- (a) Go to your working directory

E.g. cd C:\OpenModelica1.9.2\share\casadi\testmodel

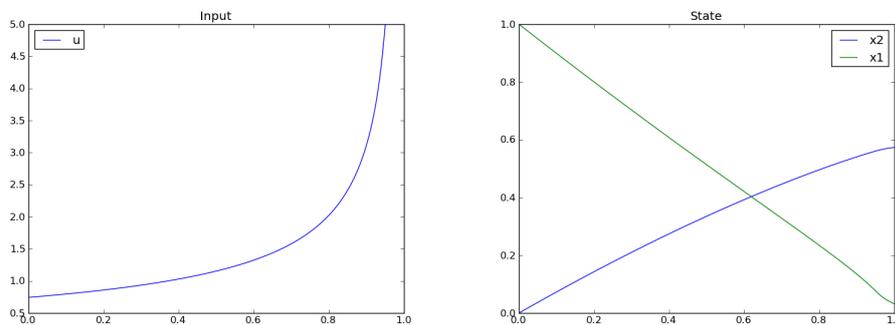
1. Go to omc path from working directory and run the following command

E.g. ..\..\bin\omc +s +g=Optimica +simCodeTarget=XML BatchReactor.mo

3. Run defaultStart.py python script from OpenModelica optimization directory

E.g. Python.exe ..\share\casadi\scripts defaultStart.py BatchReactor.xml

The control and state trajectories of the optimization results are shown below:



7.6 Parameter Sweep Optimization using OMOptim

OMOptim is a tool for parameter sweep design optimization of Modelica models. By optimization, one should understand a procedure which minimizes/maximizes one or more objective functions by adjusting one or more parameters. This is done by the optimization algorithm performing a parameter sweep, i.e., systematically adjusting values of selected parameters and running a number of simulations for different parameter combinations to find a parameter setting that gives an optimal value of the goal function.

OMOptim 0.9 contains meta-heuristic optimization algorithms which allow optimizing all sorts of models with following functionalities:

- One or several objectives optimized simultaneously
- One or several parameters (integer or real variables)

However, the user must be aware of the large number of simulations an optimization might require.

7.6.1 Preparing the Model

Before launching OMOptim, one must prepare the model in order to optimize it.

Parameters

An optimization parameter is picked up from all model variables. The choice of parameters can be done using the OMOptim interface.

For all intended parameters, please note that:

- **The corresponding variable is constant during all simulations.** The OMOptim optimization in version 0.9 only concerns static parameters' optimization *i.e.* values found for these parameters will be constant during all simulation time.
- **The corresponding variable should play an input role in the model** *i.e.* its modification influences model simulation results.

Constraints

If some constraints should be respected during optimization, they must be defined in the Modelica model itself.

For instance, if mechanical stress must be less than 5 N.m⁻², one should write in the model:

```
assert(mechanicalStress < 5, "Mechanical stress too high");
```

If during simulation, the variable *mechanicalStress* exceeds 5 N.m⁻², the simulation will stop and be considered as a failure.

Objectives

As parameters, objectives are picked up from model variables. Objectives' values are considered by the optimizer at the *final time*.

7.6.2 Set problem in OMOptim

Launch OMOptim

OMOptim can be launched using the executable placed in OpenModelicaInstallationDirectory/bin/ OMOptim/OMOptim.exe. Alternately, choose OpenModelica > OMOptim from the start menu.

Create a new project

To create a new project, click on menu File -> New project

Then set a name to the project and save it in a dedicated folder. The created file created has a .min extension. It will contain information regarding model, problems, and results loaded.

Load models

First, you need to load the model(s) you want to optimize. To do so, click on *Add .mo* button on main window or select menu *Model -> Load Mo file...*

When selecting a model, the file will be loaded in OpenModelica which runs in the background.

While OpenModelica is loading the model, you could have a frozen interface. This is due to multi-threading limitation but the delay should be short (few seconds).

You can load as many models as you want.

If an error occurs (indicated in log window), this might be because:

- Dependencies have not been loaded before (e.g. modelica library)
- Model use syntax incompatible with OpenModelica.

Dependencies

OMOptim should detect dependencies and load corresponding files. However, if some errors occur, please load by yourself dependencies. You can also load Modelica library using *Model->Load Modelica library*.

When the model correctly loaded, you should see a window similar to [Figure 7.3](#).

Create a new optimization problem

Problem->Add Problem->Optimization

A dialog should appear. Select the model you want to optimize. Only Model can be selected (no Package, Component, Block...).

A new form will be displayed. This form has two tabs. One is called Variables, the other is called Optimization.

List of Variables is Empty

If variables are not displayed, right click on model name in model hierarchy, and select *Read variables*.

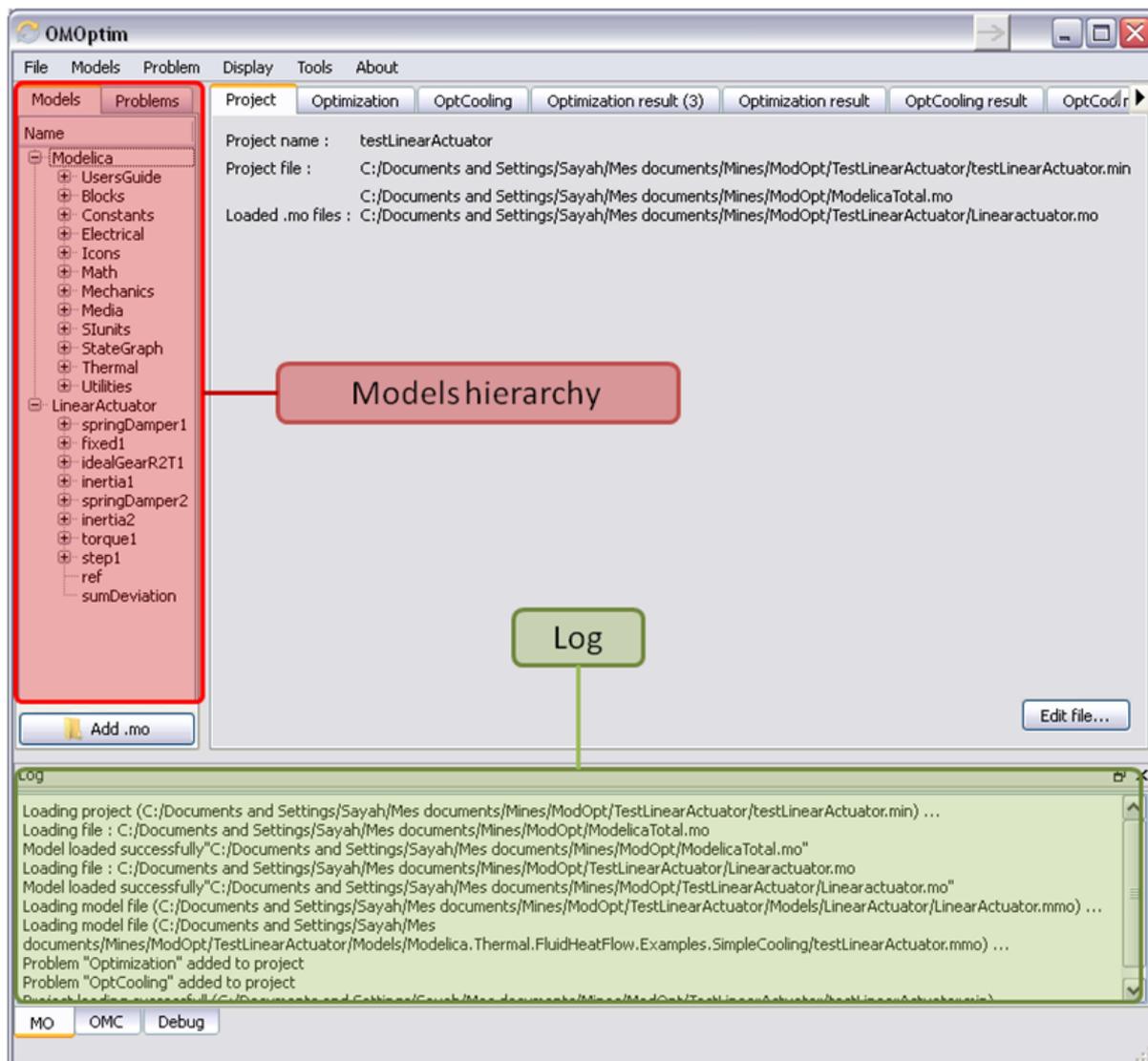


Figure 7.3: OMOptim window after having loaded model.

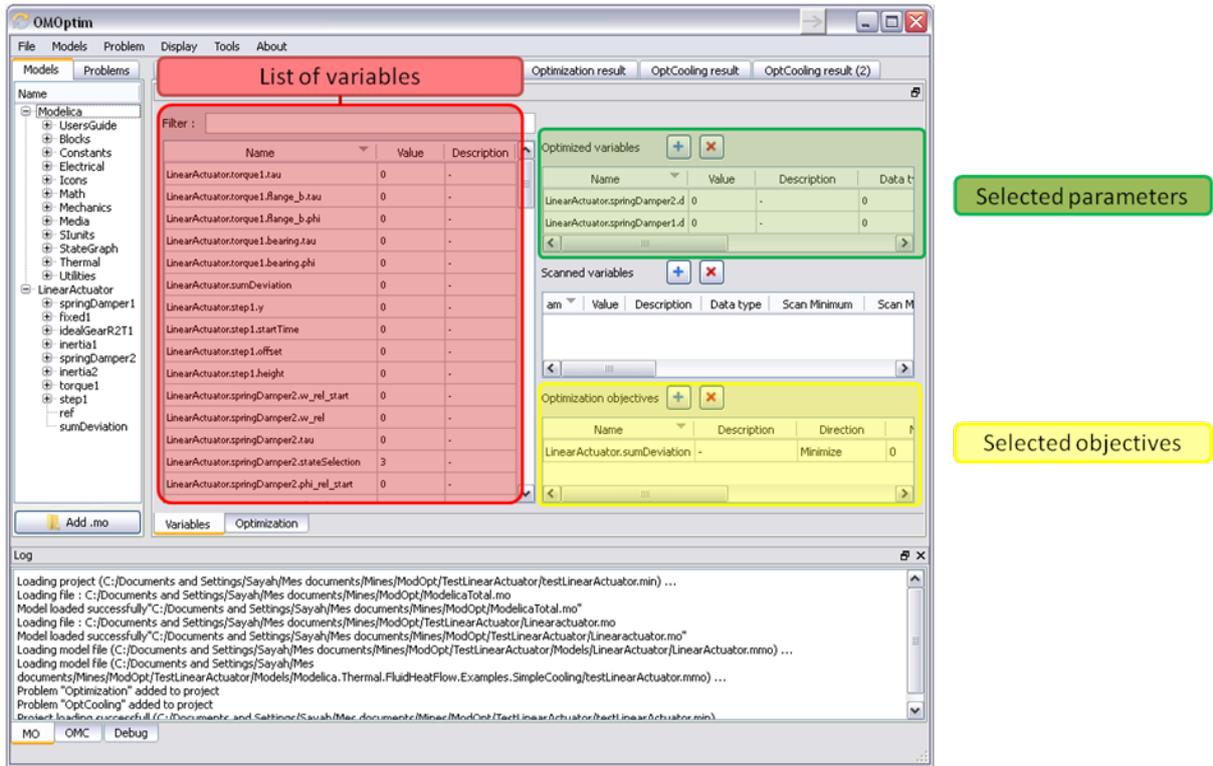


Figure 7.4: Forms for defining a new optimization problem.

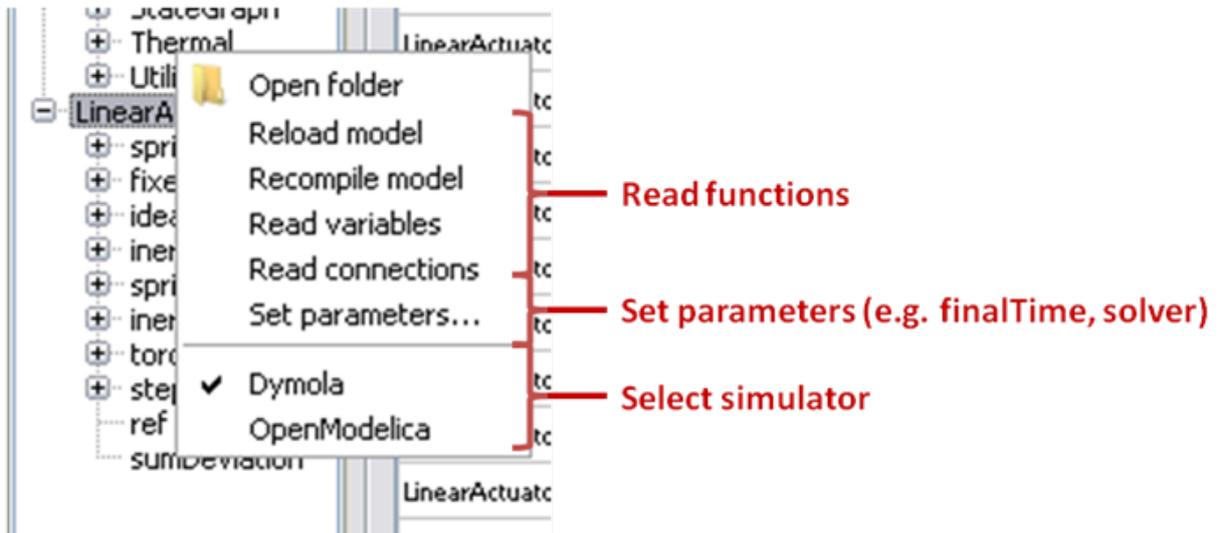


Figure 7.5: Selecting read variables, set parameters, and selecting simulator.

Select Optimized Variables

To set optimization, we first have to define the variables the optimizer will consider as free *i.e.* those that it should find best values of. To do this, select in the left list, the variables concerned. Then, add them to *Optimized variables* by clicking on corresponding button (+).

For each variable, you must set minimum and maximum values it can take. This can be done in the *Optimized variables* table.

Select objectives

Objectives correspond to the final values of chosen variables. To select these last, select in left list variables concerned and click + button of *Optimization objectives* table.

For each objective, you must:

- **Set minimum and maximum values it can take.** If a configuration does not respect these values, this configuration won't be considered. You also can set minimum and maximum equals to “-“ : it will then
- Define whether objective should be minimized or maximized.

This can be done in the *Optimized variables* table.

Select and configure algorithm

After having selected variables and objectives, you should now select and configure optimization algorithm. To do this, click on *Optimization* tab.

Here, you can select optimization algorithm you want to use. In version 0.9, OMOptim offers three different genetic algorithms. Let's for example choose SPEA2Adapt which is an auto-adaptative genetic algorithm.

By clicking on *parameters...* button, a dialog is opened allowing defining parameters. These are:

- **Population size: this is the number of configurations kept after a** generation. If it is set to 50, your final result can't contain more than 50 different points.
- **Off spring rate: this is the number of children per adult obtained** after combination process. If it is set to 3, each generation will contain 150 individual (considering population size is 50).
- **Max generations: this number defines the number of generations** after which optimization should stop. In our case, each generation corresponds to 150 simulations. Note that you can still stop optimization while it is running by clicking on *stop* button (which will appear once optimization is launched). Therefore, you can set a really high number and still stop optimization when you want without losing results obtained until there.
- **Save frequency: during optimization, best configurations can be** regularly saved. It allows to analyze evolution of best configurations but also to restart an optimization from previously obtained results. A Save Frequency parameter set to 3 means that after three generations, a file is automatically created containing best configurations. These files are named *iteration1.sav*, *iteration2.sav* and are store in *Temp* directory, and moved to *SolvedProblems* directory when optimization is finished.
- **ReinitStdDev: this is a specific parameter of EAAdapt1. It defines** whether standard deviation of variables should be reinitialized. It is used only if you start optimization from previously obtained configurations (using *Use start file* option). Setting it to yes (1) will, in most of cases, lead to a spread research of optimized configurations, forgetting parameters' variations' reduction obtained in previous optimization.

Use start file

As indicated before, it is possible to pursue an optimization finished or stopped. To do this, you must enable *Use start file* option and select file from which optimization should be started. This file is an *iteration_.sav* file created

in previous optimization. It is stored in corresponding *SolvedProblems* folder (*iteration10.sav* corresponds to the tenth generation of previous optimization).

***Note that this functionality can only work with same variables and objectives*.** However, minimum, maximum of variables and objectives can be changed before pursuing an optimization.

Launch

You can now launch Optimization by clicking *Launch* button.

Stopping Optimization

Optimization will be stopped when the generation counter will reach the generation number defined in parameters. However, you can still stop the optimization while it is running without losing obtained results. To do this, click on *Stop* button. Note that this will not immediately stop optimization: it will first finish the current generation.

This stop function is especially useful when optimum points do not vary any more between generations. This can be easily observed since at each generation, the optimum objectives values and corresponding parameters are displayed in log window.

7.6.3 Results

The result tab appear when the optimization is finished. It consists of two parts: a table where variables are displayed and a plot region.

Obtaining all Variable Values

During optimization, the values of optimized variables and objectives are memorized. The others are not. To get these last, you must recomputed corresponding points. To achieve this, select one or several points in point's list region and click on *recompute*.

For each point, it will simulate model setting input parameters to point corresponding values. All values of this point (including those which are not optimization parameters neither objectives).

7.6.4 Window Regions in OMOptim GUI

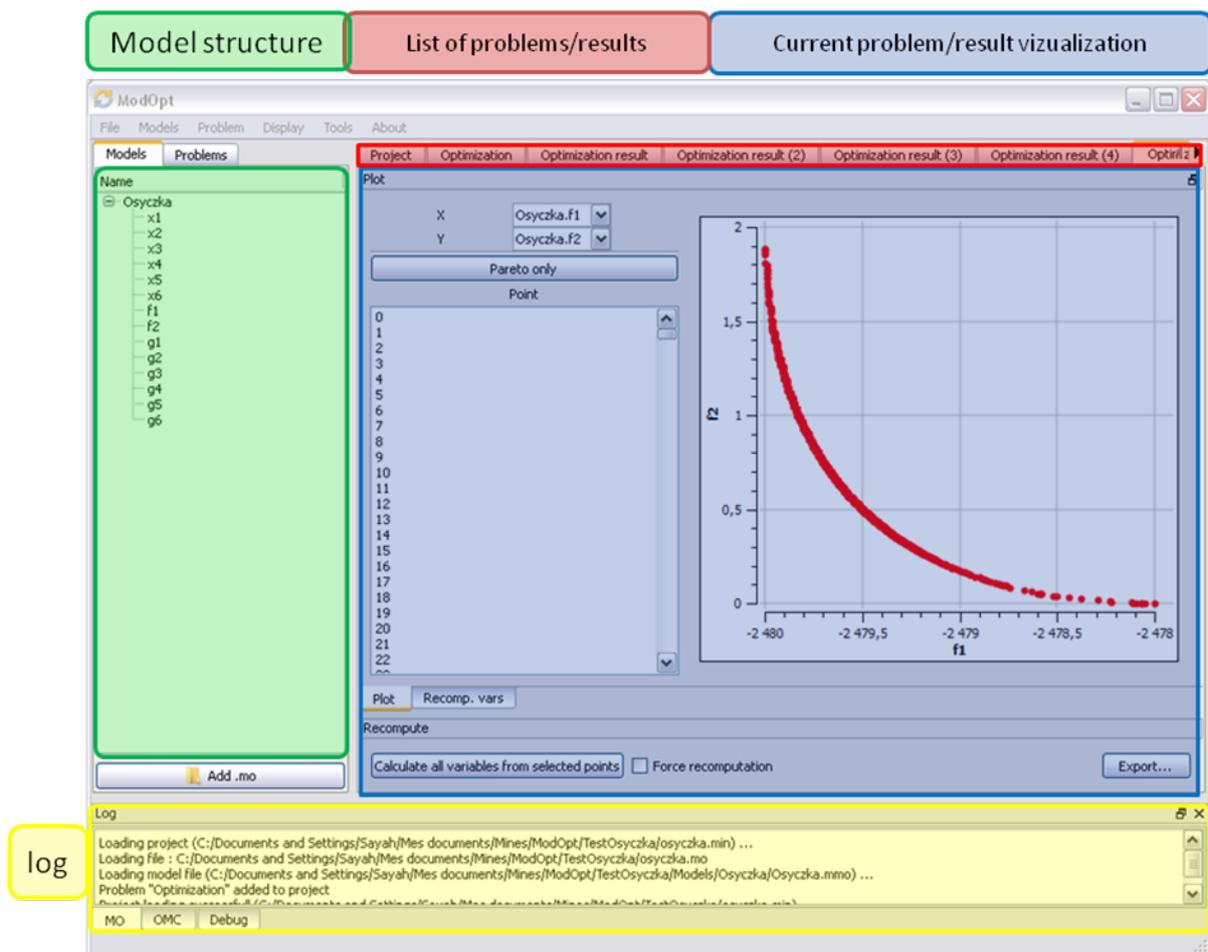


Figure 7.6: Window regions in OMOptim GUI.

MDT – THE OPENMODELICA DEVELOPMENT TOOLING ECLIPSE PLUGIN

8.1 Introduction

The Modelica Development Tooling (MDT) Eclipse Plugin as part of OMDev – The OpenModelica Development Environment integrates the OpenModelica compiler with Eclipse. MDT, together with the OpenModelica compiler, provides an environment for working with Modelica and MetaModelica development projects. This plugin is primarily intended for tool developers rather than application Modelica modelers.

The following features are available:

- Browsing support for Modelica projects, packages, and classes
- Wizards for creating Modelica projects, packages, and classes
- Syntax color highlighting
- Syntax checking
- Browsing of the Modelica Standard Library or other libraries
- Code completion for class names and function argument lists
- Goto definition for classes, types, and functions
- Displaying type information when hovering the mouse over an identifier.

8.2 Installation

The installation of MDT is accomplished by following the below installation instructions. These instructions assume that you have successfully downloaded and installed Eclipse (<http://www.eclipse.org>).

The latest installation instructions are available through the [OpenModelica Trac](#).

1. Start Eclipse
2. Select Help->Software Updates->Find and Install... from the menu
3. Select 'Search for new features to install' and click 'Next'
4. Select 'New Remote Site...'
5. Enter 'MDT' as name and <http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/MDT> as URL and click 'OK'
6. Make sure 'MDT' is selected and click 'Finish'
7. In the updates dialog select the 'MDT' feature and click 'Next'
8. Read through the license agreement, select 'I accept...' and click 'Next'
9. Click 'Finish' to install MDT

8.3 Getting Started

8.3.1 Configuring the OpenModelica Compiler

MDT needs to be able to locate the binary of the compiler. It uses the environment variable OPENMODELICAHOME to do so.

If you have problems using MDT, make sure that OPENMODELICAHOME is pointing to the folder where the OpenModelica Compiler is installed. In other words, OPENMODELICAHOME must point to the folder that contains the Open Modelica Compiler (OMC) binary. On the Windows platform it's called omc.exe and on Unix platforms it's called omc.

8.3.2 Using the Modelica Perspective

The most convenient way to work with Modelica projects is to use the Modelica perspective. To switch to the Modelica perspective, choose the Window menu item, pick Open Perspective followed by Other... Select the Modelica option from the dialog presented and click OK..

8.3.3 Selecting a Workspace Folder

Eclipse stores your projects in a folder called a workspace. You need to choose a workspace folder for this session, see [Figure 8.1](#).

8.3.4 Creating one or more Modelica Projects

To start a new project, use the New Modelica Project Wizard. It is accessible through File->New-> Modelica Project or by right-clicking in the Modelica Projects view and selecting New->Modelica Project.

You need to disable automatic build for the project(s) ([Figure 8.3](#)).

Repeat the procedure for all the projects you need, e.g. for the exercises described in the MetaModelica users guide: 01_experiment, 02a_exp1, 02b_exp2, 03_assignment, 04a_assigntwoctype, etc.

NOTE: Leave open only the projects you are working on! Close all the others!

8.3.5 Building and Running a Project

After having created a project, you eventually need to build the project ([Figure 8.4](#)).

The build options are the same as the make targets: you can build, build from scratch (clean), or run simulations depending on how the project is setup. See [Figure 8.5](#) for an example of how omc can be compiled (make omc builds OMC).

8.3.6 Switching to Another Perspective

If you need, you can (temporarily) switch to another perspective, e.g. to the Java perspective for working with an OpenModelica Java client as in [Figure 8.7](#).

8.3.7 Creating a Package

To create a new package inside a Modelica project, select File->New->Modelica Package. Enter the desired name of the package and a description of what it contains. Note: for the exercises we already have existing packages.

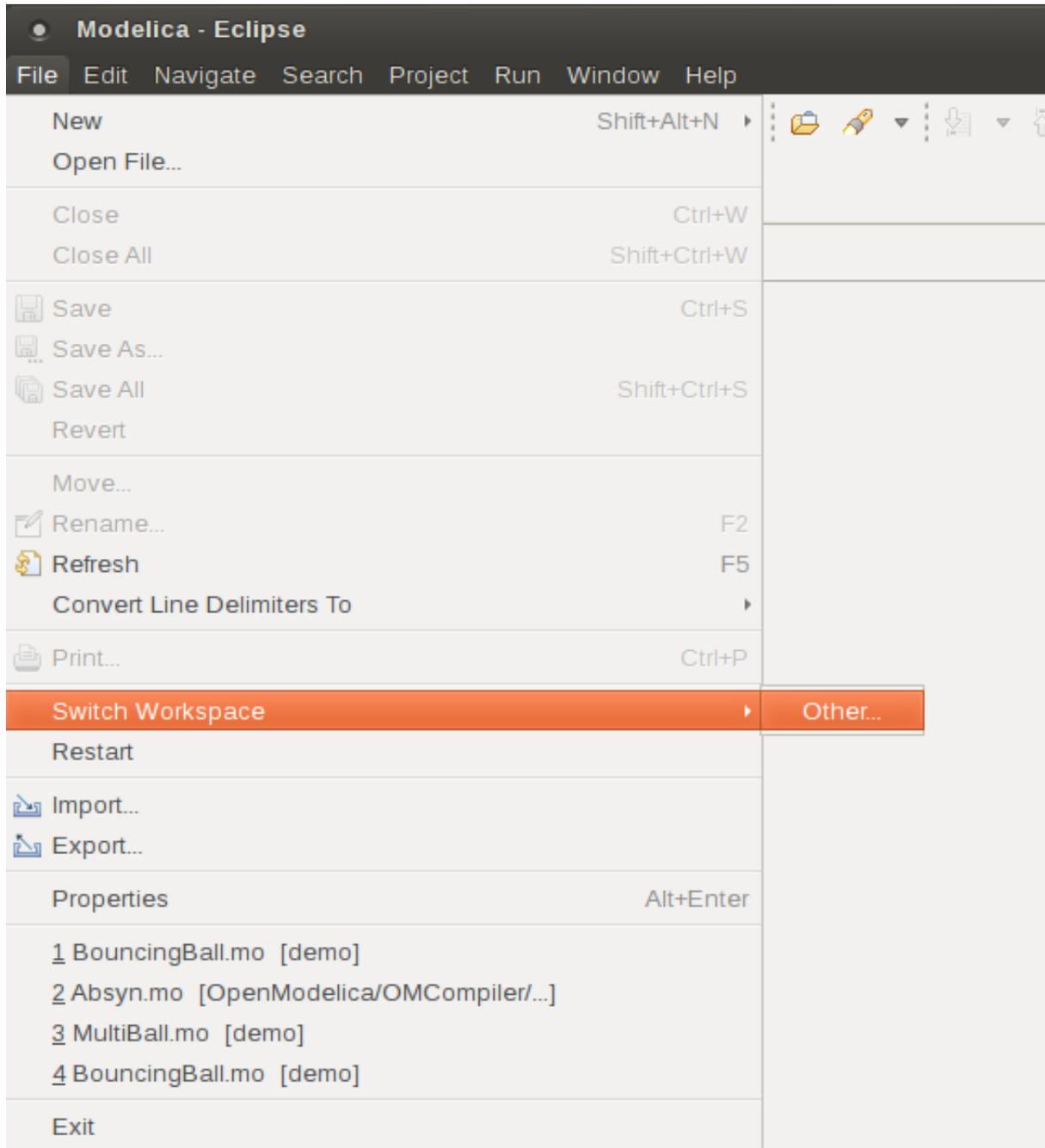


Figure 8.1: Eclipse Setup – Switching Workspace.

Create a Modelica project

Create a Modelica project in the workspace.



Project name:



Cancel

Finish

Figure 8.2: Eclipse Setup – creating a Modelica project in the workspace.

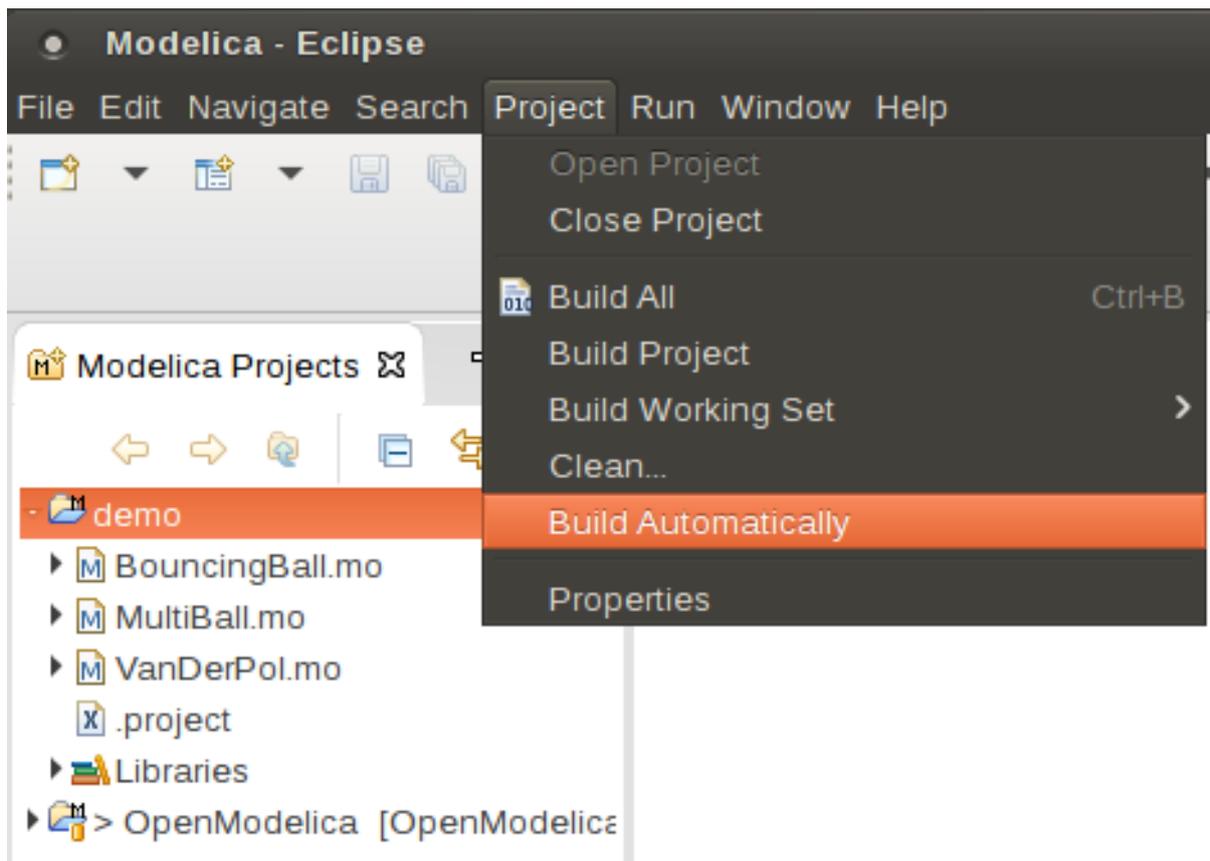


Figure 8.3: Eclipse Setup – disable automatic build for the projects.

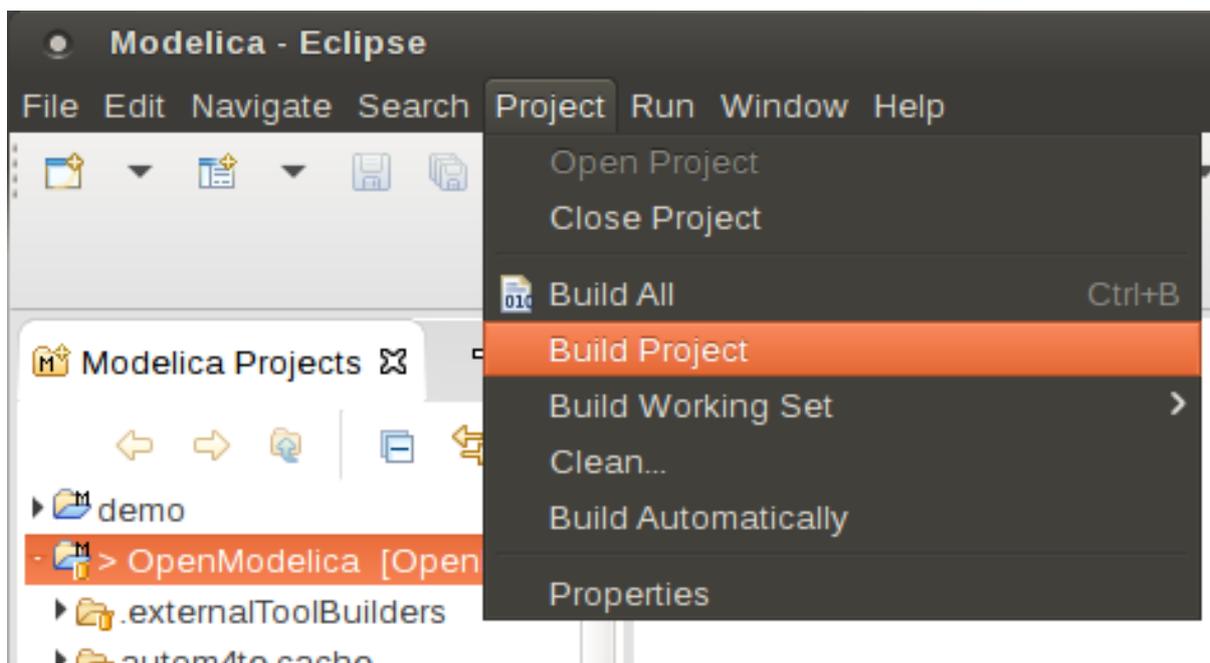


Figure 8.4: Eclipse MDT – Building a project.

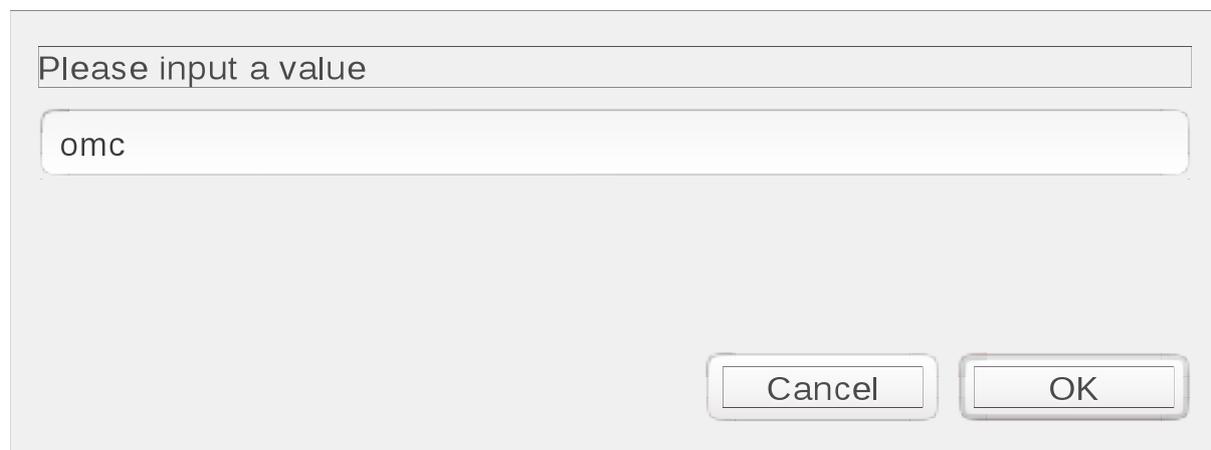


Figure 8.5: Eclipse – building a project.

```

<terminated> make [Program] /usr/bin/make
/usr/bin/make -f Makefile --no-print-directory -C /home/marsj/OpenModelica/OMCompiler/
make[5]: Nothing to be done for 'all'.
/usr/bin/make -f Makefile --no-print-directory -C /home/marsj/OpenModelica/OMCompiler/
make[5]: Nothing to be done for 'all'.
/usr/bin/make -f Makefile Makefile.sources
make[4]: 'Makefile.sources' is up to date.
/usr/bin/make -f Makefile interfaces INCLUDESOURCES=1
/home/marsj/OpenModelica/build/bin/omc +n=1 build/Absyn.stamp.mos
/usr/bin/make -f Makefile Makefile.depends INCLUDESOURCES=1
make[4]: 'Makefile.depends' is up to date.
/usr/bin/make -f Makefile generate-files INCLUDESOURCES=1 INCLUDEDDEPENDS=1
/home/marsj/OpenModelica/build/bin/omc +n=1 build/Absyn.stamp.mos
/usr/bin/make -f Makefile --no-print-directory install INCLUDESOURCES=1
clang -g -O2 -fno-stack-protector -fPIC -I"/home/marsj/OpenModelica/build/include/omc/
clang -shared -Wl,-z,origin -Wl,-rpath,'$ORIGIN/../lib/x86_64-linux-gnu/omc' -Wl,-rpat
test ! ".so" = ".dylib" || install_name_tool -id @rpath/libOpenModelicaCompiler.dylib
clang build/_main.o -Wl,-z,origin -Wl,-rpath,'$ORIGIN/../lib/x86_64-linux-gnu/omc' -Wl
cp -a build/OpenModelicaScriptingAPI.h /home/marsj/OpenModelica/build/include/omc/scri
cp -a build/omc /home/marsj/OpenModelica/build/bin/

```

Figure 8.6: Eclipse – building a project, resulting log.

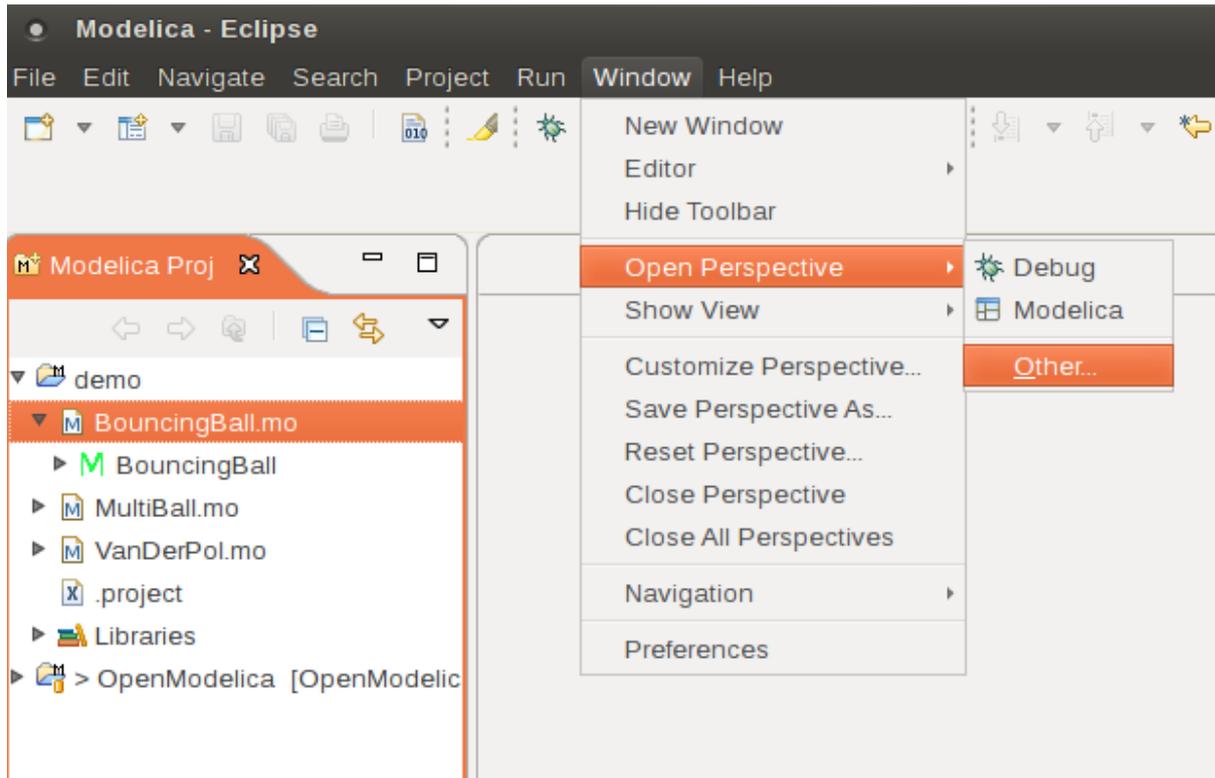


Figure 8.7: Eclipse – Switching to another perspective – e.g. the Java Perspective.

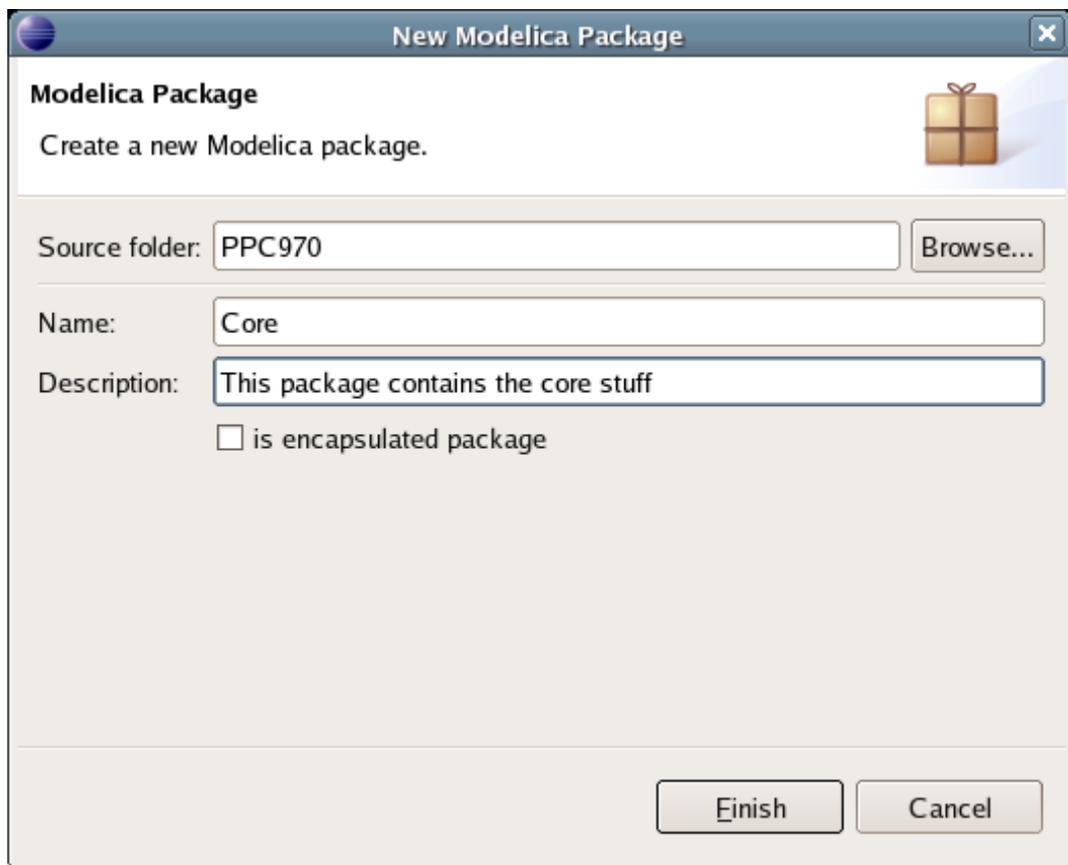


Figure 8.8: Creating a new Modelica package.

8.3.8 Creating a Class

To create a new Modelica class, select where in the hierarchy that you want to add your new class and select File->New->Modelica Class. When creating a Modelica class you can add different restrictions on what the class can contain. These can for example be model, connector, block, record, or function. When you have selected your desired class type, you can select modifiers that add code blocks to the generated code. 'Include initial code block' will for example add the line 'initial equation' to the class.

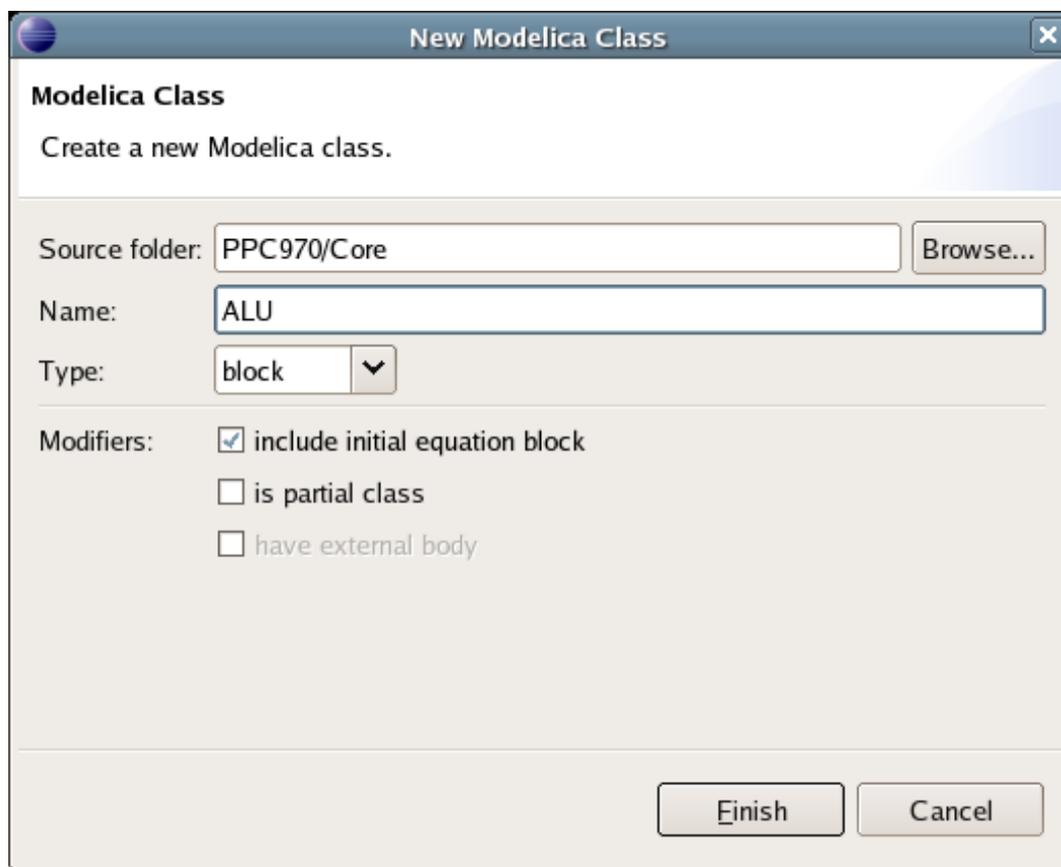


Figure 8.9: Creating a new Modelica class.

8.3.9 Syntax Checking

Whenever a build command is given to the MDT environment, modified and saved Modelica (.mo) files are checked for syntactical errors. Any errors that are found are added to the Problems view and also marked in the source code editor. Errors are marked in the editor as a red circle with a white cross, a squiggly red line under the problematic construct, and as a red marker in the right-hand side of the editor. If you want to reach the problem, you can either click the item in the Problems view or select the red box in the right-hand side of the editor.

8.3.10 Automatic Indentation Support

MDT currently has support for automatic indentation. When typing the Return (Enter) key, the next line is indented correctly. You can also correct indentation of the current line or a range selection using CTRL+I or "Correct Indentation" action on the toolbar or in the Edit menu.

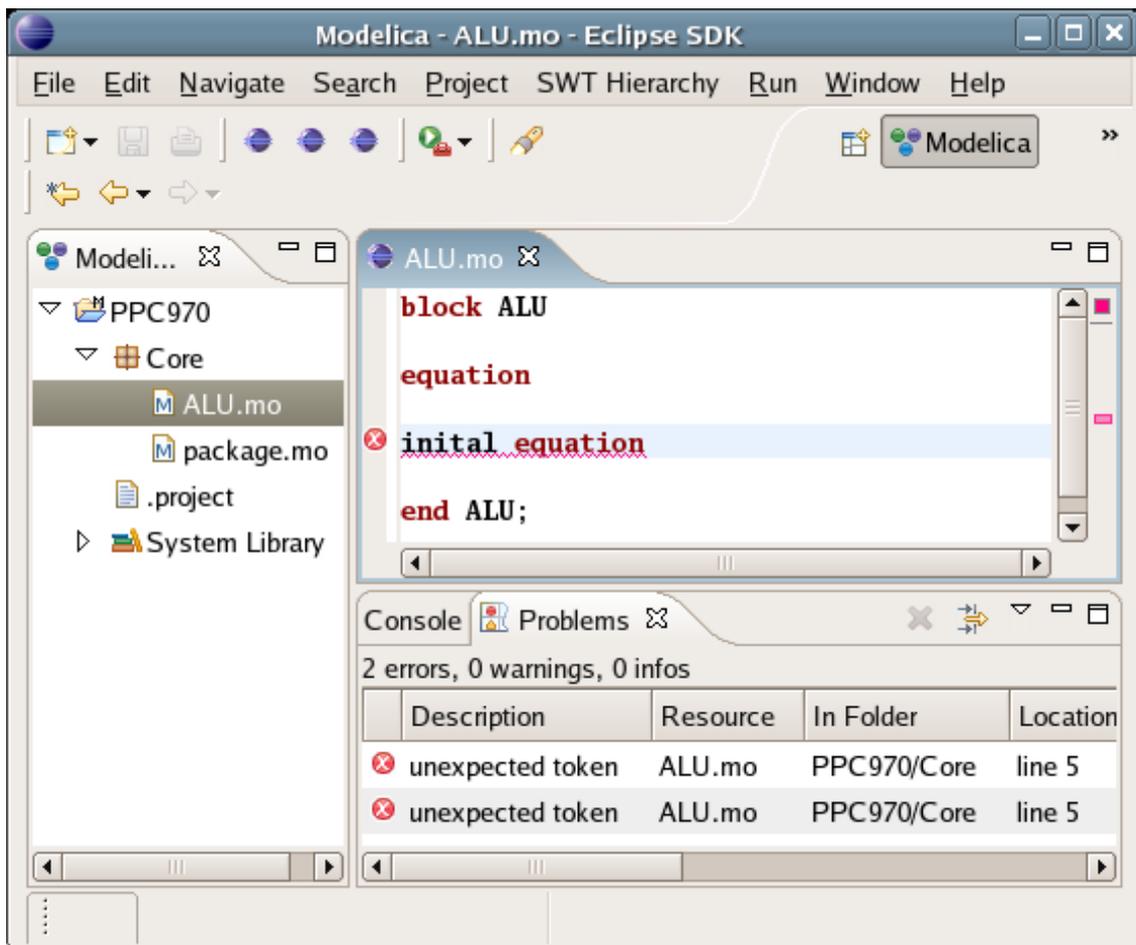


Figure 8.10: Syntax checking.

8.3.11 Code Completion

MDT supports Code Completion in two variants. The first variant, code completion when typing a dot after a class (package) name, shows alternatives in a menu. Besides the alternatives, Modelica documentation from comments is shown if is available. This makes the selection easier.

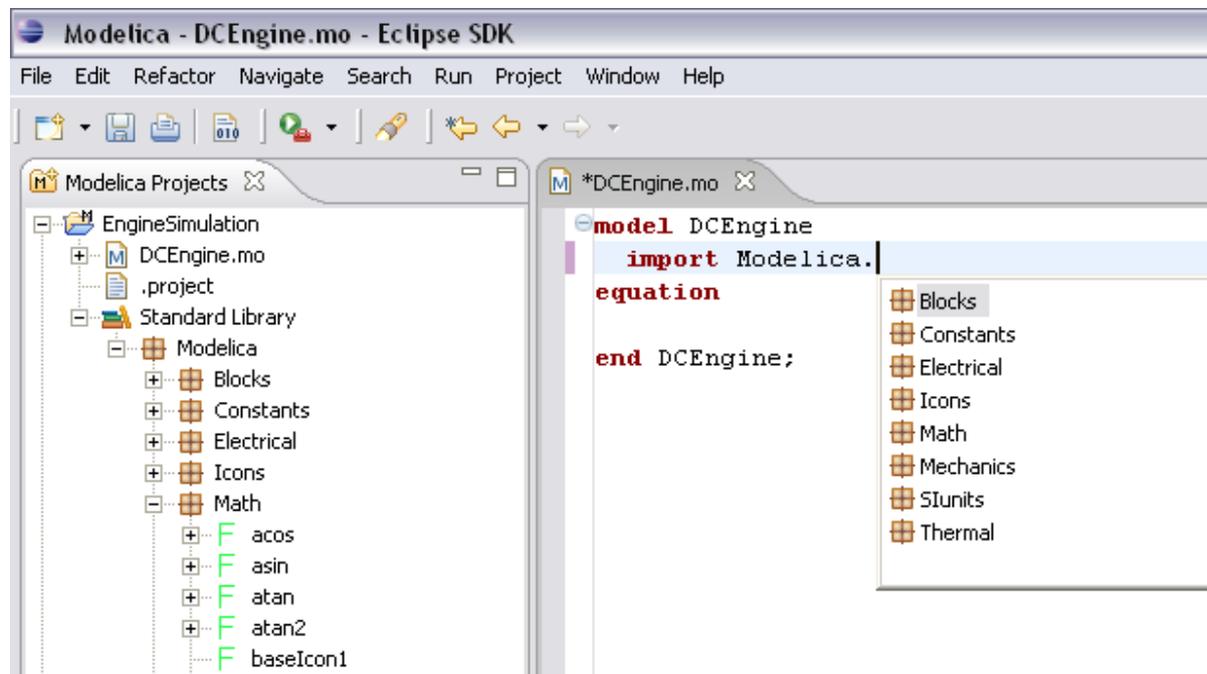


Figure 8.11: Code completion when typing a dot.

The second variant is useful when typing a call to a function. It shows the function signature (formal parameter names and types) in a popup when typing the parenthesis after the function name, here the signature `Real sin(SI.Angle u)` of the `sin` function:

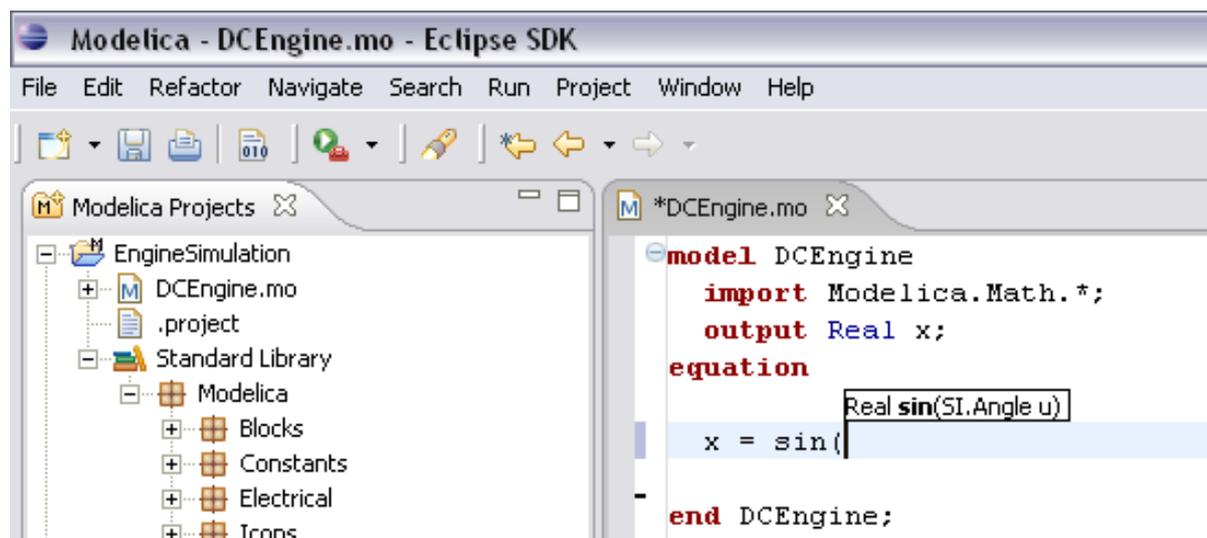


Figure 8.12: Code completion at a function call when typing left parenthesis.

8.3.12 Code Assistance on Identifiers when Hovering

When hovering with the mouse over an identifier a popup with information about the identifier is displayed. If the text is too long, the user can press F2 to focus the popup dialog and scroll up and down to examine all the text. As one can see the information in the popup dialog is syntax-highlighted.

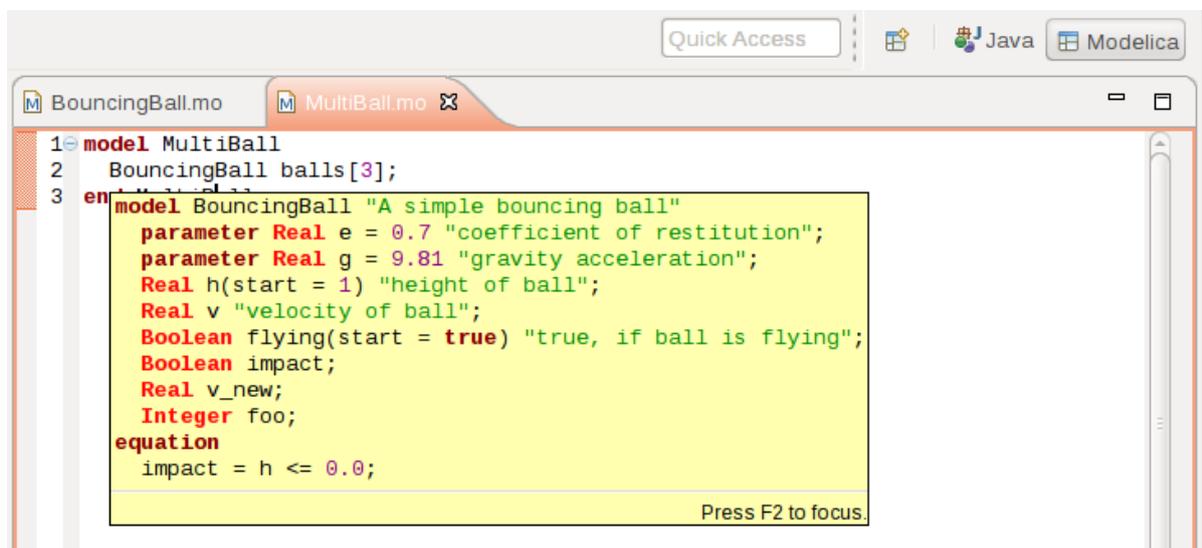


Figure 8.13: Displaying information for identifiers on hovering.

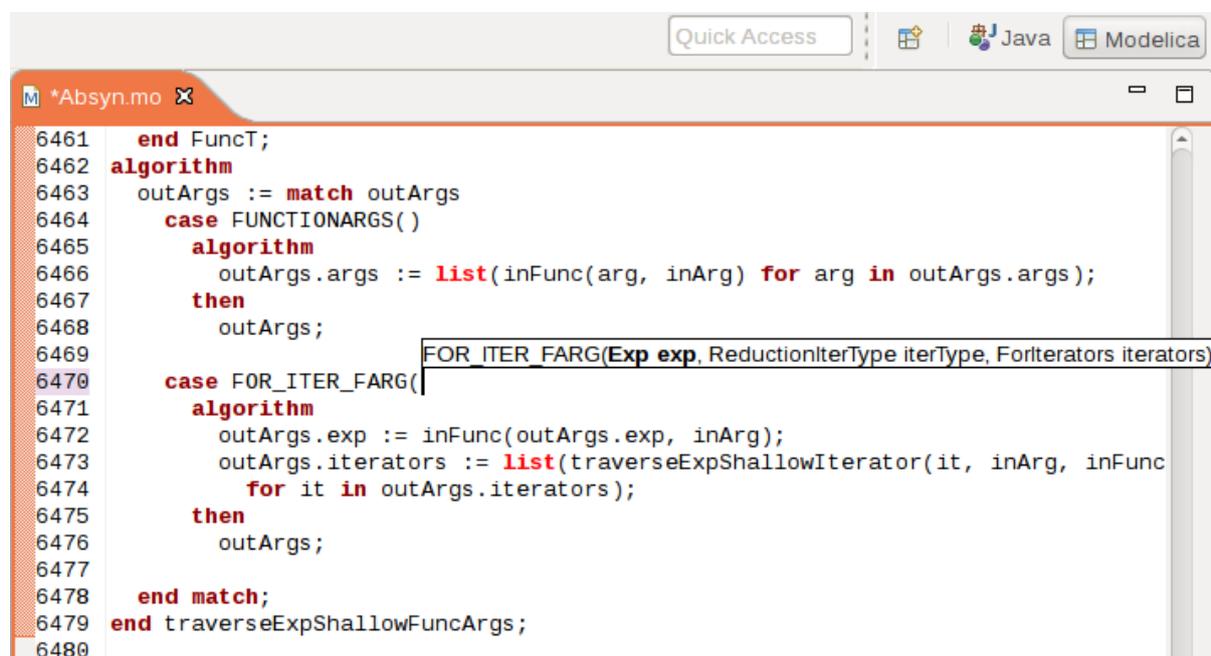
8.3.13 Go to Definition Support

Besides hovering information the user can press CTRL+click to go to the definition of the identifier. When pressing CTRL the identifier will be presented as a link and when pressing mouse click the editor will go to the definition of the identifier.

8.3.14 Code Assistance on Writing Records

When writing records, the same functionality as for function calls is used. This is useful especially in MetaModelica when writing cases in match constructs.

8.3.15 Using the MDT Console for Plotting



```
6461 end FuncT;
6462 algorithm
6463 outArgs := match outArgs
6464 case FUNCTIONARGS()
6465 algorithm
6466 outArgs.args := list(inFunc(arg, inArg) for arg in outArgs.args);
6467 then
6468 outArgs;
6469
6470 case FOR_ITER_FARG(
6471 algorithm
6472 outArgs.exp := inFunc(outArgs.exp, inArg);
6473 outArgs.iterators := list(traverseExpShallowIterator(it, inArg, inFunc
6474 for it in outArgs.iterators);
6475 then
6476 outArgs;
6477
6478 end match;
6479 end traverseExpShallowFuncArgs;
6480
```

Figure 8.14: Code assistance when writing cases with records in MetaModelica.

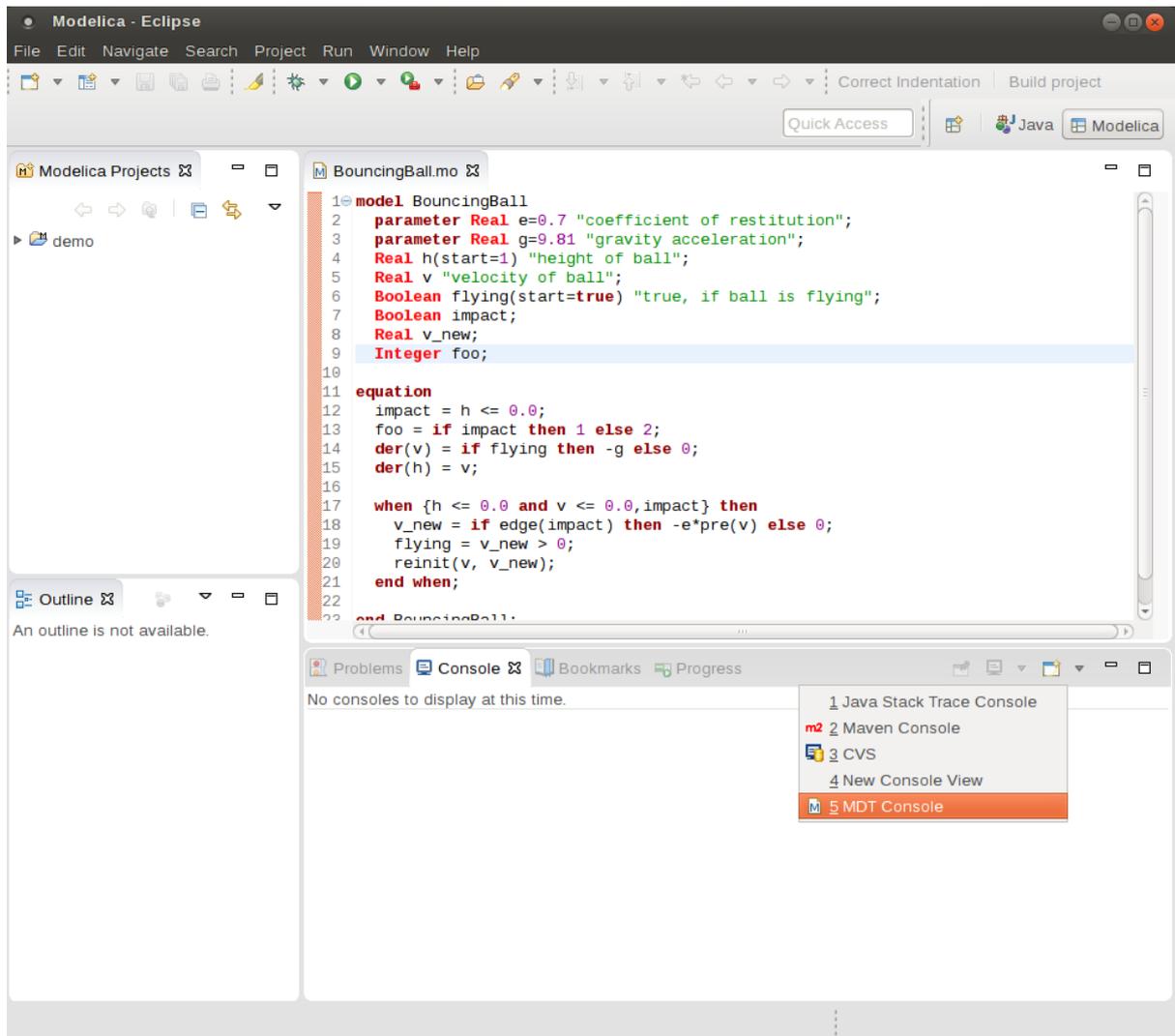


Figure 8.15: Activate the MDT Console.

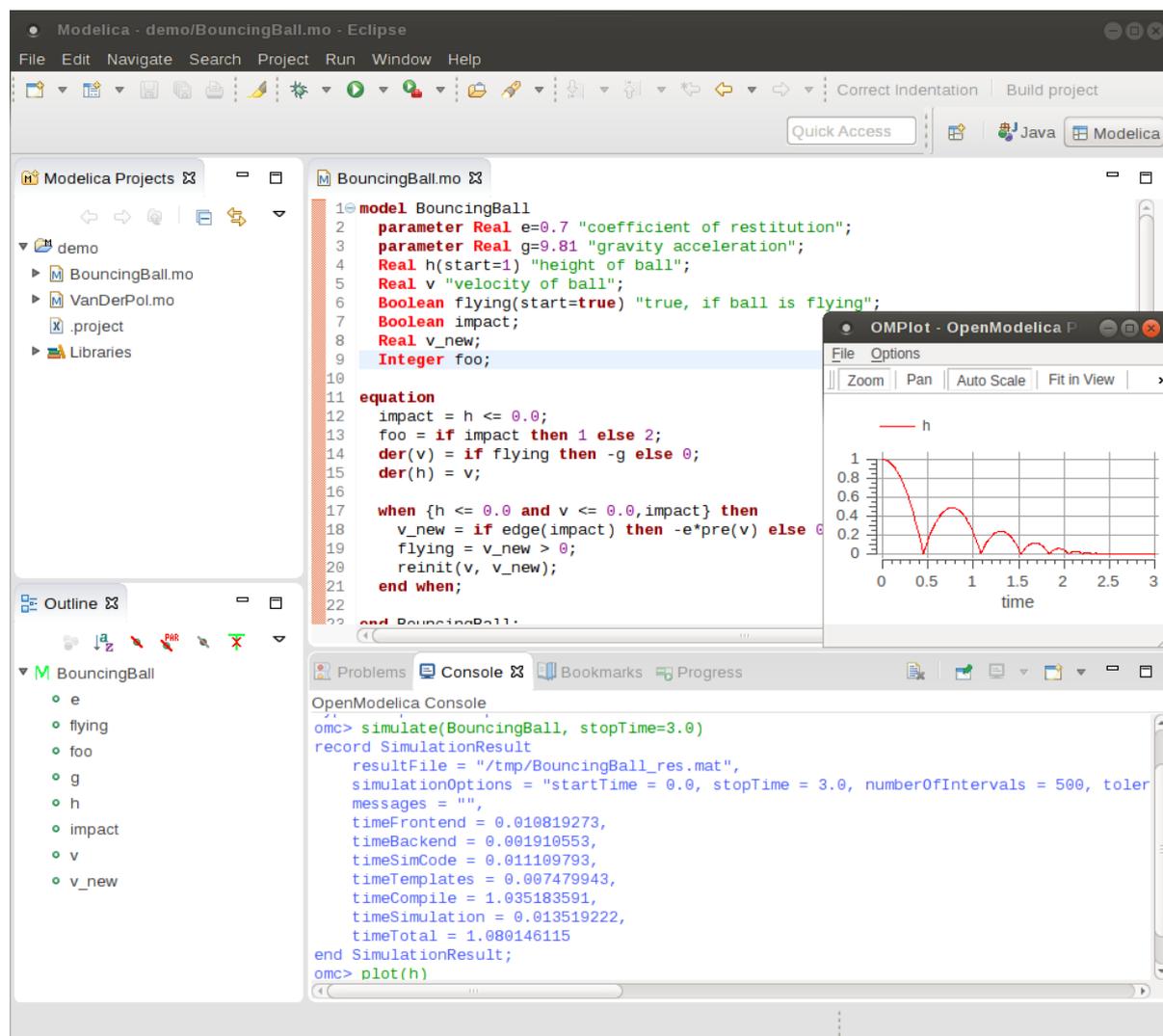


Figure 8.16: Simulation from MDT Console.

MDT DEBUGGER FOR ALGORITHMIC MODELICA

The algorithmic code debugger, used for the algorithmic subset of the Modelica language as well as the Meta-Modelica language is described in Section *The Eclipse-based Debugger for Algorithmic Modelica*. Using this debugger replaces debugging of algorithmic code by primitive means such as print statements or asserts which is complex, time-consuming and error-prone. The usual debugging functionality found in debuggers for procedural or traditional object-oriented languages is supported, such as setting and removing breakpoints, stepping, inspecting variables, etc. The debugger is integrated with Eclipse.

9.1 The Eclipse-based Debugger for Algorithmic Modelica

The debugging framework for the algorithmic subset of Modelica and MetaModelica is based on the Eclipse environment and is implemented as a set of plugins which are available from Modelica Development Tooling (MDT) environment. Some of the debugger functionality is presented below. In the right part a variable value is explored. In the top-left part the stack trace is presented. In the middle-left part the execution point is presented.

The debugger provides the following general functionalities:

- Adding/Removing breakpoints.
- Step Over – moves to the next line, skipping the function calls.
- Step In – takes the user into the function call.
- **Step Return – complete the execution of the function and takes the** user back to the point from where the function is called.
- Suspend – interrupts the running program.

9.1.1 Starting the Modelica Debugging Perspective

To be able to run in debug mode, one has to go through the following steps:

- create a mos file
- setting the debug configuration
- setting breakpoints
- running the debug configuration

All these steps are presented below using images.

Create mos file

In order to debug Modelica code we need to load the Modelica files into the OpenModelica Compiler. For this we can write a small script file like this:

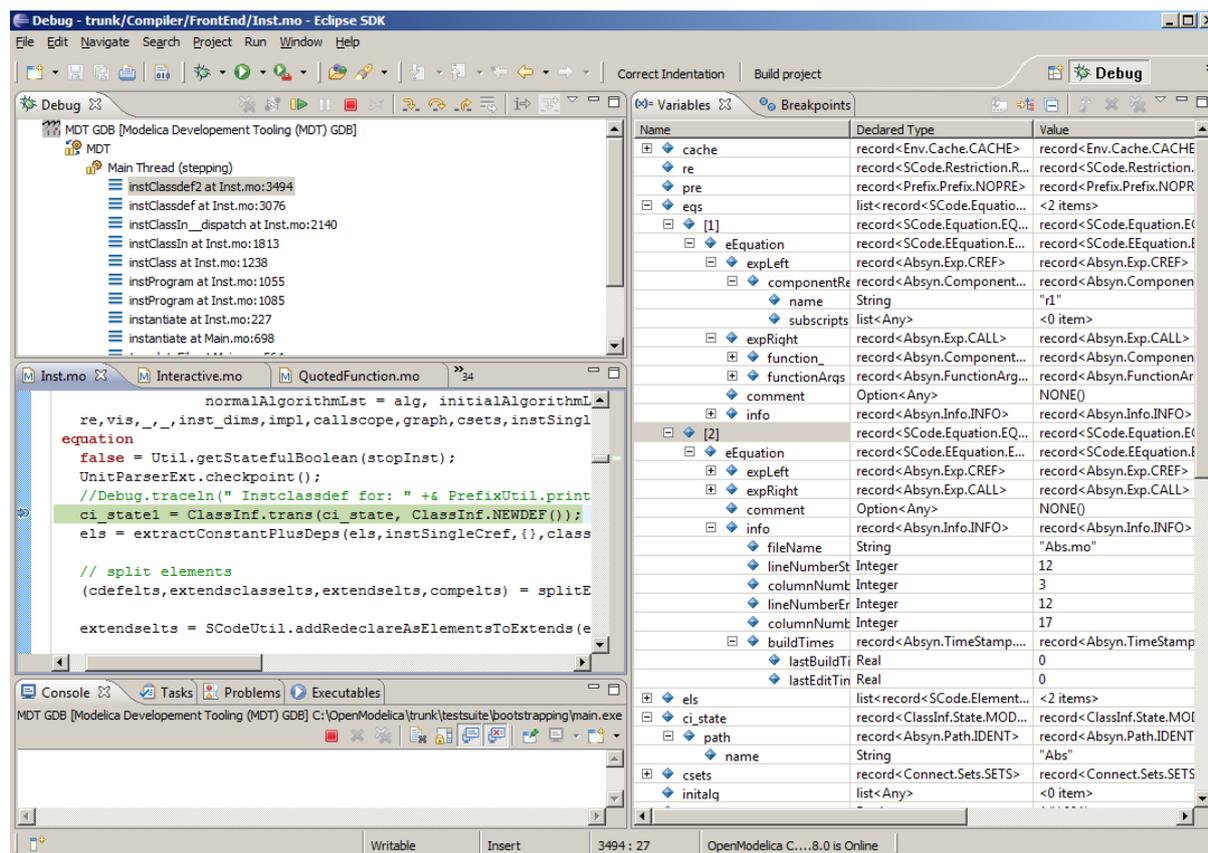


Figure 9.1: Debugging functionality.

```
function HelloWorld
  input Real r;
  output Real o;
algorithm
  o := 2 * r;
end HelloWorld;
```

```
>>> setCommandLineOptions({"+d=rml,noevalfunc", "+g=MetaModelica"})
{true,true}
>>> setCFlags(getCFlags() + " -g")
true
>>> HelloWorld(120.0)
240.0
```

So lets say that we want to debug HelloWorld.mo. For that we must load it into the compiler using the script file. Put all the Modelica files there in the script file to be loaded. We should also initiate the debugger by calling the starting function, in the above code HelloWorld(120.0);

Setting the debug configuration

While the Modelica perspective is activated the user should click on the bug icon on the toolbar and select Debug in order to access the dialog for building debug configurations.

To create the debug configuration, right click on the classification Modelica Development Tooling (MDT) GDB and select New as in figure below. Then give a name to the configuration, select the debugging executable to be executed and give it command line parameters. There are several tabs in which the user can select additional debug configuration settings like the environment in which the executable should be run.

Note that we require Gnu Debugger (GDB) for debugging session. We must specify the GDB location, also we

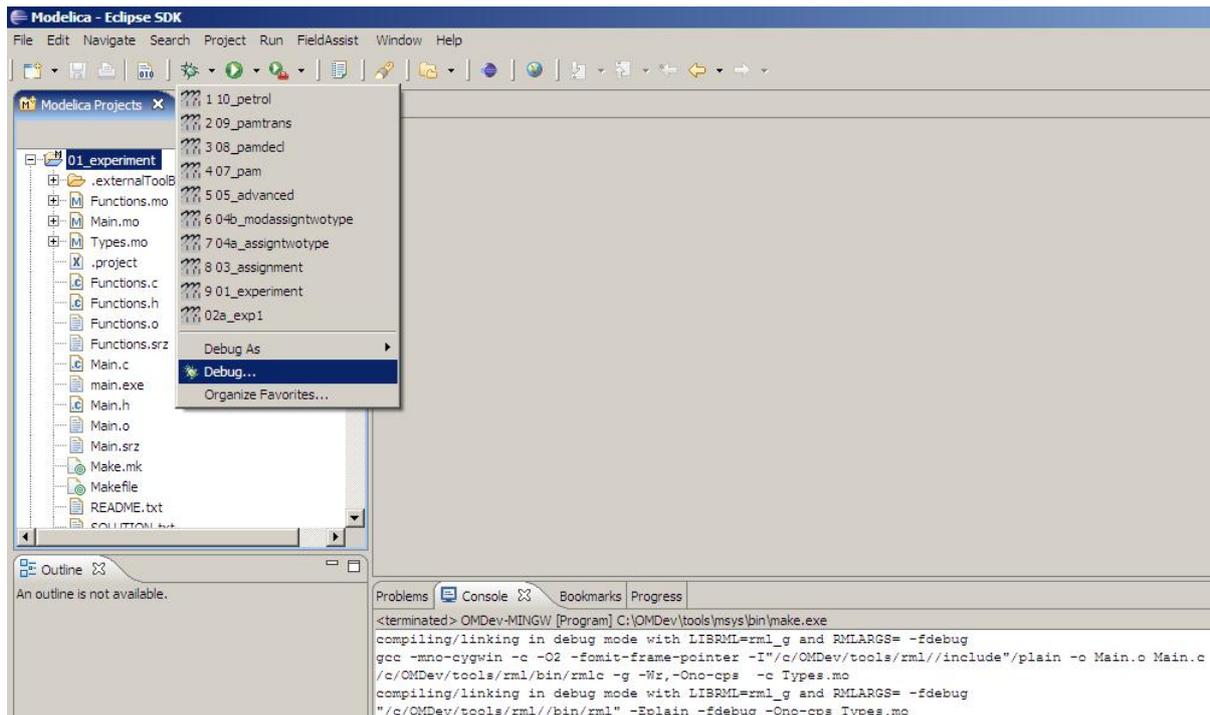


Figure 9.2: Accessing the debug configuration dialog.

must pass our script file as an argument to OMC.

Setting/Deleting Breakpoints

The Eclipse interface allows to add/remove breakpoints. At the moment only line number based breakpoints are supported. Other alternative to set the breakpoints is; function breakpoints.

Starting the debugging session and enabling the debug perspective

9.1.2 The Debugging Perspective

The debug view primarily consists of two main views:

- Stack Frames View
- Variables View

The stack frame view, shown in the figure below, shows a list of frames that indicates how the flow had moved from one function to another or from one file to another. This allows backtracing of the code. It is very much possible to select the previous frame in the stack and inspect the values of the variables in that frame. However, it is not possible to select any of the previous frame and start debugging from there. Each frame is shown as `<function_name at file_name:line_number>`.

The Variables view shows the list of variables at a certain point in the program, containing four columns:

- Name – the variable name.
- Declared Type – the Modelica type of the variable.
- Value – the variable value.
- Actual Type – the mapped C type.

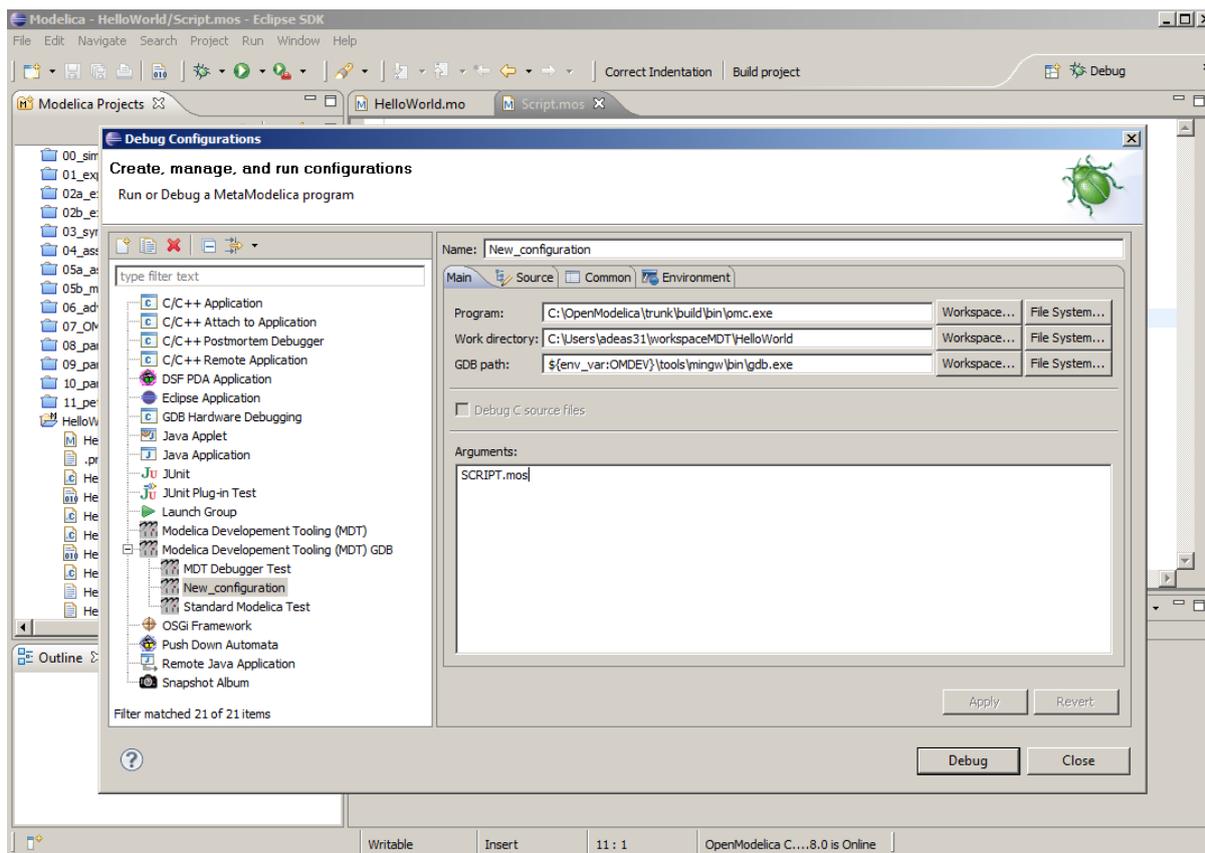


Figure 9.3: Creating the Debug Configuration.

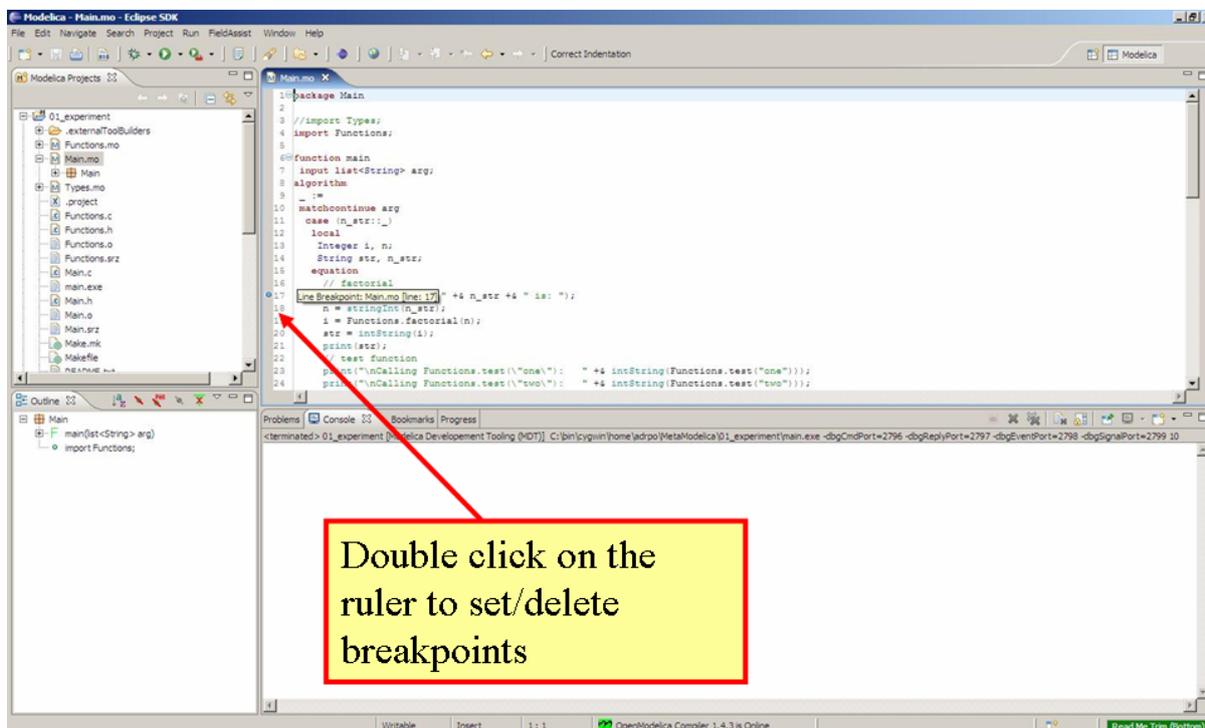


Figure 9.4: Setting/deleting breakpoints.

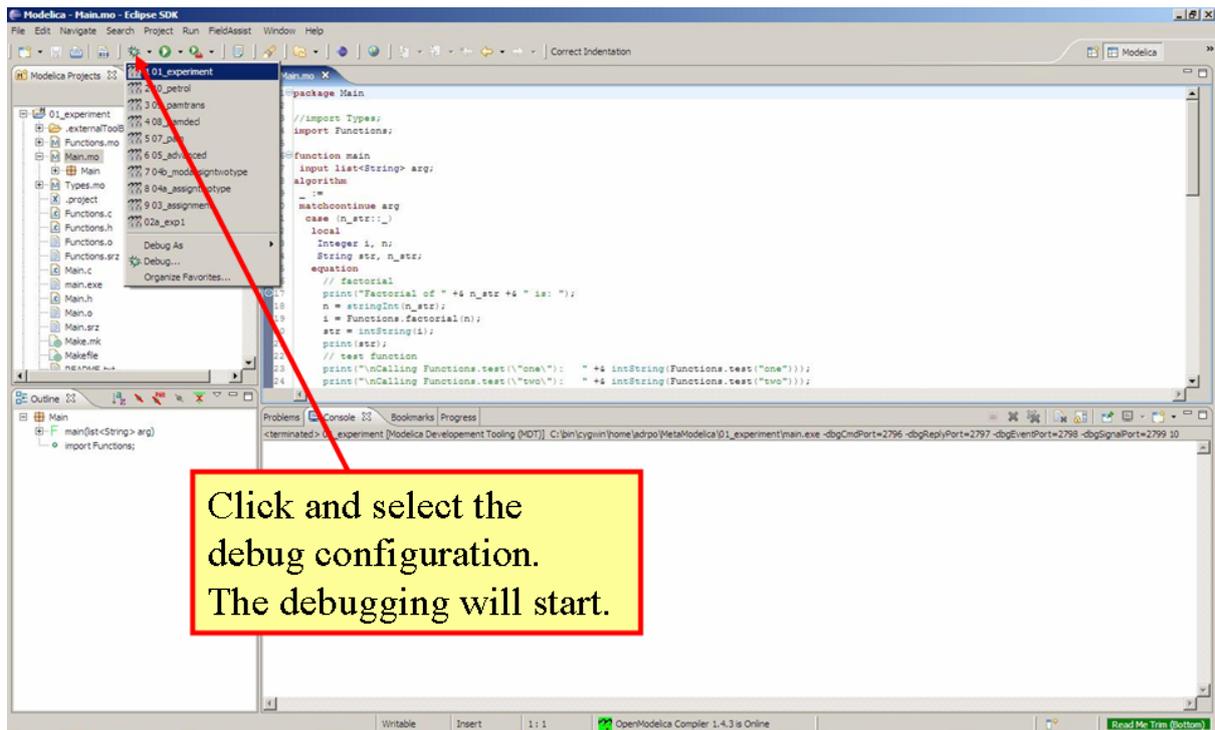


Figure 9.5: Starting the debugging session.

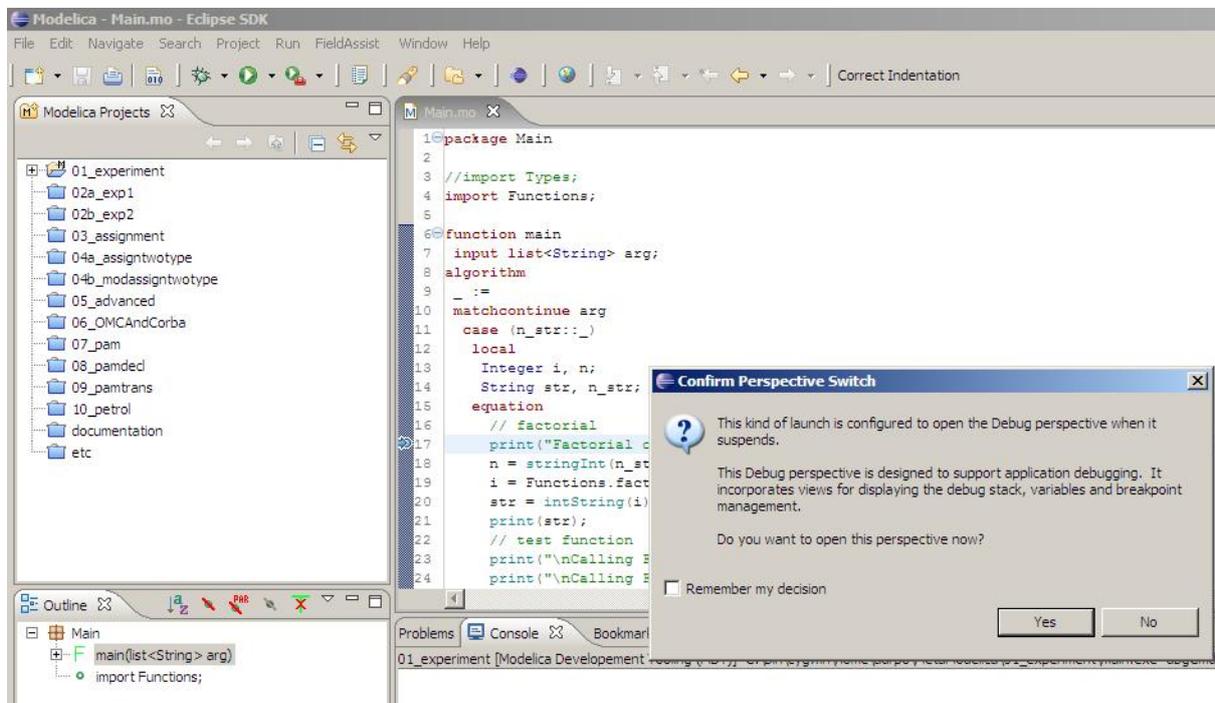


Figure 9.6: Eclipse will ask if the user wants to switch to the debugging perspective.

By preserving the stack frames and variables it is possible to keep track of the variables values. If the value of any variable is changed while stepping then that variable will be highlighted yellow (the standard Eclipse way of showing the change).

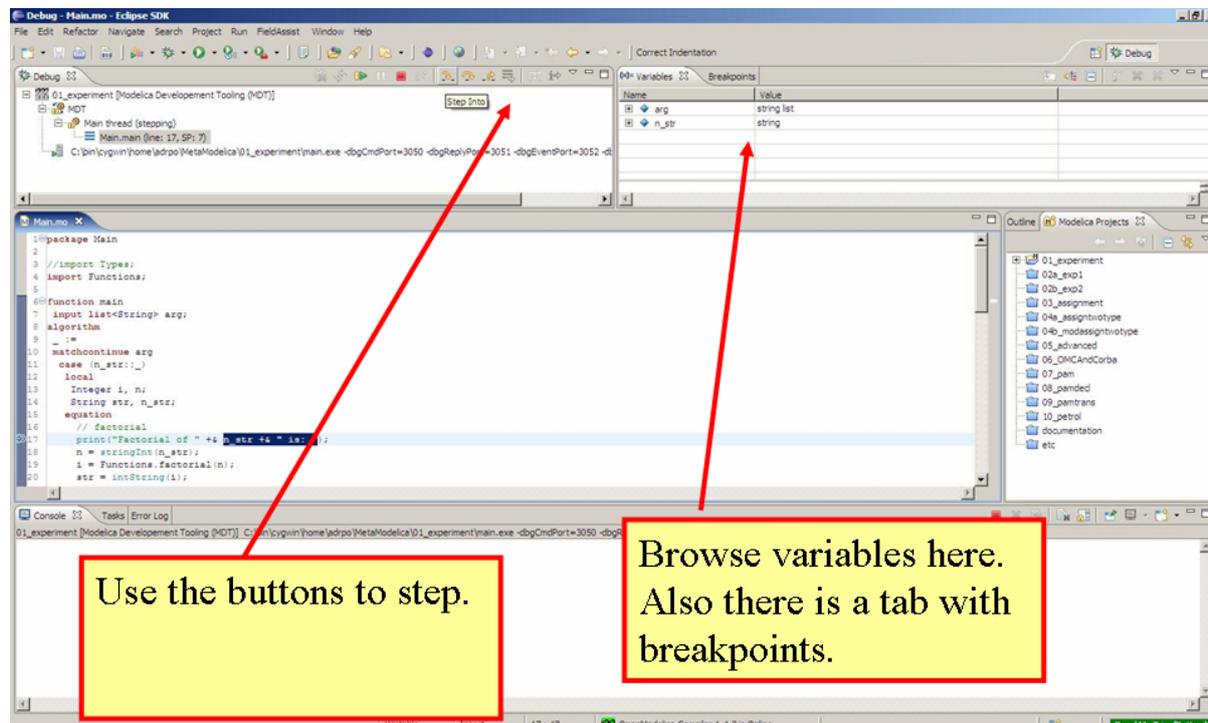


Figure 9.7: The debugging perspective.

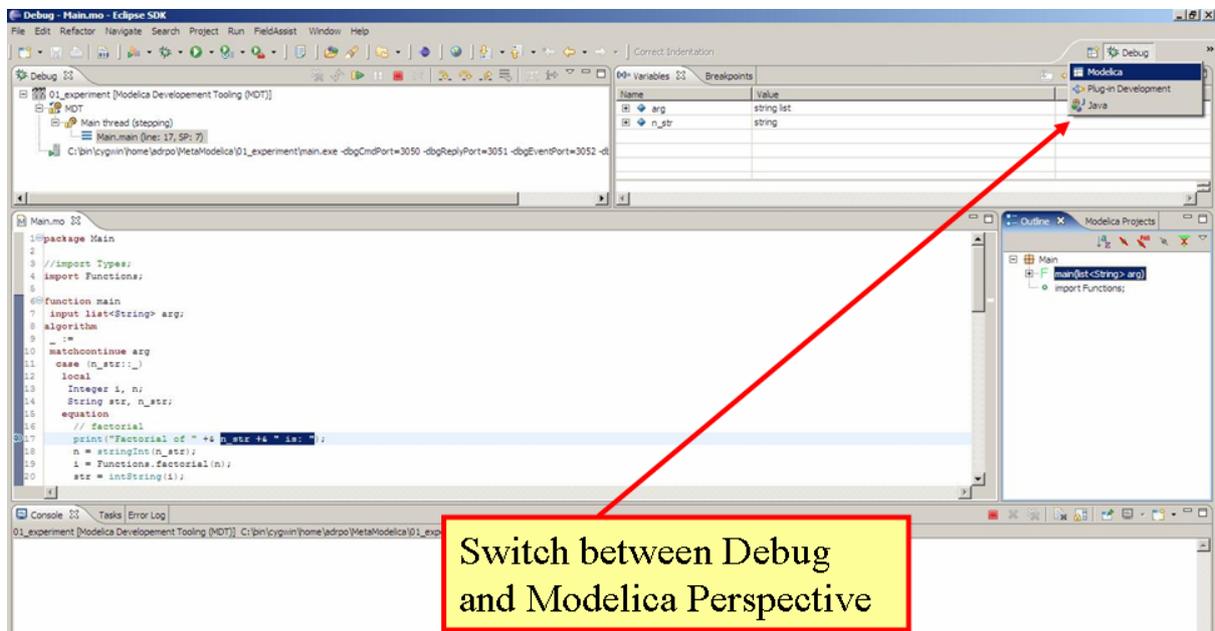


Figure 9.8: Switching between perspectives.

MODELICA PERFORMANCE ANALYZER

A common problem when simulating models in an equation-based language like Modelica is that the model may contain non-linear equation systems. These are solved in each time-step by extrapolating an initial guess and running a non-linear system solver. If the simulation takes too long to simulate, it is useful to run the performance analysis tool. The tool has around 5~25% overhead, which is very low compared to instruction-level profilers (30x-100x overhead). Due to being based on a single simulation run, the report may contain spikes in the charts.

When running a simulation for performance analysis, execution times of user-defined functions as well as linear, non-linear and mixed equation systems are recorded.

To start a simulation in this mode, just use the `measureTime` flag of the `simulate` command.

```
>>> simulate(modelname, measureTime = true)
```

The generated report is in HTML format (with images in the SVG format), stored in a file `modelname_prof.html`, but the XML database and measured times that generated the report and graphs are also available if you want to customize the report for comparison with other tools.

Below we use the performance profiler on the simple model A:

```
model ProfilingTest
  function f
    input Real r;
    output Real o = sin(r);
  end f;
  String s = "abc";
  Real x = f(x) "This is x";
  Real y(start=1);
  Real z1 = cos(z2);
  Real z2 = sin(z1);
equation
  der(y) = time;
end ProfilingTest;
```

We simulate as usual, but set `measureTime=true` to activate the profiling:

```
>>> setCommandLineOptions("--profiling=blocks+html")
true
>>> simulate(ProfilingTest)
record SimulationResult
  resultFile = "«DOCHOME»/ProfilingTest_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500, tolerance = 1e-10",
  messages = "Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
stdout      | info      | Time measurements are stored in ProfilingTest_prof.html (human-readable)
",
  timeFrontend = 0.007625929000000001,
```

```

timeBackend = 0.006777695,
timeSimCode = 0.05190009,
timeTemplates = 0.005140966,
timeCompile = 0.207390045,
timeSimulation = 0.039139536,
timeTotal = 0.318047681
end SimulationResult;
"Warning: The initial conditions are not fully specified. Use +d=initialization for more informat
Warning: There are iteration variables with default zero start attribute. Use +d=initialization f
"

```

10.1 Profiling information for ProfilingTest

10.1.1 Information

All times are measured using a real-time wall clock. This means context switching produces bad worst-case execution times (max times) for blocks. If you want better results, use a CPU-time clock or run the command using real-time privileges (avoiding context switches).

Note that for blocks where the individual execution time is close to the accuracy of the real-time clock, the maximum measured time may deviate a lot from the average.

For more details, see ProfilingTest_prof.xml.

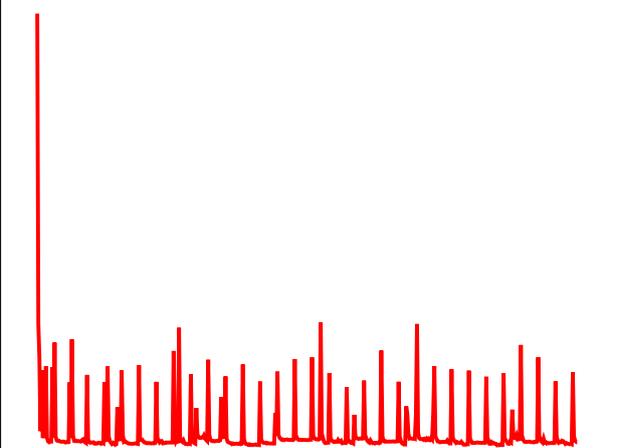
10.1.2 Settings

Name	Value
Integration method	dassl
Output format	mat
Output name	ProfilingTest_res.mat
Output size	24.0 kB
Profiling data	ProfilingTest_prof.data
Profiling size	0 B

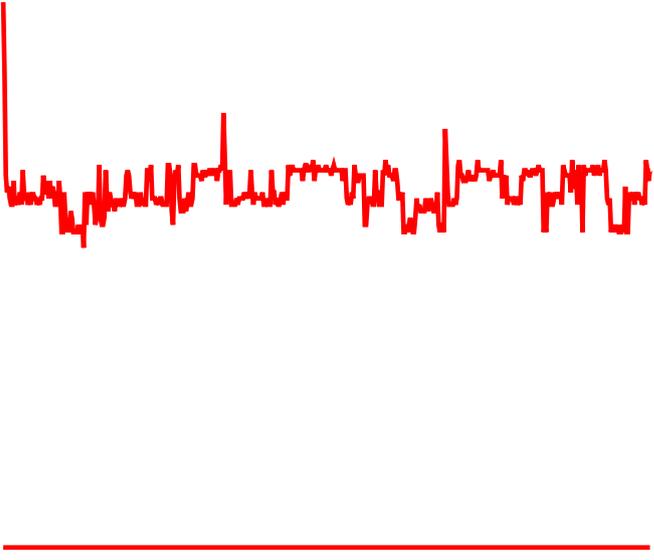
10.1.3 Summary

Task	Time	Fraction
Pre-Initialization	0.000208	7.89%
Initialization	0.000070	2.66%
Event-handling	0.000028	1.06%
Creating output file	0.000349	13.24%
Linearization	0.000000	0.00%
Time steps	0.001693	64.23%
Overhead	0.000267	10.13%
Unknown	0.000229	0.80%
Total simulation time	0.002636	100.00%

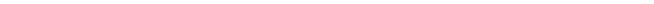
10.1.4 Global Steps

	Steps	Total Time	Fraction	Average Time	Max Time	Deviation
	499	0.00169364	2.3%	3.39278557114218e-06	1.028e-06	1.982e-06

10.1.5 Measured Function Calls

	Name	Calls	Time	Fraction	Max Time	Deviation
	<i>Pro-filingTest.f</i>	506	0.000016077	0.000004252	0.000000000	0.000000000x

10.1.6 Measured Blocks

	Name	Calls	Time	Fraction	Max Time	Deviation
	'	12	0.000034989	0.000034989%	0.0000318028x	
	'	2	0.000000818	0.000000818%	0.000001898x	

Equations

Name	Variables
	' <#var0>' __
	' <#var0>' __
	' <#var0>' __
	' <#var0>' __
	' <#var0>' __
	' <#var0>' __
	' <#var0>' __

Variables

Name	Comment
'y <<interactive>#line=8>' __	
'der(y) <<interactive>#line=8>' __	
'x <<interactive>#line=7>' __	This is x
'z1 <<interactive>#line=9>' __	
'z2 <<interactive>#line=10>' __	
's <<interactive>#line=6>' __	

This report was generated by OpenModelica on 2015-11-24 23:00:15.

10.2 Generated JSON for the Example

10.3 Using the Profiler from OMEdit

When running a simulation from OMEdit, it is possible to enable profiling information, which can be combined with the *transformations browser*.

When profiling the DoublePendulum example from MSL, the following output in [Figure 10.2](#) is a typical result. This information clearly shows which system takes longest to simulate (a linear system, where most of the time overhead probably comes from initializing LAPACK over and over).

Listing 10.1: ProfilingTest_prof.json

```

{
  "name": "ProfilingTest",
  "prefix": "ProfilingTest",
  "date": "2015-11-24 23:00:15",
  "method": "dassl",
  "outputFormat": "mat",
  "outputFilename": "ProfilingTest_res.mat",
  "outputFilesize": 24569,
  "overheadTime": 0.000267236,
  "preinitTime": 0.000208299,
  "initTime": 7.0218e-05,
  "eventTime": 2.8113e-05,
  "outputTime": 0.000348534,
  "linearizeTime": 0,
  "totalTime": 0.00263597,
  "totalStepsTime": 3.023e-06,
  "totalTimeProfileBlocks": 0.00098221,
  "numStep": 499,
  "maxTime": 7.065e-05,
  "functions": [
    { "name": "ProfilingTest.f", "ncall": 506, "time": 0.000015014, "maxTime": 0.000000425 }
  ],
  "profileBlocks": [
    { "id": 0, "ncall": 12, "time": 0.000034989, "maxTime": 0.000035803 },
    { "id": 8, "ncall": 2, "time": 0.000001818, "maxTime": 0.000001892 },
    { "id": 12, "ncall": 504, "time": 0.000370426, "maxTime": 0.000026113 },
    { "id": 14, "ncall": 504, "time": 0.000574977, "maxTime": 0.000022911 }
  ]
}

```

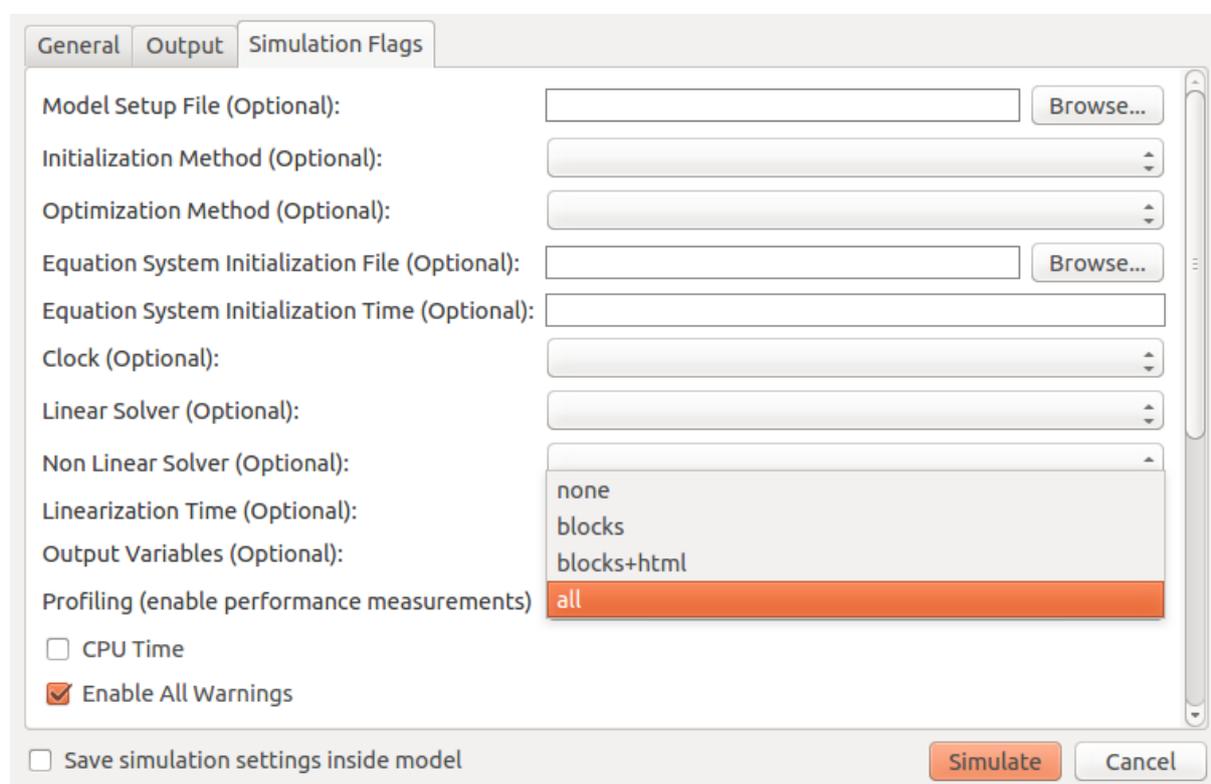


Figure 10.1: Setting up the profiler from OMEdit.

Equations Browser						Defines	
Index	Type	Equation	Executions	Max time	Time	Fraction	Variable
+ 876	regular	linear, size 2	4602	0.000199	0.0582	86.2%	
- 836	regular	(assignment) revolute2.R_rel.T[2,2] = cos(revolute2.phi)	1534	8.25e-05	0.000491	0.728%	
- 837	regular	(assignment) revolute2.R_rel.T[2,1] = -sin(revolute2.phi)	1534	7.29e-05	0.000422	0.625%	
- 841	regular	(assignment) boxBody1.frame_...[2,1] = -sin(damper.phi_rel)	1534	7.1e-05	0.000395	0.585%	
- 840	regular	(assignment) boxBody1.frame_...T[2,2] = cos(damper.phi_rel)	1534	7.08e-05	0.000361	0.535%	
- 839	regular	(assignment) revolute2.R_rel.T[1,1] = cos(revolute2.phi)	1534	7.33e-05	0.000303	0.449%	
- 842	regular	(assignment) boxBody1.frame_b.R.T[1,2] = sin(damper.phi_rel)	1534	7.45e-05	0.000303	0.449%	
- 838	regular	(assignment) revolute2.R_rel.T[1,2] = sin(revolute2.phi)	1534	7.11e-05	0.0003	0.444%	
- 849	regular	(assignment) boxBody1.frame_...T[1,1] = cos(damper.phi_rel)	1534	7.29e-05	0.000286	0.424%	
- 827	regular	(assignment) revolute1.tau = (-damper.d) * revolute1.w	1534	6.84e-05	0.000274	0.406%	

Figure 10.2: Profiling results of the Modelica standard library DoublePendulum example, sorted by execution time.

MODELICA3D

Modelica3D is a lightweight, platform independent 3D-visualisation library for Modelica. Read more about the Modelica3D library [here](#).

11.1 Installing Modelica3D

11.1.1 Windows

In order to run Modelica3D on windows you need following softwares;

- **Python** – Install python from <http://www.python.org/download/>. Python2.7.3 is recommended.
- **PyGTK** – Install GTK+ for python from <http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/2.24/>. Download the all-in-one package. Recommended is pygtk-all-in-one-2.24.2.win32-py2.7.msi.

11.1.2 MacOS

On MacOS you can use the 3d visualization like this. Note that on your system the paths used here might vary. In one terminal type:

```
# start the dbus server (you only need to do this once)
sudo launchctl load -w /opt/openmodelica/Library/LaunchDaemons/org.freedesktop.dbus-system.plist
launchctl load -w /opt/openmodelica/Library/LaunchAgents/org.freedesktop.dbus-session.plist
# export python path
export PYTHONPATH=/opt/openmodelica/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
```

11.2 Running Modelica3D

Run the Modelica3D server by executing the dbus-server.py script located in your OpenModelica or Modelica3D installation, for example:

```
>>> "python " + getInstallationDirectoryPath() + "/lib/omlibrary-modelica3d/osg-gtk/dbus-server.py"
"python «OPENMODELICAHOME»/lib/omlibrary-modelica3d/osg-gtk/dbus-server.py"
```

Running this command in a command prompt will start the Modelica3D server and on success you should see the output:

Running dbus-server...

Now run the simulation. The following commands will load the Modelica3D library and the modified DoublePendulum example:

```
>>> loadModelica3D()
true
```

```
model DoublePendulum
  extends Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum;
  inner ModelicaServices.Modelica3D.Controller m3d_control;
end DoublePendulum;
```

Then simulate the DoublePendulum:

```
>>> simulate(DoublePendulum)
```

If everything goes fine a *3D visualization of DoublePendulum* will pop-up.

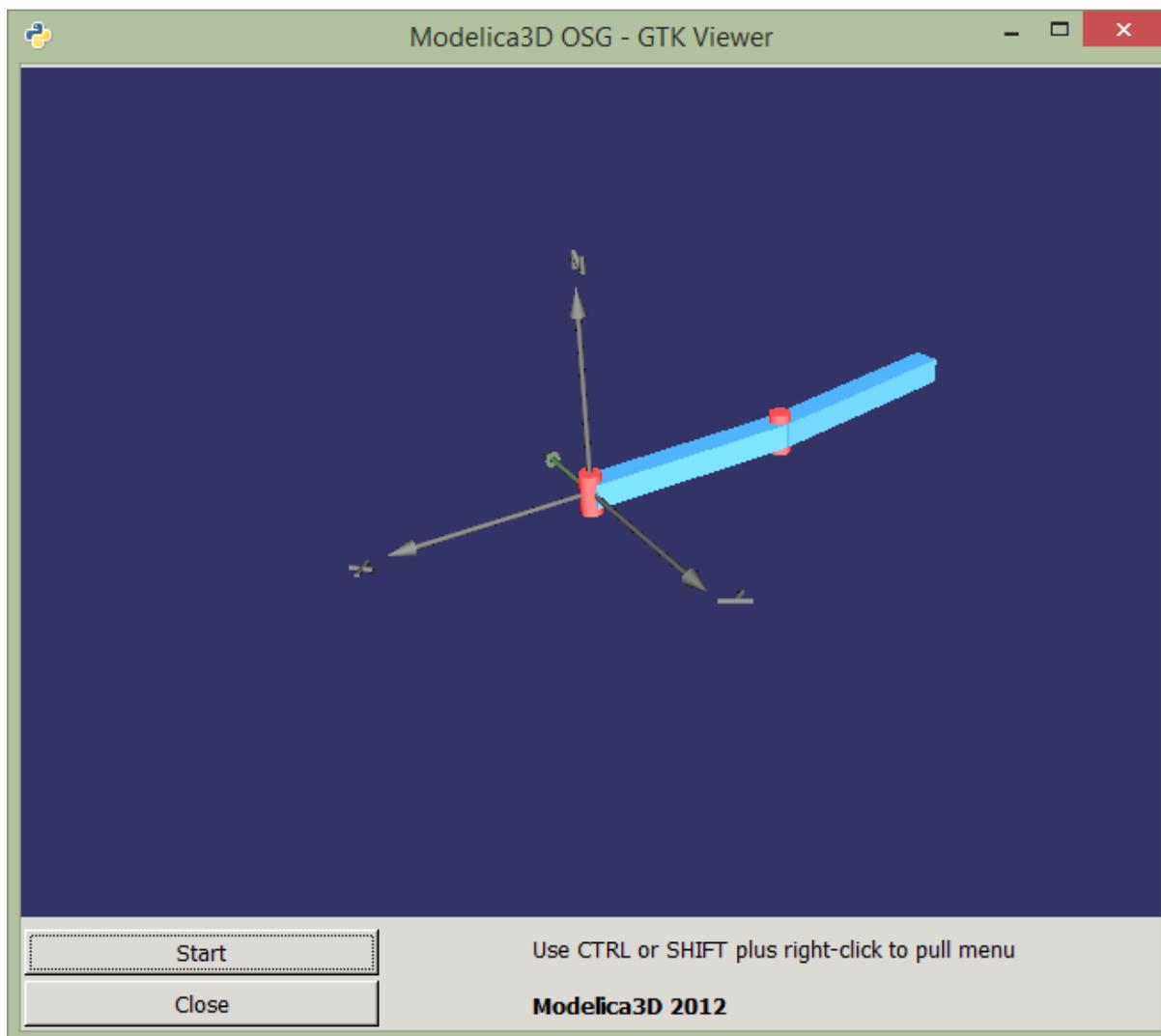


Figure 11.1: 3D visualization of DoublePendulum

To visualize any models from the MultiBody library you can use this script and change the extends to point to the model you want. Note that you will need to add visualisers to your model similarly to what Modelica.MultiBody library has. The documentation of the visualizers is available [here](#).

```
model Visualize_MyModel
  inner ModelicaServices.Modelica3D.Controller m3d_control;
  extends MyModel;
end Visualize_MyModel;
```

```
>>> simulate(Visualize_MyModel)
```

SIMULATION IN WEB BROWSER

OpenModelica can simulate in a web browser on a client computer by model code being compiled to efficient Javascript code.

For more information, see <https://github.com/tshort/openmodelica-javascript>

Below used on the MSL MultiBody RobotR3.fullRobot example model.

The screenshot shows a web browser window with the URL `http://tshort.github.io/mdpad/mdpad.html?Modelica...`. The page title is "OpenModelica simulation example" and the model path is `Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot`. The simulation status is "Simulation finished. Time: 00:40".

On the left, there are three input fields for simulation parameters:

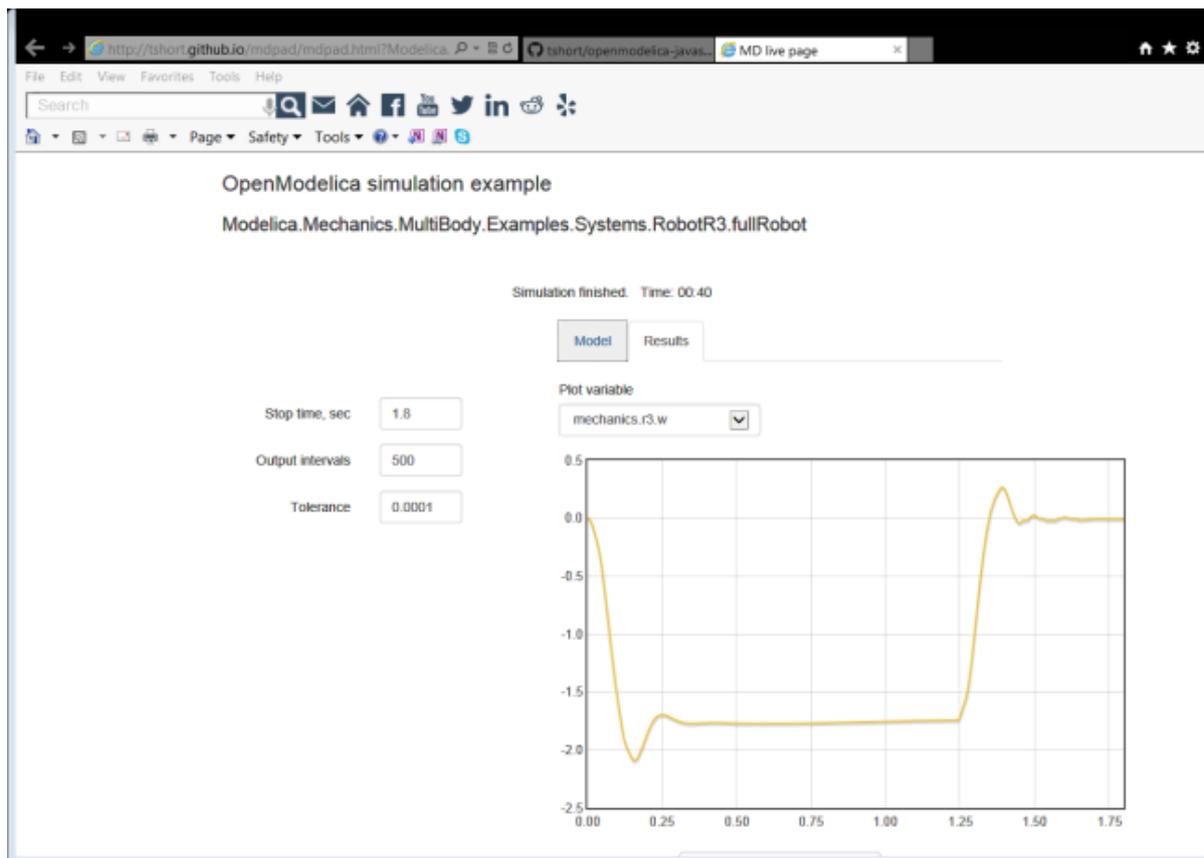
- Stop time, sec: 1.8
- Output intervals: 500
- Tolerance: 0.0001

The main simulation area is divided into two tabs: "Model" and "Results". The "Model" tab is active, showing a block diagram of the robot arm model. The diagram includes a "control" block on the left, several "motor" blocks in the middle, and a "robot" block on the right. The "robot" block contains a 3D model of a robot arm. The "Results" tab is currently empty.

Below the simulation area, there is a "Comments" section with the following text:

This simulation model is from a Modelica model. Modelica is a language for simulating electrical, thermal, and mechanical systems. OpenModelica was used to compile this model to C. Then, Emscripten was used to compile the C code to JavaScript. For more information on compiling OpenModelica to JavaScript, see [here](#).

The user interface was created in mdpad. See Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot.md for the Markdown code for this page.



INTEROPERABILITY – C AND PYTHON

Below is information and examples about the OpenModelica external C interfaces, as well as examples of Python interoperability.

13.1 Calling External C functions

The following is a small example (ExternalLibraries.mo) to show the use of external C functions:

```
model ExternalLibraries

  function ExternalFunc1
    input Real x;
    output Real y;
    external y=ExternalFunc1_ext(x) annotation(Library="ExternalFunc1.o", LibraryDirectory="modelica:");
  end ExternalFunc1;

  function ExternalFunc2
    input Real x;
    output Real y;
    external "C" annotation(Library="ExternalFunc2", LibraryDirectory="modelica://ExternalLibraries");
  end ExternalFunc2;

  Real x(start=1.0, fixed=true), y(start=2.0, fixed=true);
equation
  der(x)=-ExternalFunc1(x);
  der(y)=-ExternalFunc2(y);
end ExternalLibraries;
```

These C (.c) files and header files (.h) are needed (note that the headers are not needed since OpenModelica will generate the correct definition if it is not present; using the headers it is possible to write C-code directly in the Modelica source code or declare non-standard calling conventions):

Listing 13.1: ExternalFunc1.c

```
double ExternalFunc1_ext(double x)
{
  double res;
  res = x+2.0*x*x;
  return res;
}
```

Listing 13.2: ExternalFunc1.h

```
double ExternalFunc1_ext(double);
```

The following script file ExternalLibraries.mos will perform everything that is needed, provided you have gcc installed in your path:

Listing 13.3: ExternalFunc2.c

```
double ExternalFunc2(double x)
{
  double res;
  res = (x-1.0)*(x+2.0);
  return res;
}
```

```
>>> system(getCompiler() + " -c -o ExternalFunc1.o ExternalFunc1.c")
0
>>> system(getCompiler() + " -c -o ExternalFunc2.o ExternalFunc2.c")
0
>>> system("ar rcs libExternalFunc2.a ExternalFunc2.o")
0
>>> simulate(ExternalLibraries)
record SimulationResult
  resultFile = "«DOCHOME»/ExternalLibraries_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "",
  timeFrontend = 0.0065556990000000001,
  timeBackend = 0.002448235,
  timeSimCode = 0.045271704,
  timeTemplates = 0.003806312,
  timeCompile = 0.255194138,
  timeSimulation = 0.007489586,
  timeTotal = 0.320842554
end SimulationResult;
```

And plot the results:

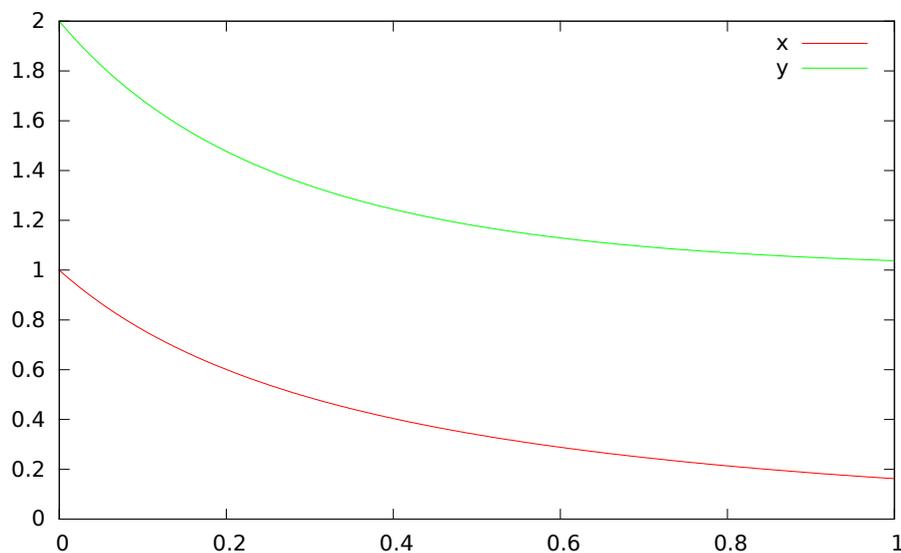


Figure 13.1: Plot generated by OpenModelica+gnuplot

13.2 Calling external Python Code from a Modelica model

The following calls external Python code through a very simplistic external function (no data is retrieved from the Python code). By making it a dynamically linked library, you might get the code to work without changing the linker settings.

```

function pyRunString
  input String s;
external "C" annotation(Include="
#include <Python.h>

void pyRunString(const char *str)
{
  Py_SetProgramName("\pyRunString\"); /* optional but recommended */
  Py_Initialize();
  PyRun_SimpleString(str);
  Py_Finalize();
}
");
end pyRunString;

model CallExternalPython
algorithm
  pyRunString("
print 'Python says: simulation time'," +String(time)+"
");
end CallExternalPython;

```

```

>>> system("python-config --cflags > pycflags")
0
>>> system("python-config --ldflags > pyldflags")
0
>>> pycflags := stringReplace(readFile("pycflags"), "\n", "");
>>> pyldflags := stringReplace(readFile("pyldflags"), "\n", "");
>>> setCFlags(getCFlags()+pycflags)
true
>>> setLinkerFlags(getLinkerFlags()+pyldflags)
true
>>> simulate(CallExternalPython, stopTime=2)
record SimulationResult
  resultFile = "«DOCHOME»/CallExternalPython_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 2.0, numberOfIntervals = 500, tolerance = 1e-6",
  messages = "Python says: simulation time 0
Python says: simulation time 0
Python says: simulation time 2
",
  timeFrontend = 0.0046240260000000001,
  timeBackend = 0.003886963,
  timeSimCode = 0.045709666,
  timeTemplates = 0.003869896,
  timeCompile = 0.380508415,
  timeSimulation = 0.033802061,
  timeTotal = 0.472481829
end SimulationResult;

```

13.3 Calling OpenModelica from Python Code

This section describes a simple-minded approach to calling Python code from OpenModelica. For a description of Python scripting with OpenModelica, see *OMPpython – OpenModelica Python Interface*.

The interaction with Python can be performed in four different ways whereas one is illustrated below. Assume that we have the following Modelica code:

In the following Python (.py) files the above Modelica model is simulated via the OpenModelica scripting interface:

A second option of simulating the above Modelica model is to use the command `buildModel` instead of the `simulate`

Listing 13.4: CalledbyPython.mo

```

model CalledbyPython
  Real x(start=1.0), y(start=2.0);
  parameter Real b = 2.0;
equation
  der(x) = -b*y;
  der(y) = x;
end CalledbyPython;

```

Listing 13.5: PythonCaller.py

```

#!/usr/bin/python
import sys,os
global newb = 0.5
execfile('CreateMosFile.py')
os.popen(r"omc CalledbyPython.mos").read()
execfile('RetrResult.py')

```

Listing 13.6: CreateMosFile.py

```

#!/usr/bin/python
mos_file = open('CalledbyPython.mos','w', 1)
mos_file.write('loadFile("CalledbyPython.mo");\n')
mos_file.write('setComponentModifierValue(CalledbyPython,b,$Code(="+str(newb)+")); \n')
mos_file.write('simulate(CalledbyPython,stopTime=10);\n')
mos_file.close()

```

Listing 13.7: RetrResult.py

```

#!/usr/bin/python
def zeros(n): #
  vec = [0.0]
  for i in range(int(n)-1): vec = vec + [0.0]
  return vec
res_file = open("CalledbyPython_res.plt",'r',1)
line = res_file.readline()
size = int(res_file.readline().split('=')[1])
time = zeros(size)
y = zeros(size)
while line != ['DataSet: time\n']:
  line = res_file.readline().split(',')[0:1]
for j in range(int(size)):
  time[j]=float(res_file.readline().split(',')[0])
while line != ['DataSet: y\n']:
  line=res_file.readline().split(',')[0:1]
for j in range(int(size)):
  y[j]=float(res_file.readline().split(',')[1])
res_file.close()

```

command and setting the parameter value in the initial parameter file, CalledbyPython_init.txt instead of using the command setComponentModifierValue. Then the file CalledbyPython.exe is just executed.

The third option is to use the Corba interface for invoking the compiler and then just use the scripting interface to send commands to the compiler via this interface.

The fourth variant is to use external function calls to directly communicate with the executing simulation process.

OPENMODELICA PYTHON INTERFACE AND PYSIMULATOR

This chapter describes the OpenModelica Python integration facilities.

- OMPython – the OpenModelica Python scripting interface, see *OMPython – OpenModelica Python Interface*.
- **PySimulator** – a Python package that provides simulation and post processing/analysis tools integrated with OpenModelica, see *PySimulator*.

14.1 OMPython – OpenModelica Python Interface

OMPython – OpenModelica Python API is a free, open source, highly portable Python based interactive session handler for Modelica scripting. It provides the modeler with components for creating a complete Modelica modeling, compilation and simulation environment based on the latest OpenModelica library standard available. OMPython is architected to combine both the solving strategy and model building. So domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research. OMPython v2.0 is not a standalone package, it depends upon the OpenModelica installation.

OMPython v2.0 is implemented in Python using the OmniORB and OmniORBpy - high performance CORBA ORBs for Python and it supports the Modelica Standard Library version 3.2 that is included in starting with OpenModelica 1.9.2. It is now primarily available using the command `pip install ompython`, but it is also possible to run `python setup.py install` manually or use the version provided in the Windows installer.

14.1.1 Features of OMPython

OMPython provides user friendly features like:

- **Interactive session handling, parsing, interpretation of commands and** Modelica expressions for evaluation, simulation, plotting, etc.
- Interface to the latest OpenModelica API calls.
- **Optimized parser results that give control over every element of the** output.
- Helper functions to allow manipulation on Nested dictionaries.
- Easy access to the library and testing of OpenModelica commands.

14.1.2 Test Commands

To test the command outputs, simply create an OMCSession object by importing from the OMPython library within Python interpreter. The module allows you to interactively send commands to the OMC server and display their output.

To get started, create an OMCSession object:

```
>>> from OMPython import OMCSession
>>> omc = OMCSession()
```

```
>>> omc.sendExpression("getVersion() ")
v1.9.4-dev.608+g7458c24
>>> omc.sendExpression("cd() ")
«DOCHOME»
>>> omc.sendExpression("loadModel(Modelica) ")
True
>>> omc.sendExpression("loadFile(getInstallationDirectoryPath() + \"/share/doc/omc/testmodels/BouncingBall.mo") ")
True
>>> omc.sendExpression("instantiateModel(BouncingBall) ")
class BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(start = 1.0) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new;
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0.0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then (-e) * pre(v) else 0.0;
    flying = v_new > 0.0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

We get the name and other properties of a class:

```
>>> omc.sendExpression("getClassNames() ")
('BouncingBall', 'ModelicaServices', 'Complex', 'Modelica')
>>> omc.sendExpression("isPartial(BouncingBall) ")
False
>>> omc.sendExpression("isPackage(BouncingBall) ")
False
>>> omc.sendExpression("isModel(BouncingBall) ")
True
>>> omc.sendExpression("checkModel(BouncingBall) ")
Check of BouncingBall completed successfully.
Class BouncingBall has 6 equation(s) and 6 variable(s).
1 of these are trivial equation(s).
>>> omc.sendExpression("getClassRestriction(BouncingBall) ")
model
>>> omc.sendExpression("getClassInformation(BouncingBall) ")
('model', '', False, False, False, '«OPENMODELICAHOME»/share/doc/omc/testmodels/BouncingBall.mo',
>>> omc.sendExpression("getConnectionCount(BouncingBall) ")
0
>>> omc.sendExpression("getInheritanceCount(BouncingBall) ")
0
>>> omc.sendExpression("getComponentModifierValue(BouncingBall,e) ")
0.7
>>> omc.sendExpression("checkSettings() ")
{'RTLIBS': ' -lOpenModelicaRuntimeC -llapack -lblas -lm -lomcgc -lpthread -rdynamic', 'OMC_FOUND'
```

The common combination of a simulation followed by getting a value and doing a plot:

```
>>> omc.sendExpression("simulate(BouncingBall, stopTime=3.0)")
{'timeCompile': 0.236631199, 'simulationOptions': "startTime = 0.0, stopTime = 3.0, numberOfInterp
"Warning: The initial conditions are not fully specified. Use +d=initialization for more informat
"
>>> omc.sendExpression("val(h , 2.0)")
0.0423943077288
```

Import As Library

To use the module from within another python program, simply import OMCSession from within the using program. Make use of the execute() function of the OMPython library to send commands to the OMC server.

For example:

```
answer = OMPython.execute(cmd)
```

Full example:

```
# test.py
from OMPython import OMCSession
omc = OMCSession()
cmds = [
    "loadModel(Modelica)",
    "model test end test;",
    'loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.mo")',
    "getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)",
    "getElementsInfo(Modelica.Electrical.Analog.Basic.Resistor)",
    "simulate(BouncingBall)",
    "plot(h)"
]
for cmd in cmds:
    answer = omc.sendExpression(cmd)
    print "\\nResult:\\n%s"%answer
```

14.1.3 Implementation

Client Implementation

The OpenModelica Python API Interface – OMPython, attempts to mimic the OMSHELL's style of operations.

OMPython is designed to,

- Initialize the CORBA communication.
- Send commands to the Omc server via the CORBA interface.
- Receive the string results.
- Use the Parser module to format the results.
- Return or display the results.

14.2 PySimulator

PySimulator provides a graphical user interface for performing analyses and simulating different model types (currently Functional Mockup Units and Modelica Models are supported), plotting result variables and applying simulation result analysis tools like Fast Fourier Transform.

Read more about the PySimulator at <https://github.com/PySimulator/PySimulator>.

SCRIPTING API

The following are short summaries of OpenModelica scripting commands. These commands are useful for loading and saving classes, reading and storing data, plotting of results, and various other tasks.

The arguments passed to a scripting function should follow syntactic and typing rules for Modelica and for the scripting function in question. In the following tables we briefly indicate the types or character of the formal parameters to the functions by the following notation:

- String typed argument, e.g. “hello”, “myfile.mo”.
- **TypeName** – class, package or function name, e.g. **MyClass**, **Modelica.Math**.
- VariableName – variable name, e.g. `v1`, `v2`, `vars1[2]` . `x`, etc.
- Integer or Real typed argument, e.g. 35, 3.14, `xintvariable`.
- options – optional parameters with named formal parameter passing.

15.1 OpenModelica Modelica Scripting Commands

The following are brief descriptions of the scripting commands available in the OpenModelica environment.

15.2 OpenModelica API Calls

All OpenModelica API commands shown in alphabetic order:

15.2.1 GC_expand_hp

Forces the GC to expand the heap to accommodate more data.

```
function GC_expand_hp
  input Integer size;
  output Boolean success;
end GC_expand_hp;
```

15.2.2 GC_gcollect_and_unmap

Forces GC to collect and unmap memory (we use it before we start and wait for memory-intensive tasks in child processes).

15.2.3 addClassAnnotation

Used to set annotations, like Diagrams and Icons in classes. The function is given the name of the class and the annotation to set.

Usage: `addClassAnnotation(Modelica, annotate = Documentation(info = "<html></html>"))`

```
function addClassAnnotation
  input TypeName class_;
  input Expression annotate;
  output Boolean bool;
end addClassAnnotation;
```

15.2.4 alarm

Like `alarm(2)`.

Note that OpenModelica also sends SIGALRM to the process group when the alarm is triggered (in order to kill running simulations).

```
impure function alarm
  input Integer seconds;
  output Integer previousSeconds;
end alarm;
```

15.2.5 appendEnvironmentVar

Appends a variable to the environment variables list.

```
function appendEnvironmentVar
  input String var;
  input String value;
  output String result "returns \"error\" if the variable could not be appended";
end appendEnvironmentVar;
```

15.2.6 basename

Returns the base name (file part) of a file path. Similar to `basename(3)`, but with the safety of Modelica strings.

```
function basename
  input String path;
  output String basename;
end basename;
```

15.2.7 buildModel

builds a modelica model by generating c code and build it. It does not run the code! The only required argument is the `className`, while all others have some default values. `simulate(className, [startTime], [stopTime], [numberOfIntervals], [tolerance], [method], [fileNamePrefix], [options], [outputFormat], [variableFilter], [cflags], [simflags])` Example command: `simulate(A);`

```
function buildModel
  input TypeName className "the class that should be built";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file. <default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> = 1e-6";
  input String method = "<default>" "integration method used for simulation. <default> = dassl";
```

```

input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
input String options = "<default>" "options. <default> = \"\"";
input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
input String variableFilter = ".*" "Filter for variables that should store in result file. <def
input String cflags = "<default>" "cflags. <default> = \"\"";
input String simflags = "<default>" "simflags. <default> = \"\"";
output String[2] buildModelResults;
end buildModel;

```

15.2.8 buildModelFMU

translates a modelica model into a Functional Mockup Unit. The only required argument is the className, while all others have some default values. Example command: buildModelFMU(className, version="2.0");

```

function buildModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation), me_cs (both mod
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"className\"";
  input String platforms[:] = {"dynamic"} "The list of platforms to generate code for. \"dynamic\
  output String generatedFileName "Returns the full path of the generated FMU.";
end buildModelFMU;

```

15.2.9 buildOpenTURNSInterface

generates wrapper code for OpenTURNS

```

function buildOpenTURNSInterface
  input TypeName className;
  input String pythonTemplateFile;
  input Boolean showFlatModelica = false;
  output String outPythonScript;
end buildOpenTURNSInterface;

```

15.2.10 cd

change directory to the given path (which may be either relative or absolute) returns the new working directory on success or a message on failure if the given path is the empty string, the function simply returns the current working directory.

```

function cd
  input String newWorkingDirectory = "";
  output String workingDirectory;
end cd;

```

15.2.11 checkAllModelsRecursive

Checks all models recursively and returns number of variables and equations.

```

function checkAllModelsRecursive
  input TypeName className;
  input Boolean checkProtected = false "Checks also protected classes if true";
  output String result;
end checkAllModelsRecursive;

```

15.2.12 checkCodeGraph

Checks if the given taskgraph has the same structure as the graph described in the codefile.

```
function checkCodeGraph
  input String graphfile;
  input String codefile;
  output String[:] result;
end checkCodeGraph;
```

15.2.13 checkInterfaceOfPackages

Verifies the `__OpenModelica_Interface=`str annotation of all loaded packages with respect to the given main class.

For each row in the dependencyMatrix, the first element is the name of a dependency type. The rest of the elements are the other accepted dependency types for this one (frontend can call frontend and util, for example). Empty entries are ignored (necessary in order to have a rectangular matrix).

```
function checkInterfaceOfPackages
  input TypeName cl;
  input String dependencyMatrix[:, :];
  output Boolean success;
end checkInterfaceOfPackages;
```

15.2.14 checkModel

Checks a model and returns number of variables and equations.

```
function checkModel
  input TypeName className;
  output String result;
end checkModel;
```

15.2.15 checkSettings

Display some diagnostics.

```
function checkSettings
  output CheckSettingsResult result;
end checkSettings;
```

15.2.16 checkTaskGraph

Checks if the given taskgraph has the same structure as the reference taskgraph and if all attributes are set correctly.

```
function checkTaskGraph
  input String filename;
  input String reffilename;
  output String[:] result;
end checkTaskGraph;
```

15.2.17 classAnnotationExists

Check if annotation exists

Returns true if `className` has a class annotation called `annotationName`.

```
function classAnnotationExists
  input TypeName className;
  input TypeName annotationName;
  output Boolean exists;
end classAnnotationExists;
```

15.2.18 clear

Clears everything: symboltable and variables.

```
function clear
  output Boolean success;
end clear;
```

15.2.19 clearCommandLineOptions

Resets all command-line flags to their default values.

```
function clearCommandLineOptions
  output Boolean success;
end clearCommandLineOptions;
```

15.2.20 clearDebugFlags

Resets all debug flags to their default values.

```
function clearDebugFlags
  output Boolean success;
end clearDebugFlags;
```

15.2.21 clearMessages

Clears the error buffer.

```
function clearMessages
  output Boolean success;
end clearMessages;
```

15.2.22 clearProgram

Clears loaded .

```
function clearProgram
  output Boolean success;
end clearProgram;
```

15.2.23 clearVariables

Clear all user defined variables.

```
function clearVariables
  output Boolean success;
end clearVariables;
```

15.2.24 closeSimulationResultFile

Closes the current simulation result file. Only needed by Windows. Windows cannot handle reading and writing to the same file from different processes. To allow OMEdit to make successful simulation again on the same file we must close the file after reading the Simulation Result Variables. Even OMEdit only use this API for Windows.

```
function closeSimulationResultFile
  output Boolean success;
end closeSimulationResultFile;
```

15.2.25 codeToString

```
function codeToString
  input $Code className;
  output String string;
end codeToString;
```

15.2.26 compareFiles

Compares *file1* and *file2* and returns true if their content is equal, otherwise false.

```
impure function compareFiles
  input String file1;
  input String file2;
  output Boolean isEqual;
end compareFiles;
```

15.2.27 compareFilesAndMove

Compares *newFile* and *oldFile*. If they differ, overwrite *oldFile* with *newFile*

Basically: `test -f ../oldFile && cmp newFile oldFile || mv newFile oldFile`

```
impure function compareFilesAndMove
  input String newFile;
  input String oldFile;
  output Boolean success;
end compareFilesAndMove;
```

15.2.28 compareSimulationResults

compares simulation results.

```
function compareSimulationResults
  input String filename;
  input String reffilename;
  input String logfilename;
  input Real relTol = 0.01;
  input Real absTol = 0.0001;
  input String[:] vars = fill("", 0);
  output String[:] result;
end compareSimulationResults;
```

15.2.29 convertUnits

Returns the scale factor and offsets used when converting two units.

Returns false if the types are not compatible and should not be converted.

```
function convertUnits
  input String s1;
  input String s2;
  output Boolean unitsCompatible;
  output Real scaleFactor1;
  output Real offset1;
  output Real scaleFactor2;
  output Real offset2;
end convertUnits;
```

15.2.30 copyClass

Copies a class within the same level

```
function copyClass
  input TypeName className "the class that should be copied";
  input String newClassName "the name for new class";
  input TypeName withIn = $Code(TopLevel) "the with in path for new class";
  output Boolean result;
end copyClass;
```

15.2.31 countMessages

Returns the total number of messages in the error buffer, as well as the number of errors and warnings.

```
function countMessages
  output Integer numMessages;
  output Integer numErrors;
  output Integer numWarnings;
end countMessages;
```

15.2.32 deleteFile

Deletes a file with the given name.

```
function deleteFile
  input String fileName;
  output Boolean success;
end deleteFile;
```

15.2.33 diffModelicaFileListings

Creates diffs of two strings corresponding to Modelica files

Creates diffs of two strings (before and after) corresponding to Modelica files. The diff is specialized to handle the *list* API moving comments around in the file and introducing or deleting whitespace.

The output can be chosen to be a colored diff (for terminals), XML, or the final text (deletions removed).

```
function diffModelicaFileListings
  input String before, after;
  input DiffFormat diffFormat = DiffFormat.color;
```

```
output String result;  
end diffModelicaFileListings;
```

15.2.34 diffSimulationResults

compares simulation results.

Takes two result files and compares them. By default, all selected variables that are not equal in the two files are output to diffPrefix.varName.csv.

The output is the names of the variables for which files were generated.

```
function diffSimulationResults  
  input String actualFile;  
  input String expectedFile;  
  input String diffPrefix;  
  input Real relTol = 1e-3 "y tolerance";  
  input Real relTolDiffMinMax = 1e-4 "y tolerance based on the difference between the maximum and  
  input Real rangeDelta = 0.002 "x tolerance";  
  input String[:] vars = fill("", 0);  
  input Boolean keepEqualResults = false;  
  output Boolean success;  
  output String[:] failVars;  
end diffSimulationResults;
```

15.2.35 diffSimulationResultsHtml

Takes two result files and compares them. By default, all selected variables that are not equal in the two files are output to diffPrefix.varName.csv.

The output is the names of the variables for which files were generated.

```
function diffSimulationResultsHtml  
  input String var;  
  input String actualFile;  
  input String expectedFile;  
  input Real relTol = 1e-3 "y tolerance";  
  input Real relTolDiffMinMax = 1e-4 "y tolerance based on the difference between the maximum and  
  input Real rangeDelta = 0.002 "x tolerance";  
  output String html;  
end diffSimulationResultsHtml;
```

15.2.36 directoryExists

```
function directoryExists  
  input String dirName;  
  output Boolean exists;  
end directoryExists;
```

15.2.37 dirname

Returns the directory name of a file path. Similar to `dirname(3)`, but with the safety of Modelica strings.

```
function dirname  
  input String path;  
  output String dirname;  
end dirname;
```

15.2.38 dumpXMLDAE

Outputs the DAE system corresponding to a specific model.

Valid translationLevel strings are: *flat*, *optimiser* (runs the backend until sorting/matching), *backEnd*, or *stateSpace*.

```
function dumpXMLDAE
  input TypeName className;
  input String translationLevel = "flat" "flat, optimiser, backEnd, or stateSpace";
  input Boolean addOriginalIncidenceMatrix = false;
  input Boolean addSolvingInfo = false;
  input Boolean addMathMLCode = false;
  input Boolean dumpResiduals = false;
  input String fileNamePrefix = "<default>" "this is the className in string form by default";
  input String rewriteRulesFile = "" "the file from where the rewriteRules are read, default is empty";
  output Boolean success "if the function succeeded true/false";
  output String xmlfileName "the Xml file";
end dumpXMLDAE;
```

15.2.39 echo

echo(false) disables Interactive output, echo(true) enables it again.

```
function echo
  input Boolean setEcho;
  output Boolean newEcho;
end echo;
```

15.2.40 escapeXML

```
function escapeXML
  input String inStr;
  output String outStr;
end escapeXML;
```

15.2.41 exit

Forces omc to quit with the given exit status.

```
function exit
  input Integer status;
end exit;
```

15.2.42 exportToFigaro

```
function exportToFigaro
  input TypeName path;
  input String directory = cd();
  input String database;
  input String mode;
  input String options;
  input String processor;
  output Boolean success;
end exportToFigaro;
```

15.2.43 extendsFrom

returns true if the given class extends from the given base class

```
function extendsFrom
  input TypeName className;
  input TypeName baseClassName;
  output Boolean res;
end extendsFrom;
```

15.2.44 filterSimulationResults

Takes one simulation result and filters out the selected variables only, producing the output file.

If numberOfIntervals<>0, re-sample to that number of intervals, ignoring event points (might be changed in the future).

```
function filterSimulationResults
  input String inFile;
  input String outFile;
  input String[:] vars;
  input Integer numberOfIntervals = 0 "0=Do not resample";
  output Boolean success;
end filterSimulationResults;
```

15.2.45 generateCode

The input is a function name for which C-code is generated and compiled into a dll/so

```
function generateCode
  input TypeName className;
  output Boolean success;
end generateCode;
```

15.2.46 generateEntryPoint

Generates a main() function that calls the given MetaModelica endpoint (assumed to have input list and no outputs).

```
function generateEntryPoint
  input String fileName;
  input TypeName entryPoint;
  input String url = "https://trac.openmodelica.org/OpenModelica/newticket";
end generateEntryPoint;
```

15.2.47 generateHeader

```
function generateHeader
  input String fileName;
  output Boolean success;
end generateHeader;
```

15.2.48 generateScriptingAPI

Work in progress

Returns OpenModelica.Scripting API entry points for the classes that we can automatically generate entry points for.

The entry points are MetaModelica code calling CevalScript directly, and Qt/C++ code that calls the MetaModelica code.

```
function generateScriptingAPI
  input TypeName cl;
  input String name;
  output Boolean success;
  output String moFile;
  output String qtFile;
  output String qtHeader;
end generateScriptingAPI;
```

15.2.49 generateSeparateCode

Under construction.

```
function generateSeparateCode
  input TypeName className;
  input Boolean cleanCache = false "If true, the cache is reset between each generated package. T";
  output Boolean success;
end generateSeparateCode;
```

15.2.50 generateSeparateCodeDependencies

Under construction.

```
function generateSeparateCodeDependencies
  input String stampSuffix = ".c" "Suffix to add to dependencies (often .c.stamp)";
  output String[:] dependencies;
end generateSeparateCodeDependencies;
```

15.2.51 generateSeparateCodeDependenciesMakefile

Under construction.

```
function generateSeparateCodeDependenciesMakefile
  input String filename "The file to write the makefile to";
  input String directory = "" "The relative path of the generated files";
  input String suffix = ".c" "Often .stamp since we do not update all the files";
  output Boolean success;
end generateSeparateCodeDependenciesMakefile;
```

15.2.52 getAlgorithmCount

Counts the number of Algorithm sections in a class.

```
function getAlgorithmCount
  input TypeName class_;
  output Integer count;
end getAlgorithmCount;
```

15.2.53 getAlgorithmItemsCount

Counts the number of Algorithm items in a class.

```
function getAlgorithmItemsCount
  input TypeName class_;
  output Integer count;
end getAlgorithmItemsCount;
```

15.2.54 getAnnotationCount

Counts the number of Annotation sections in a class.

```
function getAnnotationCount
  input TypeName class_;
  output Integer count;
end getAnnotationCount;
```

15.2.55 getAnnotationVersion

Returns the current annotation version.

```
function getAnnotationVersion
  output String annotationVersion;
end getAnnotationVersion;
```

15.2.56 getAstAsCorbaString

Print the whole AST on the CORBA format for records, e.g. record Absyn.PROGRAM classes = ..., **within_** = ..., globalBuildTimes = ... end Absyn.PROGRAM;

```
function getAstAsCorbaString
  input String fileName = "<interactive>";
  output String result "returns the string if fileName is interactive; else it returns ok or error";
end getAstAsCorbaString;
```

15.2.57 getAvailableIndexReductionMethods

```
function getAvailableIndexReductionMethods
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableIndexReductionMethods;
```

15.2.58 getAvailableLibraries

Looks for all libraries that are visible from the *getModelicaPath()*.

```
function getAvailableLibraries
  output String[:] libraries;
end getAvailableLibraries;
```

15.2.59 getAvailableMatchingAlgorithms

```
function getAvailableMatchingAlgorithms
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableMatchingAlgorithms;
```

15.2.60 getAvailableTearingMethods

```
function getAvailableTearingMethods
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableTearingMethods;
```

15.2.61 getBooleanClassAnnotation

Check if annotation exists and returns its value

Returns the value of the class annotation **annotationName** of class **className**. If there is no such annotation, or if it is not true or false, this function fails.

```
function getBooleanClassAnnotation
  input TypeName className;
  input TypeName annotationName;
  output Boolean value;
end getBooleanClassAnnotation;
```

15.2.62 getBuiltinType

Returns the builtin type e.g Real, Integer, Boolean & String of the class.

```
function getBuiltinType
  input TypeName cl;
  output String name;
end getBuiltinType;
```

15.2.63 getCFlags

CFLAGS

See *setCFlags()* for details.

```
function getCFlags
  output String outString;
end getCFlags;
```

15.2.64 getCXXCompiler

CXX

```
function getCXXCompiler
  output String compiler;
end getCXXCompiler;
```

15.2.65 getClassComment

Returns the class comment.

```
function getClassComment
  input TypeName cl;
  output String comment;
end getClassComment;
```

15.2.66 getClassInformation

Returns class information for the given class.

The dimensions are returned as an array of strings. The string is the textual representation of the dimension (they are not evaluated to Integers).

```
function getClassInformation
  input TypeName cl;
  output String restriction, comment;
  output Boolean partialPrefix, finalPrefix, encapsulatedPrefix;
  output String fileName;
  output Boolean fileReadOnly;
  output Integer lineNumberStart, columnNumberStart, lineNumberEnd, columnNumberEnd;
  output String dimensions[:];
end getClassInformation;
```

15.2.67 getClassNames

Returns the list of class names defined in the class.

```
function getClassNames
  input TypeName class_ = $Code(AllLoadedClasses);
  input Boolean recursive = false;
  input Boolean qualified = false;
  input Boolean sort = false;
  input Boolean builtin = false "List also builtin classes if true";
  input Boolean showProtected = false "List also protected classes if true";
  output TypeName classNames[:];
end getClassNames;
```

15.2.68 getClassRestriction

Returns the restriction of the given class.

```
function getClassRestriction
  input TypeName cl;
  output String restriction;
end getClassRestriction;
```

15.2.69 getClassesInModelicaPath

MathCore-specific or not? Who knows!

```
function getClassesInModelicaPath
  output String classesInModelicaPath;
end getClassesInModelicaPath;
```

15.2.70 getCompileCommand

```
function getCompileCommand
  output String compileCommand;
end getCompileCommand;
```

15.2.71 getCompiler

CC

```
function getCompiler
  output String compiler;
end getCompiler;
```

15.2.72 GetComponentModifierNames

Returns the list of class component modifiers.

```
function GetComponentModifierNames
  input TypeName class_;
  input String componentName;
  output String[:] modifiers;
end GetComponentModifierNames;
```

15.2.73 getComponentsTest

Returns the components found in the given class.

```
function getComponentsTest
  input TypeName name;
  output Component[:] components;
  record Component
    String className;
    // when building record the constructor. Records are allowed to contain only components of ba
    String name;
    String comment;
    Boolean isProtected;
    Boolean isFinal;
    Boolean isFlow;
    Boolean isStream;
    Boolean isReplaceable;
    String variability "constant", "parameter", "discrete", "";
    String innerOuter "inner", "outer", "";
    String inputOutput "input", "output", "";
    String dimensions[:];
  end Component;
end getComponentsTest;
```

15.2.74 getConfigFlagValidOptions

Returns the list of valid options for a string config flag, and the description strings for these options if available

```
function getConfigFlagValidOptions
  input String flag;
  output String validOptions[:];
  output String mainDescription;
  output String descriptions[:];
end getConfigFlagValidOptions;
```

15.2.75 getDefaultOpenCLDevice

Returns the id for the default OpenCL device to be used.

```
function getDefaultOpenCLDevice
  output Integer defdevid;
end getDefaultOpenCLDevice;
```

15.2.76 getDerivedClassModifierNames

Returns the derived class modifier names. Example command: type Resistance = Real(final quantity="Resistance",final unit="Ohm"); getDerivedClassModifierNames(Resistance) => {"quantity","unit"}

Finds the modifiers of the derived class.

```
function getDerivedClassModifierNames
  input TypeName className;
  output String[:] modifierNames;
end getDerivedClassModifierNames;
```

15.2.77 getDerivedClassModifierValue

Returns the derived class modifier value. Example command: type Resistance = Real(final quantity="Resistance",final unit="Ohm"); getDerivedClassModifierValue(Resistance, unit); => " = "Ohm"" getDerivedClassModifierValue(Resistance, quantity); => " = "Resistance""

Finds the modifier value of the derived class.

```
function getDerivedClassModifierValue
  input TypeName className;
  input TypeName modifierName;
  output String modifierValue;
end getDerivedClassModifierValue;
```

15.2.78 getDocumentationAnnotation

Returns the documentaiton annotation defined in the class.

```
function getDocumentationAnnotation
  input TypeName cl;
  output String out[2] "{info,revision} TODO: Should be changed to have 2 outputs instead of an a
end getDocumentationAnnotation;
```

15.2.79 getEnvironmentVar

Returns the value of the environment variable.

```
function getEnvironmentVar
  input String var;
  output String value "returns empty string on failure";
end getEnvironmentVar;
```

15.2.80 getEquationCount

Counts the number of Equation sections in a class.

```
function getEquationCount
  input TypeName class_;
  output Integer count;
end getEquationCount;
```

15.2.81 getEquationItemsCount

Counts the number of Equation items in a class.

```
function getEquationItemsCount
  input TypeName class_;
  output Integer count;
end getEquationItemsCount;
```

15.2.82 getErrorString

Returns the current error message. [file.mo:n:n-n:n:b] Error: message

Returns a user-friendly string containing the errors stored in the buffer. With warningsAsErrors=true, it reports warnings as if they were errors.

```
function getErrorString
  input Boolean warningsAsErrors = false;
  output String errorString;
end getErrorString;
```

15.2.83 getImportCount

Counts the number of Import sections in a class.

```
function getImportCount
  input TypeName class_;
  output Integer count;
end getImportCount;
```

15.2.84 getIndexReductionMethod

```
function getIndexReductionMethod
  output String selected;
end getIndexReductionMethod;
```

15.2.85 getInheritedClasses

Returns the list of inherited classes.

```
function getInheritedClasses
  input TypeName name;
  output TypeName inheritedClasses[:];
end getInheritedClasses;
```

15.2.86 getInitialAlgorithmCount

Counts the number of Initial Algorithm sections in a class.

```
function getInitialAlgorithmCount
  input TypeName class_;
  output Integer count;
end getInitialAlgorithmCount;
```

15.2.87 getInitialAlgorithmItemsCount

Counts the number of Initial Algorithm items in a class.

```
function getInitialAlgorithmItemsCount
  input TypeName class_;
  output Integer count;
end getInitialAlgorithmItemsCount;
```

15.2.88 getInitialEquationCount

Counts the number of Initial Equation sections in a class.

```
function getInitialEquationCount
  input TypeName class_;
  output Integer count;
end getInitialEquationCount;
```

15.2.89 getInitialEquationItemsCount

Counts the number of Initial Equation items in a class.

```
function getInitialEquationItemsCount
  input TypeName class_;
  output Integer count;
end getInitialEquationItemsCount;
```

15.2.90 getInstallationDirectoryPath

This returns OPENMODELICAHOME if it is set; on some platforms the default path is returned if it is not set.

```
function getInstallationDirectoryPath
  output String installationDirectoryPath;
end getInstallationDirectoryPath;
```

15.2.91 getLanguageStandard

Returns the current Modelica Language Standard in use.

```
function getLanguageStandard
  output String outVersion;
end getLanguageStandard;
```

15.2.92 getLinker

```
function getLinker
  output String linker;
end getLinker;
```

15.2.93 getLinkerFlags

```
function getLinkerFlags
  output String linkerFlags;
end getLinkerFlags;
```

15.2.94 getLoadedLibraries

Returns a list of names of libraries and their path on the system, for example:

```
{{"Modelica", "/usr/lib/omlibrary/Modelica 3.2.1"}, {"ModelicaServices", "/usr/lib/omlibrary/Modelica
```

```
function getLoadedLibraries
  output String[:, 2] libraries;
end getLoadedLibraries;
```

15.2.95 getMatchingAlgorithm

```
function getMatchingAlgorithm
  output String selected;
end getMatchingAlgorithm;
```

15.2.96 getMemorySize

Retrieves the physical memory size available on the system in megabytes.

```
function getMemorySize
  output Real memory(unit = "MiB");
end getMemorySize;
```

15.2.97 getMessagesString

see `getErrorString()`

```
function getMessagesString
  output String messagesString;
end getMessagesString;
```

15.2.98 getModelicaPath

Get the Modelica Library Path.

The MODELICAPATH is list of paths to search when trying to *load a library*. It is a string separated by colon (:) on all OSes except Windows, which uses semicolon (;).

To override the default path (`OPENMODELICAHOME/lib/omlibrary/~/openmodelica/libraries/`), set the environment variable `OPENMODELICALIBRARY=...`

```
function getModelicaPath
  output String modelicaPath;
end getModelicaPath;
```

15.2.99 getNoSimplify

Returns true if noSimplify flag is set.

```
function getNoSimplify
  output Boolean noSimplify;
end getNoSimplify;
```

15.2.100 getNthAlgorithm

Returns the Nth Algorithm section.

```
function getNthAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithm;
```

15.2.101 getNthAlgorithmItem

Returns the Nth Algorithm Item.

```
function getNthAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithmItem;
```

15.2.102 getNthAnnotationString

Returns the Nth Annotation section as string.

```
function getNthAnnotationString
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAnnotationString;
```

15.2.103 getNthEquation

Returns the Nth Equation section.

```
function getNthEquation
  input TypeName class_;
  input Integer index;
  output String result;
end getNthEquation;
```

15.2.104 getNthEquationItem

Returns the Nth Equation Item.

```
function getNthEquationItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthEquationItem;
```

15.2.105 getNthImport

Returns the Nth Import as string.

```
function getNthImport
  input TypeName class_;
  input Integer index;
  output String out[3] "{ \"Path\", \"Id\", \"Kind\" }";
end getNthImport;
```

15.2.106 getNthInitialAlgorithm

Returns the Nth Initial Algorithm section.

```
function getNthInitialAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithm;
```

15.2.107 getNthInitialAlgorithmItem

Returns the Nth Initial Algorithm Item.

```
function getNthInitialAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithmItem;
```

15.2.108 getNthInitialEquation

Returns the Nth Initial Equation section.

```
function getNthInitialEquation
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialEquation;
```

15.2.109 getNthInitialEquationItem

Returns the Nth Initial Equation Item.

```
function getNthInitialEquationItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialEquationItem;
```

15.2.110 getOrderConnections

Returns true if orderConnections flag is set.

```
function getOrderConnections
  output Boolean orderConnections;
end getOrderConnections;
```

15.2.111 getPackages

Returns the list of packages defined in the class.

```
function getPackages
  input TypeName class_ = $Code(AllLoadedClasses);
  output TypeName classNames[:];
end getPackages;
```

15.2.112 getParameterNames

Returns the list of parameters of the class.

```
function getParameterNames
  input TypeName class_;
  output String[:] parameters;
end getParameterNames;
```

15.2.113 getParameterValue

Returns the value of the parameter of the class.

```
function getParameterValue
  input TypeName class_;
  input String parameterName;
  output String parameterValue;
end getParameterValue;
```

15.2.114 getSettings

```
function getSettings
  output String settings;
end getSettings;
```

15.2.115 getShowAnnotations

```
function getShowAnnotations
  output Boolean show;
end getShowAnnotations;
```

15.2.116 getSimulationOptions

Returns the startTime, stopTime, tolerance, and interval based on the experiment annotation.

```
function getSimulationOptions
  input TypeName name;
  input Real defaultStartTime = 0.0;
  input Real defaultStopTime = 1.0;
  input Real defaultTolerance = 1e-6;
  input Integer defaultNumberOfIntervals = 500 "May be overridden by defining defaultInterval inst
  input Real defaultInterval = 0.0 "If = 0.0, then numberOfIntervals is used to calculate the step
  output Real startTime;
  output Real stopTime;
  output Real tolerance;
  output Integer numberOfIntervals;
```

```

output Real interval;
end getSimulationOptions;

```

15.2.117 getSourceFile

Returns the filename of the class.

```

function getSourceFile
  input TypeName class_;
  output String filename "empty on failure";
end getSourceFile;

```

15.2.118 getTearingMethod

```

function getTearingMethod
  output String selected;
end getTearingMethod;

```

15.2.119 getTempDirectoryPath

Returns the current user temporary directory location.

```

function getTempDirectoryPath
  output String tempDirectoryPath;
end getTempDirectoryPath;

```

15.2.120 getTimeStamp

The given class corresponds to a file (or a buffer), with a given last time this file was modified at the time of loading this file. The timestamp along with its String representation is returned.

```

function getTimeStamp
  input TypeName cl;
  output Real timeStamp;
  output String timeStampAsString;
end getTimeStamp;

```

15.2.121 getUsedClassNames

Returns the list of class names used in the total program defined by the class.

```

function getUsedClassNames
  input TypeName className;
  output TypeName classNames[:];
end getUsedClassNames;

```

15.2.122 getUses

Returns the libraries used by the package {{“Library1”,“Version”},{“Library2”,“Version”}}.

```

function getUses
  input TypeName pack;
  output String[:, :] uses;
end getUses;

```

15.2.123 getVectorizationLimit

```
function getVectorizationLimit
  output Integer vectorizationLimit;
end getVectorizationLimit;
```

15.2.124 getVersion

Returns the version of the Modelica compiler.

```
function getVersion
  input TypeName cl = $Code(OpenModelica);
  output String version;
end getVersion;
```

15.2.125 help

display the OpenModelica help text.

```
function help
  input String topic = "topics";
  output String helpText;
end help;
```

15.2.126 iconv

The iconv() function converts one multibyte characters from one character set to another. See man (3) iconv for more information.

```
function iconv
  input String string;
  input String from;
  input String to = "UTF-8";
  output String result;
end iconv;
```

15.2.127 importFMU

Imports the Functional Mockup Unit Example command: importFMU("A.fmu");

```
function importFMU
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files. <default> will
  input Integer loglevel = 3 "loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;loglevel_warnin
  input Boolean fullPath = false "When true the full output path is returned otherwise only the f
  input Boolean debugLogging = false "When true the FMU's debug output is printed.";
  input Boolean generateInputConnectors = true "When true creates the input connector pins.";
  input Boolean generateOutputConnectors = true "When true creates the output connector pins.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMU;
```

15.2.128 instantiateModel

Instantiates the class and returns the flat Modelica code.

```
function instantiateModel
  input TypeName className;
  output String result;
end instantiateModel;
```

15.2.129 isBlock

Returns true if the given class has restriction block.

```
function isBlock
  input TypeName cl;
  output Boolean b;
end isBlock;
```

15.2.130 isClass

Returns true if the given class has restriction class.

```
function isClass
  input TypeName cl;
  output Boolean b;
end isClass;
```

15.2.131 isConnector

Returns true if the given class has restriction connector or expandable connector.

```
function isConnector
  input TypeName cl;
  output Boolean b;
end isConnector;
```

15.2.132 isEnumeration

Returns true if the given class has restriction enumeration.

```
function isEnumeration
  input TypeName cl;
  output Boolean b;
end isEnumeration;
```

15.2.133 isExperiment

An experiment is defined as having annotation Experiment(stopTime=...)

```
function isExperiment
  input TypeName name;
  output Boolean res;
end isExperiment;
```

15.2.134 isFunction

Returns true if the given class has restriction function.

```
function isFunction
  input TypeName cl;
  output Boolean b;
end isFunction;
```

15.2.135 isModel

Returns true if the given class has restriction model.

```
function isModel
  input TypeName cl;
  output Boolean b;
end isModel;
```

15.2.136 isOperator

Returns true if the given class has restriction operator.

```
function isOperator
  input TypeName cl;
  output Boolean b;
end isOperator;
```

15.2.137 isOperatorFunction

Returns true if the given class has restriction “operator function”.

```
function isOperatorFunction
  input TypeName cl;
  output Boolean b;
end isOperatorFunction;
```

15.2.138 isOperatorRecord

Returns true if the given class has restriction “operator record”.

```
function isOperatorRecord
  input TypeName cl;
  output Boolean b;
end isOperatorRecord;
```

15.2.139 isOptimization

Returns true if the given class has restriction optimization.

```
function isOptimization
  input TypeName cl;
  output Boolean b;
end isOptimization;
```

15.2.140 isPackage

Returns true if the given class is a package.

```
function isPackage
  input TypeName c1;
  output Boolean b;
end isPackage;
```

15.2.141 isPartial

Returns true if the given class is partial.

```
function isPartial
  input TypeName c1;
  output Boolean b;
end isPartial;
```

15.2.142 isProtectedClass

Returns true if the given class c1 has class c2 as one of its protected class.

```
function isProtectedClass
  input TypeName c1;
  input String c2;
  output Boolean b;
end isProtectedClass;
```

15.2.143 isRecord

Returns true if the given class has restriction record.

```
function isRecord
  input TypeName c1;
  output Boolean b;
end isRecord;
```

15.2.144 isShortDefinition

returns true if the definition is a short class definition

```
function isShortDefinition
  input TypeName class_;
  output Boolean isShortCls;
end isShortDefinition;
```

15.2.145 isType

Returns true if the given class has restriction type.

```
function isType
  input TypeName c1;
  output Boolean b;
end isType;
```

15.2.146 linearize

creates a model with symbolic linearization matrixes

Creates a model with symbolic linearization matrixes.

At stopTime the linearization matrixes are evaluated and a modelica model is created.

The only required argument is the className, while all others have some default values.

Usage:

linearize(A, stopTime=0.0);

Creates the file "linear_A.mo" that contains the linearized matrixes at stopTime.

```
function linearize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file. <default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file. <default> = 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> = 1e-6";
  input String method = "<default>" "integration method used for simulation. <default> = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Filter for variables that should store in result file. <def
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String linearizationResult;
end linearize;
```

15.2.147 list

Lists the contents of the given class, or all loaded classes.

Pretty-prints a class definition.

15.2.148 Syntax

```
list (Modelica.Math.sin)
```

```
list (Modelica.Math.sin, interfaceOnly=true)
```

15.2.149 Description

list() pretty-prints the whole of the loaded AST while list(className) lists a class and its children. It keeps all annotations and comments intact but strips out any comments and normalizes white-space.

list(className, interfaceOnly=true) works on functions and pretty-prints only the interface parts (annotations and protected sections removed). String-comments on public variables are kept.

If the specified class does not exist (or is not a function when interfaceOnly is given), the empty string is returned.

```
function list
  input TypeName class_ = $Code(AllLoadedClasses);
  input Boolean interfaceOnly = false;
  input Boolean shortOnly = false "only short class definitions";
  input ExportKind exportKind = ExportKind.Absyn;
  output String contents;
end list;
```

15.2.150 listFile

Lists the contents of the file given by the class.

Lists the contents of the file given by the class. See also *list()*.

```
function listFile
  input TypeName class_;
  output String contents;
end listFile;
```

15.2.151 listVariables

Lists the names of the active variables in the scripting environment.

```
function listVariables
  output TypeName variables[:];
end listVariables;
```

15.2.152 loadFile

load file (*.mo) and merge it with the loaded AST.

Loads the given file using the given encoding.

Note that if the file basename is package.mo and the parent directory is the top-level class, the library structure is loaded as if loadModel(className) was called. Uses-annotations are respected if uses=true. The main difference from loadModel is that loadFile appends this directory to the MODELICAPATH (for this call only).

```
function loadFile
  input String fileName;
  input String encoding = "UTF-8";
  input Boolean uses = true;
  output Boolean success;
end loadFile;
```

15.2.153 loadFileInteractive

```
function loadFileInteractive
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end loadFileInteractive;
```

15.2.154 loadFileInteractiveQualified

```
function loadFileInteractiveQualified
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end loadFileInteractiveQualified;
```

15.2.155 loadFiles

load files (*.mo) and merges them with the loaded AST.

```
function loadFiles
  input String[:] fileNames;
  input String encoding = "UTF-8";
  input Integer numThreads = OpenModelica.Scripting.numProcessors();
  output Boolean success;
end loadFiles;
```

15.2.156 loadModel

Loads the Modelica Standard Library.

Loads a Modelica library.

15.2.157 Syntax

```
loadModel(Modelica)
```

```
loadModel(Modelica, {"3.2"})
```

15.2.158 Description

loadModel() begins by parsing the *getModelicaPath()*, and looking for candidate packages to load in the given paths (separated by : or ; depending on OS).

The candidate is selected by choosing the one with the highest priority, chosen by looking through the *priorityVersion* argument to the function. If the version searched for is “default”, the following special priority is used: no version name > highest main release > highest pre-release > lexical sort of others (see table below for examples). If none of the searched versions exist, false is returned and an error is added to the buffer.

A top-level package may either be defined in a file (“Modelica 3.2.mo”) or directory (“Modelica 3.2/package.mo”)

The encoding of any Modelica file in the package is assumed to be UTF-8. Legacy code may contain files in a different encoding. In order to handle this, add a file package.encoding at the top-level of the package, containing a single line with the name of the encoding in it. If your package contains files with mixed encodings and your system iconv supports UTF-8//IGNORE, you can ignore the bad characters in some of the files. You are recommended to convert your files to UTF-8 without byte-order mark.

Priority	Example
No version name	Modelica
Main release	Modelica 3.3
Pre-release	Modelica 3.3 Beta 1
Non-ordered	Modelica Trunk

15.2.159 Bugs

If loadModel(Modelica.XXX) is called, loadModel(Modelica) is executed instead, loading the complete library.

```

function loadModel
  input TypeName className;
  input String[:] priorityVersion = {"default"};
  input Boolean notify = false "Give a notification of the libraries and versions that were loaded";
  input String languageStandard = "" "Override the set language standard. Parse with the given set";
  input Boolean requireExactVersion = false "If the version is required to be exact, if there is";
  output Boolean success;
end loadModel;

```

15.2.160 loadModelica3D

Usage

Modelica3D requires some changes to the standard ModelicaServices in order to work correctly. These changes will make your MultiBody models unable to simulate because they need an object declared as:

```
inner ModelicaServices.Modelica3D.Controller m3d_control
```

Example session:

```

loadModelica3D();getErrorString();
loadString("model DoublePendulum
  extends Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum;
  inner ModelicaServices.Modelica3D.Controller m3d_control;
end DoublePendulum;");getErrorString();
system("python " + getInstallationDirectoryPath() + "/lib/omlibrary-modelica3d/osg-gtk/dbus-server.py");
simulate(DoublePendulum);getErrorString();

```

This API call will load the modified ModelicaServices 3.2.1 so Modelica3D runs. You can also simply call `loadModel(ModelicaServices,{"3.2.1 modelica3d"})`;

You will also need to start an m3d backend to render the results. We hid them in `$OPENMODELICAHOME/lib/omlibrary-modelica3d/osg-gtk/dbus-server.py` (or blender2.59).

For more information and example models, visit the [Modelica3D wiki](#).

```

function loadModelica3D
  input String version = "3.2.1";
  output Boolean status;
end loadModelica3D;

```

15.2.161 loadString

Parses the data and merges the resulting AST with the loaded AST. If a filename is given, it is used to provide error-messages as if the string was read in binary format from a file with the same name. The file is converted to UTF-8 from the given character set. NOTE: Encoding is deprecated as ALL strings are now UTF-8 encoded.

```

function loadString
  input String data;
  input String filename = "<interactive>";
  input String encoding = "UTF-8";
  output Boolean success;
end loadString;

```

15.2.162 mkdir

create directory of given path (which may be either relative or absolute) returns true if directory was created or already exists.

```
function mkdir
  input String newDirectory;
  output Boolean success;
end mkdir;
```

15.2.163 moveClass

moves a class up or down depending on the given direction, it returns true if the move was performed or false if we could not move the class

```
function moveClass
  input TypeName className "the class that should be moved";
  input String direction "up or down";
  output Boolean result;
end moveClass;
```

15.2.164 ngspicetoModelica

Converts ngspice netlist to Modelica code. Modelica file is created in the same directory as netlist file.

```
function ngspicetoModelica
  input String netlistfileName;
  output Boolean success = false;
end ngspicetoModelica;
```

15.2.165 numProcessors

Returns the number of processors (if compiled against hwloc) or hardware threads (if using sysconf) available to OpenModelica.

```
function numProcessors
  output Integer result;
end numProcessors;
```

15.2.166 optimize

optimize a modelica/optimica model by generating c code, build it and run the optimization executable. The only required argument is the className, while all others have some default values. simulate(className, [startTime], [stopTime], [numberOfIntervals], [stepSize], [tolerance], [fileNamePrefix], [options], [outputFormat], [variableFilter], [cflags], [simflags]) Example command: simulate(A);

```
function optimize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file. <default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file. <default> = 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> = 1e-6";
  input String method = DAE.SCONST("optimization") "optimize a modelica/optimica model.";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Filter for variables that should store in result file. <def";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
```

```

output String optimizationResults;
end optimize;

```

15.2.167 parseFile

```

function parseFile
input String filename;
input String encoding = "UTF-8";
output TypeName names[:];
end parseFile;

```

15.2.168 parseString

```

function parseString
input String data;
input String filename = "<interactive>";
output TypeName names[:];
end parseString;

```

15.2.169 plot

Launches a plot window using OMPlot.

Launches a plot window using OMPlot. Returns true on success.

Example command sequences:

- simulate(A);plot({x,y,z});
- simulate(A);plot(x, externalWindow=true);
- simulate(A,fileNamePrefix="B");simulate(C);plot(z,fileName="B.mat",legend=false);

```

function plot
input VariableNames vars "The variables you want to plot";
input Boolean externalWindow = false "Opens the plot in a new plot window";
input String fileName = "<default>" "The filename containing the variables. <default> will read
input String title = "" "This text will be used as the diagram title.";
input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed, none.";
input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically sc
input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scal
input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";
input String yLabel = "" "This text will be used as the vertical label in the diagram.";
input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is visible in the di
input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is visible in the diag
input Real curveWidth = 1.0 "Sets the width of the curve.";
input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2, DotLine=3,
input String legendPosition = "top" "Sets the POSITION of the legend i.e left, right, top, bott
input String footer = "" "This text will be used as the diagram footer.";
input Boolean autoScale = true "Use auto scale while plotting.";
input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call callback function e
output Boolean success "Returns true on success";
end plot;

```

15.2.170 plotAll

Works in the same way as plot(), but does not accept any variable names as input. Instead, all variables are part of the plot window. Example command

sequences: simulate(A);plotAll(); simulate(A);plotAll(externalWindow=true); simu-
late(A,fileNamePrefix="B");simulate(C);plotAll(x,fileName="B.mat");

```
function plotAll
  input Boolean externalWindow = false "Opens the plot in a new plot window";
  input String fileName = "<default>" "The filename containing the variables. <default> will read
  input String title = "" "This text will be used as the diagram title.";
  input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed, none.";
  input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically sca
  input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scal
  input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";
  input String yLabel = "" "This text will be used as the vertical label in the diagram.";
  input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is visible in the dia
  input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is visible in the diag
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2, DotLine=3, L
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left, right, top, bott
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call callback function e
  output Boolean success "Returns true on success";
end plotAll;
```

15.2.171 plotParametric

Launches a plotParametric window using OMPlot. Returns true on success. Example command sequences: simulate(A);plotParametric(x,y); simulate(A);plotParametric(x,y, externalWindow=true);

```
function plotParametric
  input VariableName xVariable;
  input VariableName yVariable;
  input Boolean externalWindow = false "Opens the plot in a new plot window";
  input String fileName = "<default>" "The filename containing the variables. <default> will read
  input String title = "" "This text will be used as the diagram title.";
  input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed, none.";
  input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically sca
  input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scal
  input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";
  input String yLabel = "" "This text will be used as the vertical label in the diagram.";
  input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is visible in the dia
  input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is visible in the diag
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2, DotLine=3, L
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left, right, top, bott
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call callback function e
  output Boolean success "Returns true on success";
end plotParametric;
```

15.2.172 readFile

The contents of the given file are returned. Note that if the function fails, the error message is returned as a string instead of multiple output or similar.

```
impure function readFile
  input String fileName;
  output String contents;
end readFile;
```

15.2.173 readFileNoNumeric

Returns the contents of the file, with anything resembling a (real) number stripped out, and at the end adding: Filter count from number domain: n. This should probably be changed to multiple outputs; the filtered string and an integer. Does anyone use this API call?

```
function readFileNoNumeric
  input String fileName;
  output String contents;
end readFileNoNumeric;
```

15.2.174 readSimulationResult

Reads a result file, returning a matrix corresponding to the variables and size given.

```
function readSimulationResult
  input String filename;
  input VariableNames variables;
  input Integer size = 0 "0=read any size... If the size is not the same as the result-file, this
  output Real result[:, :];
end readSimulationResult;
```

15.2.175 readSimulationResultSize

The number of intervals that are present in the output file.

```
function readSimulationResultSize
  input String fileName;
  output Integer sz;
end readSimulationResultSize;
```

15.2.176 readSimulationResultVars

Returns the variables in the simulation file; you can use val() and plot() commands using these names.

Takes one simulation results file and returns the variables stored in it.

If readParameters is true, parameter names are returned.

If openmodelicaStyle is true, the stored variable names are converted to the canonical form used by OpenModelica variables (a.der(b) becomes der(a.b), and so on).

```
function readSimulationResultVars
  input String fileName;
  input Boolean readParameters = true;
  input Boolean openmodelicaStyle = false;
  output String[:] vars;
end readSimulationResultVars;
```

15.2.177 realpath

Get full path name of file or directory name

Return the canonicalized absolute pathname. Similar to realpath(3), but with the safety of Modelica strings.

```
function realpath
  input String name "Absolute or relative file or directory name";
  output String fullName "Full path of 'name'";
end realpath;
```

15.2.178 regex

Sets the error buffer and returns -1 if the regex does not compile. The returned result is the same as POSIX regex(): The first value is the complete matched string The rest are the substrings that you wanted. For example: regex(lorem,"([A-Za-z]*) ([A-Za-z]*)",maxMatches=3) => {" ipsum dolor ","ipsum","dolor"} This means if you have n groups, you want maxMatches=n+1

```
function regex
  input String str;
  input String re;
  input Integer maxMatches = 1 "The maximum number of matches that will be returned";
  input Boolean extended = true "Use POSIX extended or regular syntax";
  input Boolean caseInsensitive = false;
  output Integer numMatches "-1 is an error, 0 means no match, else returns a number 1..maxMatches";
  output String matchedSubstrings[maxMatches] "unmatched strings are returned as empty";
end regex;
```

15.2.179 regexBool

Returns true if the string matches the regular expression.

```
function regexBool
  input String str;
  input String re;
  input Boolean extended = true "Use POSIX extended or regular syntax";
  input Boolean caseInsensitive = false;
  output Boolean matches;
end regexBool;
```

15.2.180 regularFileExists

```
function regularFileExists
  input String fileName;
  output Boolean exists;
end regularFileExists;
```

15.2.181 reloadClass

reloads the file associated with the given (loaded class)

Given an existing, loaded class in the compiler, compare the time stamp of the loaded class with the time stamp (mtime) of the file it was loaded from. If these differ, parse the file and merge it with the AST.

```
function reloadClass
  input TypeName name;
  input String encoding = "UTF-8";
  output Boolean success;
end reloadClass;
```

15.2.182 remove

removes a file or directory of given path (which may be either relative or absolute).

```
function remove
  input String path;
  output Boolean success "Returns true on success.";
end remove;
```

15.2.183 removeComponentModifiers

Removes the component modifiers.

```

function removeComponentModifiers
  input TypeName class_;
  input String componentName;
  output Boolean success;
end removeComponentModifiers;

```

15.2.184 removeExtendsModifiers

Removes the extends modifiers of a class.

```

function removeExtendsModifiers
  input TypeName className;
  input TypeName baseClassName;
  output Boolean success;
end removeExtendsModifiers;

```

15.2.185 reopenStandardStream

```

function reopenStandardStream
  input StandardStream _stream;
  input String filename;
  output Boolean success;
end reopenStandardStream;

```

15.2.186 rewriteBlockCall

Function for property modeling, transforms block calls into instantiations for a loaded model

An extension for modeling requirements in Modelica. Rewrites block calls as block instantiations.

```

function rewriteBlockCall
  input TypeName className;
  input TypeName inDefs;
  output Boolean success;
end rewriteBlockCall;

```

15.2.187 runOpenTURNSPythonScript

runs OpenTURNS with the given python script returning the log file

```

function runOpenTURNSPythonScript
  input String pythonScriptFile;
  output String logOutputFile;
end runOpenTURNSPythonScript;

```

15.2.188 runScript

Runs the mos-script specified by the filename.

```

function runScript
  input String fileName "*.mos";
  output String result;
end runScript;

```

15.2.189 runScriptParallel

As *runScript*, but runs the commands in parallel.

If `useThreads=false` (default), the script will be run in an empty environment (same as running a new `omc` process) with default config flags.

If `useThreads=true` (experimental), the scripts will run in parallel in the same address space and with the same environment (which will not be updated).

```
function runScriptParallel
  input String scripts[:];
  input Integer numThreads = numProcessors();
  input Boolean useThreads = false;
  output Boolean results[:];
end runScriptParallel;
```

15.2.190 save

```
function save
  input TypeName className;
  output Boolean success;
end save;
```

15.2.191 saveAll

save the entire loaded AST to file.

```
function saveAll
  input String fileName;
  output Boolean success;
end saveAll;
```

15.2.192 saveModel

```
function saveModel
  input String fileName;
  input TypeName className;
  output Boolean success;
end saveModel;
```

15.2.193 saveTotalModel

```
function saveTotalModel
  input String fileName;
  input TypeName className;
  output Boolean success;
end saveTotalModel;
```

15.2.194 saveTotalSCode

15.2.195 searchClassNames

Searches for the class name in the all the loaded classes. Example command: `searchClassNames("ground");`
`searchClassNames("ground", true);`

Look for searchText in All Loaded Classes and their code. Returns the list of searched classes.

```
function searchClassNames
  input String searchText;
  input Boolean findInText = false;
  output TypeName classNames[:];
end searchClassNames;
```

15.2.196 setAnnotationVersion

Sets the annotation version.

```
function setAnnotationVersion
  input String annotationVersion;
  output Boolean success;
end setAnnotationVersion;
```

15.2.197 setCFlags

CFLAGS

Sets the CFLAGS passed to the C-compiler. Remember to add -fPIC if you are on a 64-bit platform. If you want to see the defaults before you modify this variable, check the output of *getCFlags()*. `#{SIM_OR_DYNLOAD_OPT_LEVEL}` can be used to get a default lower optimization level for dynamically loaded functions. And `#{MODELICAUSERCFLAGS}` is nice to add so you can easily modify the CFLAGS later by using an environment variable.

```
function setCFlags
  input String inString;
  output Boolean success;
end setCFlags;
```

15.2.198 setCXXCompiler

CXX

```
function setCXXCompiler
  input String compiler;
  output Boolean success;
end setCXXCompiler;
```

15.2.199 setCheapMatchingAlgorithm

example input: 3

```
function setCheapMatchingAlgorithm
  input Integer matchingAlgorithm;
  output Boolean success;
end setCheapMatchingAlgorithm;
```

15.2.200 setClassComment

Sets the class comment.

```
function setClassComment
  input TypeName class_;
  input String filename;
  output Boolean success;
end setClassComment;
```

15.2.201 setCommandLineOptions

The input is a regular command-line flag given to OMC, e.g. +d=failtrace or +g=MetaModelica

```
function setCommandLineOptions
  input String option;
  output Boolean success;
end setCommandLineOptions;
```

15.2.202 setCompileCommand

```
function setCompileCommand
  input String compileCommand;
  output Boolean success;
end setCompileCommand;
```

15.2.203 setCompiler

CC

```
function setCompiler
  input String compiler;
  output Boolean success;
end setCompiler;
```

15.2.204 setCompilerFlags

```
function setCompilerFlags
  input String compilerFlags;
  output Boolean success;
end setCompilerFlags;
```

15.2.205 setCompilerPath

```
function setCompilerPath
  input String compilerPath;
  output Boolean success;
end setCompilerPath;
```

15.2.206 setDebugFlags

example input: failtrace,-noevalfunc

```
function setDebugFlags
  input String debugFlags;
  output Boolean success;
end setDebugFlags;
```

15.2.207 setDefaultOpenCLDevice

Sets the default OpenCL device to be used.

```
function setDefaultOpenCLDevice
  input Integer defdevice;
  output Boolean success;
end setDefaultOpenCLDevice;
```

15.2.208 setDocumentationAnnotation

Used to set the Documentation annotation of a class. An empty argument (e.g. for revisions) means no annotation is added.

```
function setDocumentationAnnotation
  input TypeName class_;
  input String info = "";
  input String revisions = "";
  output Boolean bool;
end setDocumentationAnnotation;
```

15.2.209 setEnvironmentVar

```
function setEnvironmentVar
  input String var;
  input String value;
  output Boolean success;
end setEnvironmentVar;
```

15.2.210 setIndexReductionMethod

example input: dynamicStateSelection

```
function setIndexReductionMethod
  input String method;
  output Boolean success;
end setIndexReductionMethod;
```

15.2.211 setInitXmlStartValue

```
function setInitXmlStartValue
  input String fileName;
  input String variableName;
  input String startValue;
  input String outputFile;
  output Boolean success = false;
end setInitXmlStartValue;
```

15.2.212 setInstallationDirectoryPath

Sets the OPENMODELICAHOME environment variable. Use this method instead of setEnvironmentVar.

```
function setInstallationDirectoryPath
  input String installationDirectoryPath;
  output Boolean success;
end setInstallationDirectoryPath;
```

15.2.213 setLanguageStandard

Sets the Modelica Language Standard.

```
function setLanguageStandard
  input String inVersion;
  output Boolean success;
end setLanguageStandard;
```

15.2.214 setLinker

```
function setLinker
  input String linker;
  output Boolean success;
end setLinker;
```

15.2.215 setLinkerFlags

```
function setLinkerFlags
  input String linkerFlags;
  output Boolean success;
end setLinkerFlags;
```

15.2.216 setMatchingAlgorithm

example input: omc

```
function setMatchingAlgorithm
  input String matchingAlgorithm;
  output Boolean success;
end setMatchingAlgorithm;
```

15.2.217 setModelicaPath

The Modelica Library Path - MODELICAPATH in the language specification; OPENMODELICALIBRARY in OpenModelica.

See *loadModel()* for a description of what the MODELICAPATH is used for.

```
function setModelicaPath
  input String modelicaPath;
  output Boolean success;
end setModelicaPath;
```

15.2.218 setNoSimplify

Sets the noSimplify flag.

```
function setNoSimplify
  input Boolean noSimplify;
  output Boolean success;
end setNoSimplify;
```

15.2.219 setOrderConnections

Sets the orderConnection flag.

```
function setOrderConnections
  input Boolean orderConnections;
  output Boolean success;
end setOrderConnections;
```

15.2.220 setPlotCommand

```
function setPlotCommand
  input String plotCommand;
  output Boolean success;
end setPlotCommand;
```

15.2.221 setPostOptModules

example input: lateInline,inlineArrayEqn,removeSimpleEquations.

```
function setPostOptModules
  input String modules;
  output Boolean success;
end setPostOptModules;
```

15.2.222 setPreOptModules

example input: removeFinalParameters,removeSimpleEquations,expandDerOperator

```
function setPreOptModules
  input String modules;
  output Boolean success;
end setPreOptModules;
```

15.2.223 setShowAnnotations

```
function setShowAnnotations
  input Boolean show;
  output Boolean success;
end setShowAnnotations;
```

15.2.224 setSourceFile

```
function setSourceFile
  input TypeName class_;
  input String filename;
  output Boolean success;
end setSourceFile;
```

15.2.225 setTearingMethod

example input: omcTearing

```
function setTearingMethod
  input String tearingMethod;
  output Boolean success;
end setTearingMethod;
```

15.2.226 setTempDirectoryPath

```
function setTempDirectoryPath
  input String tempDirectoryPath;
  output Boolean success;
end setTempDirectoryPath;
```

15.2.227 setVectorizationLimit

```
function setVectorizationLimit
  input Integer vectorizationLimit;
  output Boolean success;
end setVectorizationLimit;
```

15.2.228 simulate

simulates a modelica model by generating c code, build it and run the simulation executable. The only required argument is the className, while all others have some default values. simulate(className, [startTime], [stopTime], [numberOfIntervals], [tolerance], [method], [fileNamePrefix], [options], [outputFormat], [variableFilter], [cflags], [simflags]) Example command: simulate(A);

```
function simulate
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file. <default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> = 1e-6";
  input String method = "<default>" "integration method used for simulation. <default> = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Filter for variables that should store in result file. <def
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String simulationResults;
end simulate;
```

15.2.229 solveLinearSystem

Solve $A \cdot X = B$, using dgesv or lp_solve (if any variable in X is integer) Returns for solver dgesv: info>0: Singular for element i. info<0: Bad input. For solver lp_solve: ???

```
function solveLinearSystem
  input Real[size(B, 1), size(B, 1)] A;
  input Real[:] B;
  input LinearSystemSolver solver = LinearSystemSolver.dgesv;
  input Integer[:] isInt = {-1} "list of indices that are integers";
  output Real[size(B, 1)] X;
  output Integer info;
end solveLinearSystem;
```

15.2.230 sortStrings

Sorts a string array in ascending order.

```
function sortStrings
  input String arr[:];
  output String sorted;
end sortStrings;
```

15.2.231 stringReplace

Replaces all occurrences of the string *source* with *target*.

```
function stringReplace
  input String str;
  input String source;
  input String target;
  output String res;
end stringReplace;
```

15.2.232 stringSplit

Splits the string at the places given by the character

```
function stringSplit
  input String string;
  input String token "single character only";
  output String[:] strings;
end stringSplit;
```

15.2.233 stringTypeName

stringTypeName is used to make it simpler to create some functionality when scripting. The basic use-case is calling functions like simulate when you do not know the name of the class a priori simulate(stringTypeName(readFile("someFile"))).

```
function stringTypeName
  input String str;
  output TypeName cl;
end stringTypeName;
```

15.2.234 stringVariableName

stringVariableName is used to make it simpler to create some functionality when scripting. The basic use-case is calling functions like val when you do not know the name of the variable a priori val(stringVariableName(readFile("someFile"))).

```
function stringVariableName
  input String str;
  output VariableName cl;
end stringVariableName;
```

15.2.235 strtok

Splits the strings at the places given by the token, for example: strtok("abc bdef","b") => {"a","c","def"} strtok("abc bdef","cd") => {"ab","ef"}

```
function strtok
  input String string;
  input String token;
  output String[:] strings;
end strtok;
```

15.2.236 system

Similar to system(3). Executes the given command in the system shell.

```
impure function system
  input String callStr "String to call: sh -c $callStr";
  input String outputFile = "" "The output is redirected to this file (unless already done by callStr)";
  output Integer retval "Return value of the system call; usually 0 on success";
end system;
```

15.2.237 system_parallel

Similar to system(3). Executes the given commands in the system shell, in parallel if omc was compiled using OpenMP.

```
impure function system_parallel
  input String callStr[:] "String to call: sh -c $callStr";
  input Integer numThreads = numProcessors();
  output Integer retval[:] "Return value of the system call; usually 0 on success";
end system_parallel;
```

15.2.238 testsuiteFriendlyName

```
function testsuiteFriendlyName
  input String path;
  output String fixed;
end testsuiteFriendlyName;
```

15.2.239 threadWorkFailed

(Experimental) Exits the current (*worker thread*) signalling a failure.

15.2.240 translateGraphics

```
function translateGraphics
  input TypeName className;
  output String result;
end translateGraphics;
```

15.2.241 translateModelFMU

translates a modelica model into a Functional Mockup Unit. The only required argument is the className, while all others have some default values. Example command: translateModelFMU(className, version="2.0");

```
function translateModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation), me_cs (both model exchange and co-simulation)";
end translateModelFMU;
```

```

input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"className\"";
output String generatedFileName "Returns the full path of the generated FMU.";
end translateModelFMU;

```

15.2.242 typeNameString

```

function typeNameString
input TypeName cl;
output String out;
end typeNameString;

```

15.2.243 typeNameStrings

```

function typeNameStrings
input TypeName cl;
output String out[:];
end typeNameStrings;

```

15.2.244 typeOf

```

function typeOf
input VariableName variableName;
output String result;
end typeOf;

```

15.2.245 uriToFilename

Handles modelica:// and file:// URI's. The result is an absolute path on the local system. modelica:// URI's are only handled if the class is already loaded. Returns the empty string on failure.

```

function uriToFilename
input String uri;
output String filename = "";
output String message = "";
end uriToFilename;

```

15.2.246 val

Return the value of a variable at a given time in the simulation results

Return the value of a variable at a given time in the simulation results.

Works on the filename pointed to by the scripting variable currentSimulationResult or a given filename.

For parameters, any time may be given. For variables the startTime<=time<=stopTime needs to hold.

On error, nan (Not a Number) is returned and the error buffer contains the message.

```

function val
input VariableName var;
input Real time;
input String fileName = "<default>" "The contents of the currentSimulationResult variable";
output Real valAtTime;
end val;

```

15.2.247 verifyCompiler

```
function verifyCompiler
  output Boolean compilerWorks;
end verifyCompiler;
```

15.2.248 writeFile

Write the data to file. Returns true on success.

```
impure function writeFile
  input String fileName;
  input String data;
  input Boolean append = false;
  output Boolean success;
end writeFile;
```

15.3 Examples

The following is an interactive session with the OpenModelica environment including some of the abovementioned commands and examples. First we start the system, and use the command line interface from OMShell, OMNotebook, or command window of some of the other tools.

We type in a very small model:

```
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0+time; y = 6.0;
end Test;
```

We give the command to flatten a model:

```
>>> instantiateModel(Test)
class Test "Testing OpenModelica Scripts"
  Real x;
  Real y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

A range expression is typed in:

```
>>> a:=1:10
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

It is multiplied by 2:

```
>>> a*2
{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

The variables are cleared:

```
>>> clearVariables()
true
```

We print the loaded class test from its internal representation:

```
>>> list(Test)
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

We get the name and other properties of a class:

```
>>> getClassNames()
{Test,ProfilingTest}
>>> getClassComment(Test)
"Testing OpenModelica Scripts"
>>> isPartial(Test)
false
>>> isPackage(Test)
false
>>> isModel(Test)
true
>>> checkModel(Test)
"Check of Test completed successfully.
Class Test has 2 equation(s) and 2 variable(s).
2 of these are trivial equation(s)."
```

The common combination of a simulation followed by getting a value and doing a plot:

```
>>> simulate(Test, stopTime=3.0)
record SimulationResult
  resultFile = "«DOCHOME»/Test_res.mat",
  simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500, tolerance = 1e-10",
  messages = "stdout          | info          | Time measurements are stored in Test_prof.html (human readable)"
  timeFrontend = 0.003949935,
  timeBackend = 0.001843431,
  timeSimCode = 0.047957722,
  timeTemplates = 0.003767149,
  timeCompile = 0.206772698,
  timeSimulation = 0.025803652,
  timeTotal = 0.2901764619999999
end SimulationResult;
>>> val(x , 2.0)
7.0
```

```
>>> plotall()
```

15.3.1 Interactive Function Calls, Reading, and Writing

We enter an assignment of a vector expression, created by the range construction expression 1:12, to be stored in the variable x. The type and the value of the expression is returned.

```
>>> x := 1:12
{1,2,3,4,5,6,7,8,9,10,11,12}
```

The function bubblesort is called to sort this vector in descending order. The sorted result is returned together with its type. Note that the result vector is of type Real[:], instantiated as Real[12], since this is the declared type of the function result. The input Integer vector was automatically converted to a Real vector according to the Modelica type coercion rules.

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/bubblesort.mo")
true
```

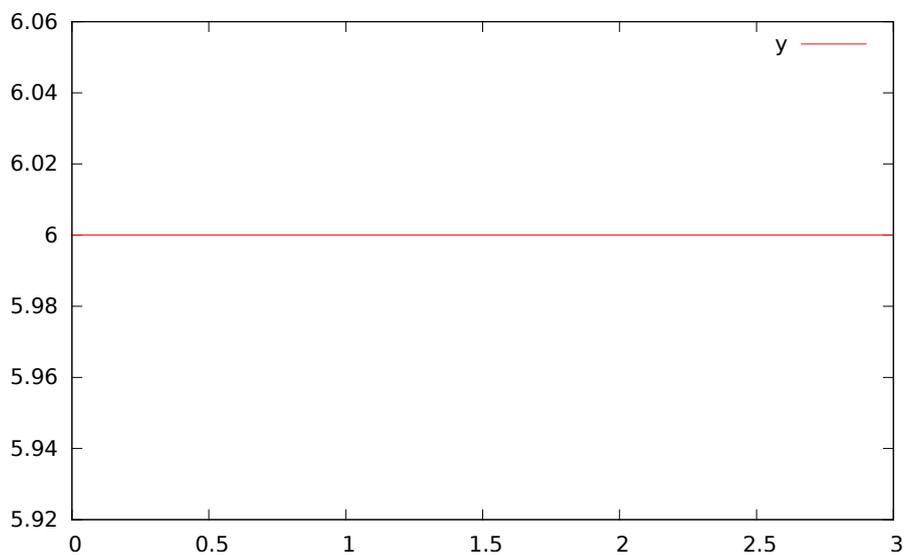


Figure 15.1: Plot generated by OpenModelica+gnuplot

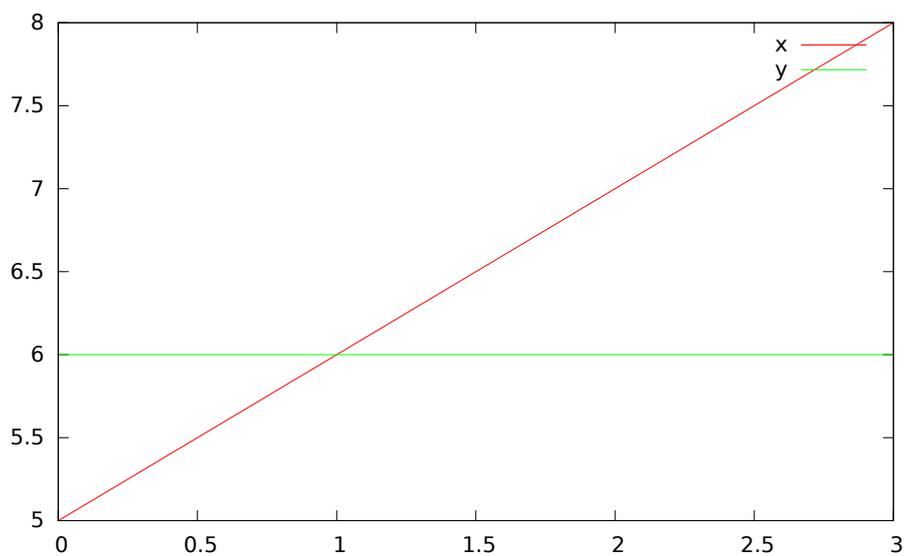


Figure 15.2: Plot generated by OpenModelica+gnuplot

```
>>> bubblesort(x)
{12.0,11.0,10.0,9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0,1.0}
```

Now we want to try another small application, a simplex algorithm for optimization. First read in a small matrix containing coefficients that define a simplex problem to be solved:

```
>>> a := {
  {-1,-1,-1, 0, 0, 0, 0, 0, 0},
  {-1, 1, 0, 1, 0, 0, 0, 0, 5},
  { 1, 4, 0, 0, 1, 0, 0, 0, 45},
  { 2, 1, 0, 0, 0, 1, 0, 0, 27},
  { 3,-4, 0, 0, 0, 0, 1, 0, 24},
  { 0, 0, 1, 0, 0, 0, 0, 1, 4}
}
{{-1,-1,-1,0,0,0,0,0,0}, {-1,1,0,1,0,0,0,0,5}, {1,4,0,0,1,0,0,0,45}, {2,1,0,0,0,1,0,0,27}, {3,-4,0,0,0,0,1,0,24}, {0,0,1,0,0,0,0,1,4}}
```

```
function pivot1
  input Real b[:, :];
  input Integer p;
  input Integer q;
  output Real a[size(b,1), size(b,2)];
protected
  Integer M;
  Integer N;
algorithm
  a := b;
  N := size(a,1)-1;
  M := size(a,2)-1;
  for j in 1:N loop
    for k in 1:M loop
      if j<>p and k<>q then
        a[j,k] := a[j,k]-0.3*j;
      end if;
    end for;
  end for;
  a[p,q] := 0.05;
end pivot1;

function misc_simplex1
  input Real matr[:, :];
  output Real x[size(matr,2)-1];
  output Real z;
  output Integer q;
  output Integer p;
protected
  Real a[size(matr,1), size(matr,2)];
  Integer M;
  Integer N;
algorithm
  N := size(a,1)-1;
  M := size(a,2)-1;
  a := matr;
  p:=0;q:=0;
  a := pivot1(a,p+1,q+1);
  while not (q==(M) or p==(N)) loop
    q := 0;
    while not (q == (M) or a[0+1,q+1]>1) loop
      q:=q+1;
    end while;
    p := 0;
    while not (p == (N) or a[p+1,q+1]>0.1) loop
      p:=p+1;
    end while;
  end while;
end misc_simplex1;
```

```
    if (q < M) and (p < N) and (p>0) and (q>0) then
      a := pivot1(a,p,q);
    end if;
  if (p<=0) and (q<=0) then
    a := pivot1(a,p+1,q+1);
  end if;
  if (p<=0) and (q>0) then
    a := pivot1(a,p+1,q);
  end if;
  if (p>0) and (q<=0) then
    a := pivot1(a,p,q+1);
  end if;
end while;
z := a[1,M];
x := {a[1,i] for i in 1:size(x,1)};
for i in 1:10 loop
  for j in 1:M loop
    x[j] := x[j]+x[j]*0.01;
  end for;
end for;
end misc_simplex1;
```

Then call the simplex algorithm implemented as the Modelica function `simplex1`. This function returns four results, which are represented as a tuple of four return values:

```
>>> misc_simplex1(a)
({0.05523110627056022, -1.104622125411205, -1.104622125411205, 0.0, 0.0, 0.0, 0.0, 0.0}, 0.0, 8, 1)
```

OPENMODELICA COMPILER FLAGS

Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]

- Libraries: Fully qualified names of libraries to load before processing Model or Script. The libraries should be separated by spaces: Lib1 Lib2 ... LibN.

16.1 Options

-d, -debug

Sets debug flags. Use *-help=debug* to see available flags.

String list (default *empty*). *-h, -help*

Displays the help text. Use *-help=topics* for more information.

String (default *empty*). *-v, -version*

Print the version and exit.

Boolean (default *false*). *-target*

Sets the target compiler to use.

String (default *gcc*). Valid options:

- *gcc*
- *msvc*
- *vxworks69*
- *debugrt*

-g, -grammar

Sets the grammar and semantics to accept.

String (default *Modelica*). Valid options:

- *Modelica*
- *MetaModelica*
- *ParModelica*
- *Optimica*

-annotationVersion

Sets the annotation version that should be used.

String (default *3.x*). Valid options:

- *1.x*
- *2.x*

- 3.x

-std

Sets the language standard that should be used.

String (default latest). Valid options:

- 1.x
- 2.x
- 3.1
- 3.2
- 3.3
- latest

*-showErrorMessage*s

Show error messages immediately when they happen.

Boolean (default false). *-showAnnotations*

Show annotations in the flattened code.

Boolean (default false). *-noSimplify*

Do not simplify expressions if set.

Boolean (default false). *-preOptModules*

Sets the pre optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default evaluateReplaceProtectedFinalEvaluateParameters,simplifyIfEquations,expandDerOperator,removeEqualFunction)

Valid options:

- clockPartitioning (Does the clock partitioning.)
- comSubExp (replaces common sub expressions)
- CSE_EachCall (Common Function Call Elimination)
- dumpDAE (dumps the DAE representation of the current transformation state)
- dumpDAEXML (dumps the DAE as xml representation of the current transformation state)
- encapsulateWhenConditions (This module replaces each when condition with a boolean variable.)
- evalFunc (evaluates functions partially)
- evaluateAllParameters (Evaluates all parameters to increase simulation speed.)
- evaluateEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateFinalEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateFinalParameters (Structural parameters and parameters declared as final are evaluated and replaced with their value in other vars. They may no longer be changed in the init file.)
- evaluateReplaceEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceFinalEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceFinalParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceProtectedFinalEvaluateParameters (Structural parameters and parameters declared as final or protected are removed and replaced with their value. They may no longer be changed in the init file.)

- `expandDerOperator` (Expands `der(expr)` using `Derive.differentiateExpTime`.)
- `findStateOrder` (Sets derivative information to states.)
- `inlineArrayEqn` (This module expands all array equations to scalar equations.)
- `inputDerivativesForDynOpt` (Allowed derivatives of inputs in dyn. optimization.)
- `introduceDerAlias` (Adds for every `der`-call an alias equation e.g. $dx = \text{der}(x)$.)
- `removeEqualFunctionCalls` (Detects equal function calls of the form $a=f(b)$ and $c=f(b)$ and substitutes them to get speed up.)
- `removeProtectedParameters` (Replace all parameters with `protected=true` in the system.)
- `removeSimpleEquations` (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- `removeUnusedParameter` (Strips all parameter not present in the equations from the system.)
- `removeUnusedVariables` (Strips all variables not present in the equations from the system.)
- `replaceEdgeChange` (Replace $\text{edge}(b) = b$ and not $\text{pre}(b)$ and $\text{change}(b) = v \Leftrightarrow \text{pre}(v)$.)
- `residualForm` (Transforms simple equations $x=y$ to zero-sum equations $0=y-x$.)
- `resolveLoops` (resolves linear equations in loops)
- `simplifyAllExpressions` (Does simplifications on all expressions.)
- `simplifyIfEquations` (Tries to simplify if equations by use of information from evaluated parameters.)
- `sortEqnsVars` (Heuristic sorting for equations and variables. This module requires `+d=sortEqnsAndVars`.)
- `stateMachineElab` (Does the elaboration of state machines.)
- `unitChecking` (advanced unit checking: 1. calculation of unspecified unit information for variables; 2. unit consistency check for equations)

-cheapmatchingAlgorithm

Sets the cheap matching algorithm to use. A cheap matching algorithm gives a jump start matching by heuristics.

Integer (default 3). Valid options:

- 0 (No cheap matching.)
- 1 (Cheap matching, traverses all equations and match the first free variable.)
- 3 (Random Karp-Sipser: R. M. Karp and M. Sipser. Maximum matching in sparse random graphs.)

-matchingAlgorithm

Sets the matching algorithm to use. See *-help=optmodules* for more info.

String (default PFPlusExt). Valid options:

- BFSB (Breadth First Search based algorithm.)
- DFSB (Depth First Search based algorithm.)
- MC21A (Depth First Search based algorithm with look ahead feature.)
- PF (Depth First Search based algorithm with look ahead feature.)
- PFPlus (Depth First Search based algorithm with look ahead feature and fair row traversal.)
- HK (Combined BFS and DFS algorithm.)
- HKDW (Combined BFS and DFS algorithm.)
- ABMP (Combined BFS and DFS algorithm.)
- PR (Matching algorithm using push relabel mechanism.)

- DFSBExt (Depth First Search based Algorithm external c implementation.)
- BFSBExt (Breadth First Search based Algorithm external c implementation.)
- MC21AExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFPlusExt (Depth First Search based Algorithm with look ahead feature and fair row traversal external c implementation.)
- HKExt (Combined BFS and DFS algorithm external c implementation.)
- HKDWExt (Combined BFS and DFS algorithm external c implementation.)
- ABMPExt (Combined BFS and DFS algorithm external c implementation.)
- PRExt (Matching algorithm using push relabel mechanism external c implementation.)
- BB (BBs try.)

-indexReductionMethod

Sets the index reduction method to use. See *-help=optmodules* for more info.

String (default `dynamicStateSelection`). Valid options:

- `uode` (Use the underlying ODE without the constraints.)
- `dynamicStateSelection` (Simple index reduction method, select (dynamic) dummy states based on analysis of the system.)
- `dummyDerivatives` (Simple index reduction method, select (static) dummy states based on heuristic.)

-postOptModules

Sets the post optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default `lateInlineFunction,inlineArrayEqn,constantLinearSystem,simplifysemiLinear,removeSimpleEquations,simplifyC`)
Valid options:

- `addScaledVars` (added `var_norm = var/nominal`, where `var` is state)
- `addTimeAsState` (Experimental feature: this replaces each occurrence of variable time with a new introduced state `$time` with equation `der($time) = 1.0`)
- `calculateStateSetsJacobians` (Generates analytical Jacobian for dynamic state selection sets.)
- `calculateStrongComponentJacobians` (Generates analytical Jacobian for non-linear strong components.)
- `constantLinearSystem` (Evaluates constant linear systems ($a*x+b*y=c$; $d*x+e*y=f$; a,b,c,d,e,f are constants) at compile-time.)
- `countOperations` (Count the mathematical operations of the system.)
- `CSE` (Common Sub-expression Elimination)
- `detectJacobianSparsePattern` (Detects the sparse pattern for Jacobian A.)
- `dumpComponentsGraphStr` (Dumps the assignment graph used to determine strong components to format suitable for Mathematica)
- `dumpDAE` (dumps the DAE representation of the current transformation state)
- `dumpDAEXML` (dumps the DAE as xml representation of the current transformation state)
- `evaluateEvaluateParameters` (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- `evaluateFinalEvaluateParameters` (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- `evaluateFinalParameters` (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)

- evaluateReplaceEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceFinalEvaluateParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceFinalParameters (Structural parameters and parameters declared as final are removed and replaced with their value. They may no longer be changed in the init file.)
- evaluateReplaceProtectedFinalEvaluateParameters (Structural parameters and parameters declared as final and protected parameters are removed and replaced with their value. They may no longer be changed in the init file.)
- extendDynamicOptimization (Move loops to constraints.)
- generateSymbolicJacobian (Generates symbolic Jacobian matrix, where $\text{der}(x)$ is differentiated w.r.t. x . This matrix can be used to simulate with `dasslColorSymJac`.)
- generateSymbolicLinearization (Generates symbolic linearization matrices A,B,C,D for linear model: $\dot{x} = Ax + Buty = Cx + Du$)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- lateInlineFunction (Perform function inlining for function with annotation `LateInline=true`.)
- partIntornsystem (partitions linear torn systems.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires `+d=reduceDynOpt`.)
- relaxSystem (relaxation from gaussian elimination)
- removeConstants (Remove all constants in the system.)
- removeEqualFunctionCalls (Detects equal function calls of the form $a=f(b)$ and $c=f(b)$ and substitutes them to get speed up.)
- removeSimpleEquations (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- removeUnusedParameter (Strips all parameter not present in the equations from the system to get speed up for compilation of target code.)
- removeUnusedVariables (Strips all variables not present in the equations from the system to get speed up for compilation of target code.)
- reshufflePost (Reshuffles algebraic loops.)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires `+gDynOpt`.)
- simplifyLoops (Simplifies algebraic loops. This modules requires `+simplifyLoops`.)
- simplifyTimeIndepFuncCalls (Simplifies time independent built in function calls like `pre(param) -> param`, `der(param) -> 0.0`, `change(param) -> false`, `edge(param) -> false`.)
- simplifysemiLinear (Simplifies calls to `semiLinear`.)
- solveLinearSystem (solve linear system with newton step)
- solveSimpleEquations (Solves simple equations)

- symEuler (Rewrites the ode system for implicit Euler method. This module requires +symEuler.)
- tearingSystem (For method selection use flag tearingMethod.)

-simCodeTarget

Sets the target language for the code generation.

String (default C). Valid options:

- C
- CSharp
- Cpp
- Adevs
- sfmi
- XML
- Java
- JavaScript
- None

-orderConnections

Orders connect equations alphabetically if set.

Boolean (default true). *-t, -typeinfo*

Prints out extra type information if set.

Boolean (default false). *-a, -keepArrays*

Sets whether to split arrays or not.

Boolean (default false). *-m, -modelicaOutput*

Enables valid modelica output for flat modelica.

Boolean (default false). *-q, -silent*

Turns on silent mode.

Boolean (default false). *-c, -corbaSessionName*

Sets the name of the corba session if *-d=interactiveCorba* is used.

String (default empty). *-n, -numProcs*

Sets the number of processors to use (0=default=auto).

Integer (default 0). *-l, -latency*

Sets the latency for parallel execution.

Integer (default 0). *-b, -bandwidth*

Sets the bandwidth for parallel execution.

Integer (default 0). *-i, -instClass*

Instantiate the class given by the fully qualified path.

String (default empty). *-v, -vectorizationLimit*

Sets the vectorization limit, arrays and matrices larger than this will not be vectorized.

Integer (default 0). *-s, -simulationCg*

Turns on simulation code generation.

Boolean (default false). *-evalAnnotationParams*

Sets whether to evaluate parameters in annotations or not.

Boolean (default `false`). *-generateLabeledSimCode*

Turns on labeled SimCode generation for reduction algorithms.

Boolean (default `false`). *-reduceTerms*

Turns on reducing terms for reduction algorithms.

Boolean (default `false`). *-reductionMethod*

Sets the reduction method to be used.

String (default deletion). Valid options:

- deletion
- substitution
- linearization

-demoMode

Disable Warning/Error Messages.

Boolean (default `false`). *-locale*

Override the locale from the environment.

String (default `empty`). *-o, -defaultOCLDevice*

Sets the default OpenCL device to be used for parallel execution.

Integer (default 0). *-maxTraversals*

Maximal traversals to find simple equations in the acausal system.

Integer (default 2). *-dumpTarget*

Redirect the dump to file. If the file ends with `.html` HTML code is generated.

String (default `empty`). *-delayBreakLoop*

Enables (very) experimental code to break algebraic loops using the `delay()` operator. Probably messes with initialization.

Boolean (default `true`). *-tearingMethod*

Sets the tearing method to use. Select no tearing or choose tearing method.

String (default `cellier`). Valid options:

- `noTearing` (Skip tearing.)
- `omcTearing` (Tearing method developed by TU Dresden: Frenkel, Schubert.)
- `cellier` (Tearing based on Celliers method, revised by FH Bielefeld: Täuber, Patrick)

-tearingHeuristic

Sets the tearing heuristic to use for Cellier-tearing.

String (default `MC3`). Valid options:

- `MC1` (Original cellier with consideration of impossible assignments and discrete Vars.)
- `MC2` (Modified cellier, drop first step.)
- `MC11` (Modified MC1, new last step 'count impossible assignments'.)
- `MC21` (Modified MC2, new last step 'count impossible assignments'.)
- `MC12` (Modified MC1, step 'count impossible assignments' before last step.)
- `MC22` (Modified MC2, step 'count impossible assignments' before last step.)

- MC13 (Modified MC1, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC23 (Modified MC2, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC231 (Modified MC23, Two rounds, choose better potentials-set.)
- MC3 (Modified cellier, build sum of impossible assignment and causalizable equations for all vars, choose var with biggest sum.)
- MC4 (Modified cellier, use all heuristics, choose var that occurs most in potential sets)

-disableLinearTearing

Disables the tearing of linear systems. That might improve the performance of large linear systems(N>1000) in combination with a sparse solver (e.g. umfpack) at runtime (usage with: -ls umfpack).

Boolean (default `false`). *-scalarizeMinMax*

Scalarizes the builtin min/max reduction operators if true.

Boolean (default `false`). *-scalarizeBindings*

Always scalarizes bindings if set.

Boolean (default `false`). *-corbaObjectReferenceFilePath*

Sets the path for corba object reference file if -d=interactiveCorba is used.

String (default `empty`). *-hpcScheduler*

Sets the scheduler for task graph scheduling (list | listr | level | levelfix | ext | mcp | taskdep | tds | bls | rand | none). Default: level.

String (default level). *-hpcCode*

Sets the code-type produced by hpc (openmp | pthreads | pthreads_spin | tbb | mpi). Default: openmp.

String (default openmp). *-rewriteRulesFile*

Activates user given rewrite rules for Absyn expressions. The rules are read from the given file and are of the form `rewrite(fromExp, toExp);`

String (default `empty`). *-replaceHomotopy*

Replaces homotopy(actual, simplified) with the actual expression or the simplified expression. Good for debugging models which use homotopy. The default is to not replace homotopy.

String (default none). Valid options:

- none (Default, do not replace homotopy.)
- actual (Replace homotopy(actual, simplified) with actual.)
- simplified (Replace homotopy(actual, simplified) with simplified.)

-generateSymbolicJacobian

Generates symbolic Jacobian matrix, where $\text{der}(x)$ is differentiated w.r.t. x . This matrix can be utilise by `dassl` with the runtime option: `-dasslJacobian=coloredSymbolicallysymbolical`.

Boolean (default `false`). *-generateSymbolicLinearization*

Generates symbolic linearization matrices A,B,C,D for linear model: $\dot{x} = Ax + Bu$ $y = Cx + Du$

Boolean (default `false`). *-intEnumConversion*

Allow Integer to enumeration conversion.

Boolean (default `false`). *-profiling*

Sets the profiling level to use. Profiled equations and functions record execution time and count for each time step taken by the integrator.

String (default none). Valid options:

- none (Generate code without profiling)
- blocks (Generate code for profiling function calls as well as linear and non-linear systems of equations)
- blocks+html (Like blocks, but also run xsltproc and gnuplot to generate an html report)
- all (Generate code for profiling of all functions and equations)
- all_perf (Generate code for profiling of all functions and equations with additional performance data using the papi-interface (cpp-runtime))
- all_stat (Generate code for profiling of all functions and equations with additional statistics (cpp-runtime))

-reshuffle

sets tolerance of reshuffling algorithm: 1: conservative, 2: more tolerant, 3 resolve all

Integer (default 1). *-gDynOpt*

Generate dynamic optimization problem based on annotation approach.

Boolean (default false). *-maxSizeSolveLinearSystem*

Max size for solveLinearSystem.

Integer (default 0). *-cppFlags*

Sets extra flags for compilation with the C++ compiler (e.g. +cppFlags=-O3,-Wall)

String list (default). *-removeSimpleEquations*

Specifies method that removes simple equations.

String (default default). Valid options:

- none (Disables module)
- default (Performs alias elimination and removes constant variables. Default case uses in preOpt phase the fastAcausal and in postOpt phase the causal implementation.)
- causal (Performs alias elimination and removes constant variables. Causal implementation.)
- fastAcausal (Performs alias elimination and removes constant variables. fastImplementation fastAcausal.)
- allAcausal (Performs alias elimination and removes constant variables. Implementation allAcausal.)
- new (New implementation (experimental))

-dynamicTearing

Activates dynamic tearing (TearingSet can be changed automatically during runtime, strict set vs. casual set.)

Boolean (default false). *-symEuler*

Rewrite the ode system for implicit euler.

Boolean (default false). *-loop2con*

Specifies method that transform loops in constraints. hint: using initial guess from file!

String (default none). Valid options:

- none (Disables module)
- lin (linear loops -> constraints)
- noLin (no linear loops -> constraints)
- all (loops -> constraints)

-forceTearing

Use tearing set even if it is not smaller than the original component.

Boolean (default false). *-simplifyLoops*

Simplify algebraic loops.

Integer (default 0). Valid options:

- 0 (do nothing)
- 1 (special modification of residual expressions)
- 2 (special modification of residual expressions with helper variables)

-recursiveTearing

Inline and repeat tearing.

Integer (default 0). Valid options:

- 0 (do nothing)
- 1 (linear tearing set of size 1)
- 2 (linear tearing)

-flowThreshold

Sets the minimum threshold for stream flow rates

Real (default $1e-07$). *-matrixFormat*

Sets the matrix format type in cpp runtime which should be used (dense | sparse). Default: dense.

String (default dense). *-partIntorn*

Sets the limit for partitionin of linear torn systems.

Integer (default 0). *-initOptModules*

Sets the initialization optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default simplifyComplexFunction,tearingSystem,calculateStrongComponentJacobians,solveSimpleEquations,simplifyAll)
Valid options:

- calculateStrongComponentJacobians (Generates analytical Jacobian for non-linear strong components.)
- constantLinearSystem (Evaluates constant linear systems ($a*x+b*y=c$; $d*x+e*y=f$; a,b,c,d,e,f are constants) at compile-time.)
- extendDynamicOptimization (Move loops to constraints.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires $+d=reduceDynOpt.$)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires $+gDynOpt.$)
- simplifyLoops (Simplifies algebraic loops. This modules requires $+simplifyLoops.$)
- solveSimpleEquations (Solves simple equations)
- tearingSystem (For method selection use flag tearingMethod.)

-maxMixedDeterminedIndex

Sets the maximum mixed-determined index that is handled by the initialization.

Integer (default 3). *-useLocalDirection*

Keeps the input/output prefix for all variables in the flat model, not only top-level ones.

Boolean (default `false`). *-defaultOptModulesOrdering*

If this is activated, then the specified pre-/post-/init-optimization modules will be rearranged to the recommended ordering.

Boolean (default `true`). *-preOptModules+*

Sets additional pre-optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default `empty`). *-preOptModules-*

Disables a list of pre-optimization modules. See *-help=optmodules* for more info.

String list (default `empty`). *-postOptModules+*

Sets additional post-optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default `empty`). *-postOptModules-*

Disables a list of post-optimization modules. See *-help=optmodules* for more info.

String list (default `empty`). *-initOptModules+*

Sets additional init-optimization modules to use in the back end. See *-help=optmodules* for more info.

String list (default `empty`). *-initOptModules-*

Disables a list of init-optimization modules. See *-help=optmodules* for more info.

String list (default `empty`).

16.2 Debug flags

The debug flag takes a comma-separated list of flags which are used by the compiler for debugging or experimental purposes. Flags prefixed with “-” or “no” will be disabled. The available flags are (+ are enabled by default, - are disabled):

***Cache* (default: on)** Turns off the instantiation cache.

***NLSanalyticJacobian* (default: off)** Generates analytical Jacobian for non-linear algebraic loops.

***acceptTooManyFields* (default: off)** Accepts passing records with more fields than expected to a function. This is not allowed, but is used in Fluid.Dissipation. See <https://trac.modelica.org/Modelica/ticket/1245> for details.

***addDerAliases* (default: off)** Adds for every der-call an alias equation e.g. $dx = \text{der}(x)$. It's a work-a-round flag, which helps in some cases to simulate the models e.g. Modelica.Fluid.Examples.HeatExchanger.HeatExchangerSimulation.

***addScaledVars* (default: off)** Adds an alias equation $\text{var_nrom} = \text{var}/\text{nominal}$ where var is state

***addScaledVarsInput* (default: off)** Adds an alias equation $\text{var_nrom} = \text{var}/\text{nominal}$ where var is input

***advanceTearing* (default: off)** Using ExpressionSolve in adjacencyRowEnhanced

***backendaefinfo* (default: off)** Enables dumping of back-end information about system (Number of equations before back-end,...).

***bltdump* (default: off)** Dumps information from index reduction.

***buildExternalLibs* (default: on)** Use the autotools project in the Resources folder of the library to build missing external libraries.

***ceval* (default: off)** Prints extra information from Ceval.

***cgraph* (default: off)** Prints out connection graph information.

***cgraphGraphVizFile* (default: off)** Generates a graphviz file of the connection graph.

***cgraphGraphVizShow* (default: off)** Displays the connection graph with the GraphViz lefty tool.

checkASUB (default: off) Prints out a warning if an ASUB is created from a CREF expression.

checkBackendDae (default: off) Do some simple analyses on the datastructure from the frontend to check if it is consistent.

checkDAECrefType (default: off) Enables extra type checking for cref expressions.

checkSimplify (default: off) Enables checks for expression simplification and prints a notification whenever an undesirable transformation has been performed.

constjac (default: off) solves linear systems with constant Jacobian and variable b-Vector symbolically

countOperations (default: off) Count operations.

daedumpgraphviz (default: off) Dumps the DAE in graphviz format.

debugAlias (default: off) Dump the found alias variables.

debugDifferentiation (default: off) Dumps debug output for the differentiation process.

debugDifferentiationVerbose (default: off) Dumps verbose debug output for the differentiation process.

disableComSubExp (default: off) Deactivates module 'comSubExp'

disablePartitioning (default: off) Deactivates partitioning of entire equation system.

disableRecordConstructorOutput (default: off) Disables output of record constructors in the flat code.

disableSimplifyComplexFunction (default: off) disable simplifyComplexFunction

disableSingleFlowEq (default: off) Disables the generation of single flow equations.

disableStartCalc (default: off) Deactivates the pre-calculation of start values during compile-time.

disableSymbolicLinearization (default: off) For FMI 2.0 only dependency analysis will be perform.

disableWindowsPathCheckWarning (default: off) Disables warnings on Windows if OPENMODELICA-HOME/MinGW is missing.

discreteinfo (default: off) Enables dumping of discrete variables. Extends -d=backendaefinfo.

dummyselect (default: off) Dumps information from dummy state selection heuristic.

dump (default: off) Dumps the absyn representation of a program.

dumpCSE (default: off) Additional output for CSE module.

dumpCSE_verbose (default: off) Additional output for CSE module.

dumpConstrepl (default: off) Dump the found replacements for constants.

dumpEArepl (default: off) Dump the found replacements for evaluate annotations (evaluate=true) parameters.

dumpEncapsulateConditions (default: off) Dumps the results of the preOptModule encapsulateWhenConditions.

dumpEqInUC (default: off) Dumps all equations handled by the unit checker.

dumpEqUCStruct (default: off) Dumps all the equations handled by the unit checker as tree-structure.

dumpFPrepl (default: off) Dump the found replacements for final parameters.

dumpFunctions (default: off) Add functions to backend dumps.

dumpHomotopy (default: off) Dumps the results of the postOptModule optimizeHomotopyCalls.

dumpInlineSolver (default: off) Dumps the inline solver equation system.

dumpPPrepl (default: off) Dump the found replacements for protected parameters.

dumpParamrepl (default: off) Dump the found replacements for remove parameters.

dumpRecursiveTearing (default: off) Dump between steps of recursiveTearing

dumpSCCGraphML (default: off) Dumps graphml files with the strongly connected components.

- dumpSimCode* (default: off) Dumps the simCode model used for code generation.
- dumpSimplifyLoops* (default: off) Dump between steps of simplifyLoops
- dumpSparsePattern* (default: off) Dumps sparse pattern with coloring used for simulation.
- dumpSparsePatternVerbose* (default: off) Dumps in verbose mode sparse pattern with coloring used for simulation.
- dumpSynchronous* (default: off) Dumps information of the clock partitioning.
- dumpTransformedModelica* (default: off) Dumps the back-end DAE to a Modelica-like model after all symbolic transformations are applied.
- dumpUnits* (default: off) Dumps all the calculated units.
- dumpdaelow* (default: off) Dumps the equation system at the beginning of the back end.
- dumpeqinorder* (default: off) Enables dumping of the equations in the order they are calculated.
- dumpindxdae* (default: off) Dumps the equation system after index reduction and optimization.
- dumpinitialsystem* (default: off) Dumps the initial equation system.
- dumprepl* (default: off) Dump the found replacements for simple equation removal.
- dynamicTearingInfo* (default: off) Dumps information about the strict and casual sets of the tearing system.
- dynload* (default: off) Display debug information about dynamic loading of compiled functions.
- evalAllParams* (default: off) Evaluates all parameters in order to increase simulation speed.
- evalConstFuncs* (default: on) Evaluates functions complete and partially and checks for constant output.
- evalFuncDump* (default: off) dumps debug information about the function evaluation
- evalOutputOnly* (default: off) Generates equations to calculate outputs only.
- evalfunc* (default: on) Turns on/off symbolic function evaluation.
- evalparam* (default: off) Constant evaluates parameters if set.
- events* (default: on) Turns on/off events handling.
- execHash* (default: off) Measures the time it takes to hash all simcode variables before code generation.
- execstat* (default: off) Prints out execution statistics for the compiler.
- experimentalReductions* (default: off) Turns on custom reduction functions (OpenModelica extension).
- failtrace* (default: off) Sets whether to print a failtrace or not.
- fmuExperimental* (default: off) Include an extra function in the FMU fmi2GetSpecificDerivatives.
- gcProfiling* (default: off) Prints garbage collection stats to standard output.
- gen* (default: on) Turns on/off dynamic loading of functions that are compiled during translation. Only enable this if external functions are needed to calculate structural parameters or constants.
- gendebugsymbols* (default: off) Generate code with debugging symbols.
- generateCodeCheat* (default: off) Used to generate code for the bootstrapped compiler.
- graphInst* (default: off) Do graph based instantiation.
- graphInstGenGraph* (default: off) Dumps a graph of the program. Use with -d=graphInst
- graphInstRunDep* (default: off) Run scode dependency analysis. Use with -d=graphInst
- graphInstShowGraph* (default: off) Display a graph of the program interactively. Use with -d=graphInst
- graphml* (default: off) Dumps .graphml files for the bipartite graph after Index Reduction and a task graph for the SCCs. Can be displayed with yEd.
- graphviz* (default: off) Dumps the absyn representation of a program in graphviz format.

- graphvizDump* (default: off) Activates additional graphviz dumps (as .dot files). It can be used in addition to one of the following flags: {dumpdaelowdumpinitialsystemsldumpindxdae}.
- hardcodedStartValues* (default: off) Embed the start values of variables and parameters into the c++ code and do not read it from xml file.
- hpcom* (default: off) Enables parallel calculation based on task-graphs.
- hpcomDump* (default: off) Dumps additional information on the parallel execution with hpcom.
- hpcomMemoryOpt* (default: off) Optimize the memory structure regarding the selected scheduler
- implOde* (default: off) activates implicit codegen
- infoXmlOperations* (default: off) Enables output of the operations in the _info.xml file when translating models.
- initialization* (default: off) Shows additional information from the initialization process.
- inlineFunctions* (default: on) Controls if function inlining should be performed.
- inlineSolver* (default: off) Generates code for inline solver.
- instance* (default: off) Prints extra failtrace from InstanceHierarchy.
- interactive* (default: off) Starts omc as a server listening on the socket interface.
- interactiveCorba* (default: off) Starts omc as a server listening on the Corba interface.
- interactivedump* (default: off) Prints out debug information for the interactive server.
- iterationVars* (default: off) Shows a list of all iteration variables.
- lookup* (default: off) Print extra failtrace from lookup.
- modelInfoJson* (default: on) Experimental: Generates a file with suffix _info.json instead of _info.xml.
- onRelaxation* (default: off) Perform O(n) relaxation.
- optdaedump* (default: off) Dumps information from the optimization modules.
- paramdlowdump* (default: off) Enables dumping of the parameters in the order they are calculated.
- parmodauto* (default: off) Experimental: Enable parallelization of independent systems of equations in the translated model.
- patternmAllInfo* (default: off) Adds notifications of all pattern-matching optimizations that are performed.
- patternmDeadCodeElimination* (default: on) Performs dead code elimination in match-expressions.
- patternmMoveLastExp* (default: on) Optimization that moves the last assignment(s) into the result of a match-expression. For example: equation $c = fn(b)$; then c ; \Rightarrow then $fn(b)$;
- patternmSkipFilterUnusedBindings* (default: off)
- pedantic* (default: off) Switch into pedantic debug-mode, to get much more feedback.
- printStructuralParameters* (default: off) Prints the structural parameters identified by the front-end
- pthreads* (default: off) Experimental: Unused parallelization.
- reduceDynOpt* (default: off) remove eqs which not need for the calculations of cost and constraints
- reliidx* (default: off) Prints out debug information about relations, that are used as zero crossings.
- reshufflePost* (default: off) Reshuffles the systems of equations.
- resolveLoops* (default: off) Activates the resolveLoops module.
- rml* (default: off) Converts Modelica-style arrays to lists.
- runtimeStaticLinking* (default: off) Use the static simulation runtime libraries (C++ simulation runtime).
- scodeDep* (default: on) Does scode dependency analysis prior to instantiation. Defaults to true.
- scodeInst* (default: off) Enables experimental SCode instantiation phase.

- semiLinear* (default: off) Enables dumping of the optimization information when optimizing calls to semiLinear.
- shortOutput* (default: off) Enables short output of the simulate() command. Useful for tools like OMNotebook.
- showDaeGeneration* (default: off) Show the dae variable declarations as they happen.
- showEquationSource* (default: off) Display the element source information in the dumped DAE for easier debugging.
- showExpandableInfo* (default: off) Show information about expandable connector handling.
- showInstCacheInfo* (default: off) Prints information about instantiation cache hits and additions. Defaults to false.
- showStartOrigin* (default: off) Enables dumping of the DAE startOrigin attribute of the variables.
- showStatement* (default: off) Shows the statement that is currently being evaluated when evaluating a script.
- sortEqnsAndVars* (default: off) Heuristic sorting for equations and variables. Influenced: removeSimpleEquations and tearing.
- stateselection* (default: off) Enables dumping of selected states. Extends -d=backenddaeinfo.
- static* (default: off) Enables extra debug output from the static elaboration.
- stripPrefix* (default: on) Strips the environment prefix from path/crefs. Defaults to true.
- symjacadump* (default: off) Dumps information about symbolic Jacobians. Can be used only with postOptModules: generateSymbolicJacobian, generateSymbolicLinearization.
- symjacadumpeqn* (default: off) Dump for debug purpose of symbolic Jacobians. (deactivated now).
- symjacadumpverbose* (default: off) Dumps information in verbose mode about symbolic Jacobians. Can be used only with postOptModules: generateSymbolicJacobian, generateSymbolicLinearization.
- symjacwarnings* (default: off) Prints warnings regarding symoblic jacobians.
- tail* (default: off) Prints out a notification if tail recursion optimization has been applied.
- tearingdump* (default: off) Dumps tearing information.
- tearingdumpV* (default: off) Dumps verbose tearing information.
- tplPerfTimes* (default: off) Enables output of template performance data for rendering text to file.
- transformsbeforedump* (default: off) Applies transformations required for code generation before dumping flat code.
- types* (default: off) Prints extra failtrace from Types.
- uncertainties* (default: off) Enables dumping of status when calling modelEquationsUC.
- updmod* (default: off) Prints information about modification updates.
- useMPI* (default: off) Add MPI init and finalize to main method (CPPruntime).
- vectorize* (default: off) Activates vectorization in the backend.
- visxml* (default: off) Outputs a xml-file that contains information for visualization.
- writeToBuffer* (default: off) Enables writing simulation results to buffer.

16.3 Flags for Optimization Modules

Flags that determine which symbolic methods are used to produce the causalized equation system.

The *-preOptModules* flag sets the optimization modules which are used before the matching and index reduction in the back end. These modules are specified as a comma-separated list.

The *-matchingAlgorithm* sets the method that is used for the matching algorithm, after the pre optimization modules.

The *-indexReductionMethod* sets the method that is used for the index reduction, after the pre optimization modules.

The *-initOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for initialization, specified as a comma-separated list.

The *-postOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for simulation, specified as a comma-separated list.

SMALL OVERVIEW OF SIMULATION FLAGS

This chapter contains a *short overview of simulation flags* as well as additional details of the *numerical integration methods*.

17.1 OpenModelica (C-runtime) Simulation Flags

The simulation executable takes the following flags:

-abortSlowSimulation Aborts if the simulation chatters.

-alarm=value or -alarm value Aborts after the given number of seconds (default=0 disables the alarm).

-clock=value or -clock value Selects the type of clock to use. Valid options include:

- RT (monotonic real-time clock)
- CYC (cpu cycles measured with RDTSC)
- CPU (process-based CPU-time)

-cpu Dumps the cpu-time into the result-file using the variable named \$cpuTime

-dasslJacobian=value or -dasslJacobian value Selects the type of the Jacobian that is used for the dassl solver:

- coloredNumerical (colored numerical Jacobian, the default).
- internalNumerical (internal dassl numerical Jacobian).
- coloredSymbolical (colored symbolical Jacobian. Only usable if the simulation is compiled with `-generateSymbolicalJacobian` or `-generateSymbolicalLinearization`).
- numerical - numerical Jacobian.
- symbolical - symbolical Jacobian. Only usable if the simulation is compiled with `-generateSymbolicalJacobian` or `-generateSymbolicalLinearization`.

-dasslnoRootFinding Deactivates the internal root finding procedure of dassl.

-dasslnoRestart Deactivates the restart of dassl after an event is performed.

-emit_protected Emits protected variables to the result-file.

-f=value or -f value Value specifies a new setup XML file to the generated simulation code.

-help=value or -help value Get detailed information that specifies the command-line flag For example, `-help=f` prints detailed information for command-line flag `f`.

-iif=value or -iif value Value specifies an external file for the initialization of the model.

-iim=value or -iim value Value specifies the initialization method.

- none (sets all variables to their start values and skips the initialization process)
- symbolic (solves the initialization problem symbolically - default)

-iit=value or -iit value Value [Real] specifies a time for the initialization of the model.

-ils=value or -ils value Value specifies the number of steps for homotopy method (required: -iim=symbolic) or 'start value homotopy' method (required: -iim=numeric -iom=nelder_mead_ex). The value is an Integer with default value 1.

-initialStepSize=value or -initialStepSize value Value specifies an initial stepsize for the dassl solver.

-exInputFile=value or -exInputFile value Value specifies an external file with inputs for the simulation/optimization of the model.

-stateFile=value or -stateFile value Value specifies an file with states start values for the optimization of the model.

-ipopt_hesse=value or -ipopt_hesse value Value specifies the hessematrix for Ipopt(OMC, BFGS, const).

-ipopt_jac=value or -ipopt_jac value Value specifies the jacobian for Ipopt(SYM, NUM, NUMDENSE).

-ipopt_init=value or -ipopt_init value Value specifies the initial guess for optimization (sim, const).

-ipopt_max_iter=value or -ipopt_max_iter value Value specifies the max number of iteration for ipopt.

-ipopt_warm_start=value or -ipopt_warm_start value Value specifies lvl for a warm start in ipopt: 1,2,3,...

-l=value or -l value Value specifies a time where the linearization of the model should be performed.

-logFormat=value or -logFormat value Value specifies the log format of the executable:

- text (default)
- xml

-ls=value or -ls value Value specifies the linear solver method

- lapack (method using lapack LU factorization)
- lis (method using iterativ solver Lis)
- klu (method using klu sparse linear solver)
- umfpack (method using umfpack sparse linear solver)
- totalpivot (method using a total pivoting LU factorization for underdetermination systems)
- default (default method - lapack with total pivoting as fallback)

-ls_ipopt=value or -ls_ipopt value Value specifies the linear solver method for Ipopt, default mumps. Note: Use if you build ipopt with other linear solver like ma27

-lv=value or -lv value Value (a comma-separated String list) specifies which logging levels to enable. Multiple options can be enabled at the same time.

- LOG_DASSL (additional information about dassl solver)
- LOG_DASSL_STATES (outputs the states at every dassl call)
- LOG_DEBUG (additional debug information)
- LOG_DSS (outputs information about dynamic state selection)
- LOG_DSS_JAC (outputs jacobian of the dynamic state selection)
- LOG_DT (additional information about dynamic tearing)
- LOG_EVENTS (additional information during event iteration)
- LOG_EVENTS_V (verbose logging of event system)
- LOG_INIT (additional information during initialization)
- LOG_IPOPT (information from Ipopt)
- LOG_IPOPT_FULL (more information from Ipopt)
- LOG_IPOPT_JAC (check jacobian matrix with Ipopt)
- LOG_IPOPT_HESSE (check hessian matrix with Ipopt)

- LOG_IPOPT_ERROR (print max error in the optimization)
- LOG_JAC (outputs the jacobian matrix used by dassl)
- LOG_LS (logging for linear systems)
- LOG_LS_V (verbose logging of linear systems)
- LOG-NLS (logging for nonlinear systems)
- LOG-NLS_V (verbose logging of nonlinear systems)
- LOG-NLS_HOMOTOPY (logging of homotopy solver for nonlinear systems)
- LOG-NLS_JAC (outputs the jacobian of nonlinear systems)
- LOG-NLS_JAC_TEST (tests the analytical jacobian of nonlinear systems)
- LOG-NLS_RES (outputs every evaluation of the residual function)
- LOG-NLS_EXTRAPOLATE (outputs debug information about extrapolate process)
- LOG_RES_INIT (outputs residuals of the initialization)
- LOG_SIMULATION (additional information about simulation process)
- LOG_SOLVER (additional information about solver process)
- LOG_SOTI (final solution of the initialization)
- LOG_STATS (additional statistics about timer/events/solver)
- LOG_STATS_V (additional statistics for LOG_STATS)
- LOG_UTIL (???)
- LOG_ZEROCROSSINGS (additional information about the zerocrossings)

-maxStepSize=value or -maxStepSize value Value specifies maximum absolute step size, used by dassl solver.

-maxIntegrationOrder=value or -maxIntegrationOrder value Value specifies maximum integration order, used by dassl solver.

-measureTimePlotFormat=value or -measureTimePlotFormat value Value specifies the output format of the measure time functionality

- svg
- jpg
- ps
- gif
- ...

-newton=value or -newton value Value specifies the damping strategy for the newton solver.

- damped (Newton with a damping strategy)
- damped2 (Newton with a damping strategy 2)
- damped_ls (Newton with a damping line search)
- damped_bt (Newton with a damping backtracking and a minimum search via golden ratio method)
- pure (Newton without damping strategy)

-nls=value or -nls value Value specifies the nonlinear solver:

- hybrid
- kinsol
- newton
- mixed

- hybrid (Modification of the Powell hybrid method from minpack - former default solver)
- kinsol (sundials/kinsol - prototype implementation)
- newton (Newton Raphson - prototype implementation)
- homotopy (Damped Newton solver if failing case fixed-point and Newton homotopies are tried.)
- mixed (Mixed strategy. First the homotopy solver is tried and then as fallback the hybrid solver.)

-nlsInfo Outputs detailed information about solving process of non-linear systems into csv files.

-noemit Do not emit any results to the result file.

-noEquidistantTimeGrid Output the internal steps given by dassl instead of interpolating results into an equidistant time grid as given by stepSize or numberOfIntervals.

-noEquidistantOutputFrequency=value or -noEquidistantOutputFrequency value Integer value n controls the output frequency in noEquidistantTimeGrid mode and outputs every n-th time step

-noEquidistantOutputTime=value or -noEquidistantOutputTime value Real value timeValue controls the output time point in noEquidistantOutputTime mode and outputs every $\text{time} \geq k * \text{timeValue}$, where k is an integer

-noEventEmit Do not emit event points to the result file.

-optDebugJac=value or -optDebugJac value Value specifies the number of iterations from the dynamic optimization, which will be debugged, creating .csv and .py files.

-output=value or -output value Output the variables a, b and c at the end of the simulation to the standard output: time = value, a = value, b = value, c = value

-override=value or -override value Override the variables or the simulation settings in the XML setup file For example: var1=start1,var2=start2,par3=start3,startTime=val1,stopTime=val2

-overrideFile=value or -overrideFile value Will override the variables or the simulation settings in the XML setup file with the values from the file. Note that: -overrideFile CANNOT be used with -override. Use when variables for -override are too many. overrideFileName contains lines of the form: var1=start1

-optimizerNP=value or -optimizerNP value Value specifies the number of points in a subinterval. Currently supports numbers 1 and 3.

-optimizerTimeGrid=value or -optimizerTimeGrid value Value specifies external file with time points.

-keepHessian=value or -keepHessian value Value specifies the number of steps, which keep hessian matrix constant.

-port=value or -port value Value specifies the port for simulation status (default disabled).

-r=value or -r value Value specifies the name of the output result file. The default file-name is based on the model name and output format. For example: Model_res.mat.

-s=value or -s value Value specifies the solver (integration method).

- euler
- rungekutta
- dassl with colored numerical jacobian, with interval root finding - default
- optimization
- radau5 [sundial/kinsol needed]
- radau3 [sundial/kinsol needed]
- radau1 [sundial/kinsol needed]
- lobatto2 [sundial/kinsol needed]
- lobatto4 [sundial/kinsol needed]
- lobatto6 [sundial/kinsol needed]

- symbolic implicit euler, [compiler flag +symEuler needed]
- symbolic implicit euler with step-size control, [compiler flag +symEuler needed]
- qss

-w Shows all warnings even if a related log-stream is inactive.

17.2 Integration Methods

This section contains additional information about the different integration methods in OpenModelica, selected by the method flag of the *simulate* command or the *-s simflag*.

17.2.1 dassl

Default integration method in OpenModelica. Adams Moulton; the default uses a colored numerical Jacobian and interval root finding. To change settings, use simulation flags such as *dasslJacobian*, *dasslNoRootFinding*, *dasslNoRestart*, *initialStepSize*, *maxStepSize*, *maxIntegrationOrder*, *noEquidistantTimeGrid*.

Order:	1-5
Step Size Control:	true
Order Control:	true
Stability Region:	variable; depend from order

17.2.2 euler

Explicit Euler.

Order:	1
Step Size Control:	false
Order Control:	false
Stability Region:	$ (1,0) \text{ Padé} \leq 1$

17.2.3 rungekutta

Classical Runge-Kutta method.

Order:	4
Step Size Control:	false
Order Control:	false
Stability Region:	$ (4,0) \text{ Padé} \leq 1$

17.2.4 radau1

Radau IIA with one point.

Order:	1
Step Size Control:	false
Order Control:	false
Stability Region:	$ (0,1) \text{ Padé} \leq 1$

17.2.5 radau3

Radau IIA with two points.

Order:	3
Step Size Control:	false
Order Control:	false
Stability Region:	$ (1,2) \text{ Padé} \leq 1$

17.2.6 radau5

Radau IIA with three points.

Order:	5
Step Size Control:	false
Order Control:	false
Stability Region:	$ (2,3) \text{ Padé} \leq 1$

17.2.7 lobatto2

Lobatto IIIA with two points.

Order:	2
Step Size Control:	false
Order Control:	false
Stability Region:	$ (2,2) \text{ Padé} \leq 1$

17.2.8 lobatto4

Lobatto IIIA with three points.

Order:	4
Step Size Control:	false
Order Control:	false
Stability Region:	$ (3,3) \text{ Padé} \leq 1$

17.2.9 lobatto6

Lobatto IIIA with four points.

Order:	6
Step Size Control:	false
Order Control:	false
Stability Region:	$ (4,4) \text{ Padé} \leq 1$

17.2.10 Notes

Simulation flags *maxStepSize* and *maxIntegrationOrder* specify maximum absolute step size and maximum integration order used by the dassl solver.

General step size without control $\approx \frac{\text{stopTime} - \text{startTime}}{\text{numberOfIntervals}}$. Events change the step size (see [Modelica spec 3.3 p. 88](#)).

For (a,b) Padé see [wikipedia](#).

FREQUENTLY ASKED QUESTIONS (FAQ)

Below are some frequently asked questions in three areas, with associated answers.

18.1 OpenModelica General

- **Q: OpenModelica does not read the MODELICAPATH environment variable**, even though this is part of the Modelica Language Specification.
- **A: Use the OPENMODELICALIBRARY environment variable instead. We have** temporarily switched to this variable, in order not to interfere with other Modelica tools which might be installed on the same system. In the future, we might switch to a solution with a settings file, that also allows the user to turn on the MODELICAPATH functionality if desired.
- **Q: How do I enter multi-line models into OMShell since it evaluates** when typing the Enter/Return key?
- **A: There are basically three methods: 1) load the model from a file** using the pull-down menu or the loadModel command. 2) Enter the model/function as one (possibly long) line. 3) Type in the model in another editor, where using multiple lines is no problem, and copy/paste the model into OMShell as one operation, then push Enter. Another option is to use OMNotebook instead to enter and evaluate models.

18.2 OMNotebook

- **Q: OMNotebook hangs, what to do?**
- **A: It is probably waiting for the omc.exe (compiler) process. (Under windows):** Kill the processes omc.exe, g++.exe (C-compiler), as.exe (assembler), if present. If OMNotebook then asks whether to restart OMC, answer yes. If not, kill the process OMNotebook.exe and restart manually.
- **Q: After a previous session, when starting OMNotebook again, I get a** strange message.
- **A: You probably quit the previous OpenModelica session in the wrong** way, which left the process omc.exe running. Kill that process, and try starting OMNotebook again.
- **Q: I copy and paste a graphic figure from Word or some other** application into OMNotebook, but the graphic does not appear. What is wrong?
- **A: OMNotebook supports the graphic picture formats supported by Qt 4**, including the .png, .bmp (bitmap) formats, but not for example the gif format. Try to convert your picture into one of the supported formats, (e.g. in Word, first do paste as bitmap format), and then copy the converted version into a text cell in OMNotebook.
- **Q: I select a cell, copy it (e.g. Ctrl-C), and try to paste it at** another place in the notebook. However, this does not work. Instead some other text that I earlier put on the clipboard is pasted into the nearest text cell.

- **A: The problem is wrong choice of cursor mode, which can be text** insertion or cell insertion. If you click inside a cell, the cursor become vertical, and OMNotebook expects you to paste text inside the cell. To paste a cell, you must be in cell insertion mode, i.e., click between two cells (or after a cell), you will get a vertical line. Place the cursor carefully on that vertical line until you see a small horizontal cursor. Then you should past the cell.
- **Q: I am trying to click in cells to place the vertical character** cursor, but it does not seem to react.
- **A: This seems to be a Qt feature. You have probably made a selection** (e.g. for copying) in the output section of an evaluation cell. This seems to block cursor position. Click again in the output section to disable the selection. After that it will work normally.
- **Q: I have copied a text cell and start writing at the beginning of** the cell. Strangely enough, the font becomes much smaller than it should be.
- **A: This seems to be a Qt feature. Keep some of the old text and start** writing the new stuff inside the text, i.e., at least one character position to the right. Afterwards, delete the old text at the beginning of the cell.

18.3 OMDev - OpenModelica Development Environment

- **Q: I get problems compiling and linking some files when using OMDev** with the MINGW (Gnu) C compiler under Windows.
- **A: You probably have some Logitech software installed. There is a** known bug/incompatibility in Logitech products. For example, if lvprcsrv.exe is running, kill it and/or prevent it to start again at reboot; it does not do anything really useful, not needed for operation of web cameras or mice.

MAJOR OPENMODELICA RELEASES

This Appendix lists the most important OpenModelica releases and a brief description of their contents. Right now the versions from 1.3.1 to 1.9.3 are described.

19.1 OpenModelica 1.9.3, September 2015

The most important enhancements in the OpenModelica 1.9.3 release:

- Enhanced collaborative development and testing of OpenModelica by moving to the [GitHub](#) framework for versioning and parallel development.
- More accessible and up-to-date automatically generated documentation provided in both [html](#) and [pdf](#).
- Further improved simulation speed and coverage of several libraries.
- OMEdit graphic connection editor improvements.
- OMNotebook improvements.

19.1.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- Further improved simulation speed and coverage for several libraries.
- Faster generated code for functions involving arrays, factor 2 speedup for many power generation models.
- Better initialization.
- An implicit inline Euler solver available.

An implicit inline Euler solver with adaptive step size control available.

- Code generation to enable vectorization of for-loops.
- Improved non-linear, linear and mixed system solving.
- Cross-compilation for the ARMhf architecture.
- A prototype state machine implementation.
- Improved performance and stability of the C++ runtime option.
- More accessible and up-to-date automatically generated documentation provided in both [html](#) and [.pdf](#).

19.1.2 OpenModelica Notebook (OMNotebook)

Several improvements:

- Support for moving cells from one place to another in a notebook.

- A button for evaluation of whole notebooks.
- A new cell type called Latex cells, supporting Latex formatted input that provides mathematical typesetting of formulae when evaluated.

19.1.3 OpenModelica Shell (OMShell)

No changes.

19.1.4 OpenModelica Eclipse Plug-in (MDT)

No changes apart from bug fixing.

19.1.5 OpenModelica Development Environment (OMDev)

A big change: version handling and parallel development has been improved by moving from SVN to GitHub. This makes it easier for each developer to test his/her fixes and enhancements before committing the code. Automatic mirroring of all code is still performed to the OpenModelica site.

19.1.6 Graphic Editor OMEdit

There are several improvements to the OpenModelica graphic connection editor OMEdit:

- Support for uses annotations.
- Support for declaring components as vectors.
- Faster messages browser with clickable error messages.
- Support for managing the stacking order of graphical shapes.
- Several improvements to the plot tool and text editor in OMEdit.

19.1.7 Optimization

Several improvements of the Dynamic Optimization module with collocation, using Ipopt:

- Better performance due to smart treatment of algebraic loops for optimization.
- Improved formulation of optimization problems with an annotation approach which also allows graphical problem formulation.
- It is now possible to formulate final constraints.

19.1.8 FMI Support

Further improved FMI 2.0 co-simulation support.

19.2 OpenModelica 1.9.2, February 2015

The most important enhancements in the OpenModelica 1.9.2 release:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.

- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Further improved simulation coverage for a number of libraries.
- OMEdit graphic connection editor improvements

19.2.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.
- Further improved simulation coverage for a number of libraries compared to *OpenModelica 1.9.1, October 2014*. For example:
 - MSL 3.2.1 100% compilation, 97% simulation (3% increase)
 - MSL Trunk 99% compilation (1% increase), 93% simulation (3% increase)
 - ModelicaTest 3.2.1 99% compilation (2% increase), 95% simulation (6% increase)
 - ThermoSysPro 100% compilation, 80% simulation (17% increase)
 - ThermoPower 97% compilation (5% increase), 85% simulation (5% increase)
 - Buildings 80% compilation (1% increase), 73% simulation (1% increase)
- Further enhanced OMC compiler front-end coverage, scalability, speed and memory.
- Better initialization.
- Improved tearing.
- Improved non-linear, linear and mixed system solving.
- Common subexpression elimination support - drastically increases performance of some models.

19.2.2 OpenModelica Notebook (OMNotebook)

No changes apart from bug fixing.

19.2.3 OpenModelica Shell (OMShell)

No changes.

19.2.4 OpenModelica Eclipse Plug-in (MDT)

No changes apart from bug fixing.

19.2.5 OpenModelica Development Environment (OMDev)

No changes apart from bug fixing.

19.2.6 Graphic Editor OMEdit

- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Enhanced simulation setup window in OMEdit, which among other things include better support for integration methods and dassl options.
- Support for running multiple simultaneous simulation.
- Improved handling of modifiers.
- Re-simulate with changed options, including history support and re-simulating with previous options possibly edited.
- More user friendly user interface by improved connection line drawing, added snap to grid for icons and conversion of icons from PNG to SVG, and some additional fixes.

19.2.7 Optimization

Some smaller improvements of the Dynamic Optimization module with collocation, using Ipopt.

19.2.8 FMI Support

Further improved for FMI 2.0 model exchange import and export, now compliant according to the FMI compliance tests. FMI 1.0 support has been further improved.

19.3 OpenModelica 1.9.1, October 2014

The most important enhancements in the OpenModelica 1.9.1 release:

- Improved library support.
- Further enhanced OMC compiler front-end coverage and scalability
- Significant improved simulation support for libraries using Fluid and Media.
- Dynamic model debugger for equation-based models integrated with OMEdit.
- Dynamic algorithm model debugger with OMEdit; including support for MetaModelica when using the bootstrapped compiler.

New features: Dynamic debugger for equation-based models; Dynamic Optimization with collocation built into OpenModelica, performance analyzer integrated with the equation model debugger.

19.3.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- Further improved OMC model compiler support for a number of libraries including MSL 3.2.1, ModelicaTest 3.2.1, PetriNet, Buildings, PowerSystems, OpenHydraulics, ThermoPower, and ThermoSysPro.
- Further enhanced OMC compiler front-end coverage, scalability, speed and memory.
- Better coverage of Modelica libraries using Fluid and Media.
- Automatic differentiation of algorithms and functions.
- Improved testing facilities and library coverage reporting.

- Improved model compilation speed by compiling model parts in parallel (bootstrapped compiler).
- Support for running model simulations in a web browser.
- New faster initialization that handles over-determined systems, under-determined systems, or both.
- Compiler back-end partly redesigned for improved scalability and better modularity.
- Better tearing support.
- The first run-time Modelica equation-based model debugger, not available in any other Modelica tool, integrated with OMEdit.
- Enhanced performance profiler integrated with the debugger.
- Improved parallelization prototype with several parallelization strategies, task merging and duplication, shorter critical paths, several scheduling strategies.
- Some support for general solving of mixed systems of equations.
- Better error messages.
- Improved bootstrapped OpenModelica compiler.
- Better handling of array subscripts and dimensions.
- Improved support for reduction functions and operators.
- Better support for partial functions.
- Better support for function tail recursion, which reduces memory usage.
- Partial function evaluation in the back-end to improve solving singular systems.
- Better handling of events/zero crossings.
- Support for colored Jacobians.
- New differentiation package that can handle a much larger number of expressions.
- Support for sparse solvers.
- Better handling of asserts.
- Improved array and matrix support.
- Improved overloaded operators support.
- Improved handling of overconstrained connection graphs.
- Better support for the cardinality operator.
- Parallel compilation of generated code for speeding up compilation.
- Split of model files into several for better compilation scalability.
- Default linear tearing.
- Support for impure functions.
- Better compilation flag documentation.
- Better automatic generation of documentation.
- Better support for calling functions via instance.
- New text template based unparsing for DAE, Absyn, SCode, TaskGraphs, etc.
- Better support for external objects.
- Improved C++ runtime.
- Improved testing facilities.
- New unit checking implementation.
- Support for model rewriting expressions via rewriting rules in an external file.

19.3.2 OpenModelica Notebook (OMNotebook)

No changes apart from bug fixing.

19.3.3 OpenModelica Shell (OMShell)

No changes.

19.3.4 OpenModelica Eclipse Plug-in (MDT)

No changes apart from bug fixing.

19.3.5 OpenModelica Development Environment (OMDev)

No changes apart from bug fixing.

19.3.6 Graphic Editor OMEdit

- Convenient editing of model parameter values and re-simulation without recompilation after parameter changes.
- Improved plotting.
- Better handling of flags/units/resources/crashes.
- Run-time Modelica equation-based model debugger that provides both dynamic run-time debugging and debugging of symbolic transformations.
- Run-time Modelica algorithmic code debugger; also MetaModelica debugger with the bootstrapped Open-Modelica compiler.

19.3.7 Optimization

A builtin integrated Dynamic Optimization module with collocation, using Ipopt, is now available.

19.3.8 FMI Support

Support for FMI 2.0 model exchange import and export has been added. FMI 1.0 support has been further improved.

19.4 OpenModelica 1.9.0, October 2013

The three most important enhancements in the OpenModelica 1.9.0 release:

- OpenModelica compiler support for most of the Fluid library.
- Support for the significantly updated library MSL 3.2.1 final version.
- Significantly enhanced graphical user interface in OMEdit.

New features: integration of the PySimulator analysis package; Dynamic Optimization with CasADi.

19.4.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- A more stable and complete OMC model compiler. The 1.9.0 final version simulates many more models than the previous 1.8.1 version and OpenModelica 1.9.0 beta versions.
- Much better simulation support for MSL 3.2.1, now 270 out of 274 example models compile (98%) and 247 (90%) simulate, compared to 30% simulating in the 1.9.0 beta1 release.
- Much better simulation for the ModelicaTest 3.2.1 library, now 412 out of 428 models compile (96%), and 380 (88%) simulate, compared to 32% in November 2012.
- Improved tearing algorithm for the compiler backend. Tearing is by default used.
- Much faster matching and dynamic state selection algorithms for the compiler backend.
- New index reduction algorithm implementation.
- New default initialization method that symbolically solves the initialization problem much faster and more accurately. This is the first version that in general initialize hybrid models correctly.
- Better class loading from files. The package.order file is now respected and the file structure is more thoroughly examined.
- Basic support for pure/impure functions.
- It is now possible to translate the error messages in the omc kernel.
- Enhanced ModelicaML version with support for value bindings in requirements-driven modeling available for the latest Eclipse and Papyrus versions. GUI specific adaptations. Automated model composition workflows (used for model-based design verification against requirements) are modularized and have improved in terms of performance.
- FMI for co-simulation with OMC as master. Improved FMI import/export, model exchange.
- Checking (when possible) that variables have been assigned to before they are used in algorithmic code.
- Full version of Python scripting.
- 3D graphics visualization using the Modelica3D library.
- The PySimulator package from DLR for additional analysis is integrated with OpenModelica (see Modelica2012 paper), and included in the OpenModelica distribution.
- Prototype support for uncertainty computations, special feature enabled by special flag.
- Parallel algorithmic Modelica support (ParModelica) for efficient portable parallel algorithmic programming based on the OpenCL standard, for CPUs and GPUs.
- Support for optimization of semiLinear according to [semiLinear](#).

19.4.2 OpenModelica Notebook (OMNotebook)

The DrModelica interactive document has been updated and the models tested. Almost all models now simulate with OpenModelica.

19.4.3 OpenModelica Shell (OMShell)

No changes.

19.4.4 OpenModelica Eclipse Plug-in (MDT)

Enhanced debugger for algorithmic Modelica code, supporting both standard Modelica algorithmic code called from simulation models, and MetaModelica code.

19.4.5 OpenModelica Development Environment (OMDev)

Migration of version handling and configuration management from CodeBeamer to Trac.

19.4.6 Graphic Editor OMEdit

- General GUI: backward and forward navigation support in Documentation view, enhanced parameters window with support for Dialog annotation. Most of the images are converted from raster to vector graphics i.e PNG to SVG.
- Libraries Browser: better loading of libraries, library tree can now show protected classes, show library items class names as middle ellipses if the class name text is larger, more options via the right click menu for quick usage.
- ModelWidget: add the partial class as a replaceable component, look for the default component prefixes and name when adding the component.
- GraphicsView: coordinate system manipulation for icon and diagram layers. Show red box for models that do not exist. Show default graphical annotation for the components that doesn't have any graphical annotations. Better resizing of the components. Properties dialog for primitive shapes i.e Line, Polygon, Rectangle, Ellipse, Text and Bitmap.
- File Opening: open one or more Modelica files, allow users to select the encoding while opening the file, convert files to UTF-8 encoding, allow users to open the OpenModelica result files.
- Variables Browser: find variables in the variables browser, sorting in the variables browser.
- Plot Window: clear all curves of the plot window, preserve the old selected variable and update its value with the new simulation result.
- Simulation: support for all the simulation flags, read the simulation output as soon as it is obtained, output window for simulations, options to set matching algorithm and index reduction method for simulation. Display all the files generated during the simulation is now supported. Options to set OMC command line flags.
- Options: options for loading libraries via loadModel and loadFile each time GUI starts, save the last open file directory location, options for setting line wrap mode and syntax highlighting.
- Modelica Text Editor: preserving user customizations, new search & replace functionality, support for comment/uncomment.
- Notifications: show custom dialogs to users allowing them to choose whether they want to see this dialog again or not.
- Model Creation: Better support for creating new classes. Easy creation of extends classes or nested classes.
- Messages Widget: Multi line error messages are now supported.
- Crash Detection: The GUI now automatically detects the crash and writes a stack trace file. The user is given an option to send a crash report along with the stack trace file and few other useful files via email.
- Autosave: OMEdit saves the currently edited model regularly, in order to avoid losing edits after GUI or compiler crash. The save interval can be set in the Options menu.

19.4.7 Optimization

Dynamic optimization with XML export to the CaSAdi package is now integrated with OpenModelica. Moreover, a native integrated Dynamic Optimization prototype using Ipopt is now in the OpenModelica release, but currently needs a special flag to be turned on since it needs more testing and refinement before being generally made available.

19.4.8 FMI Support

FMI co-simulation with OpenModelica as master. Improved FMI Import and export for model exchange. Simulation of multiple instances of the FMU is now possible. Partial support for FMI 2.0 model exchange.

19.5 OpenModelica 1.8.1, March 2012

The OpenModelica 1.8.1 release has a faster and more stable OMC model compiler. It flattens and simulates more models than the previous 1.8.0 version. Significant flattening speedup of the compiler has been achieved for certain large models. It also contains a New ModelicaML version with support for value bindings in requirements-driven modeling and importing Modelica library models into ModelicaML models. A beta version of the new OpenModelica Python scripting is also included.

19.5.1 OpenModelica Compiler (OMC)

This release includes bug fixes and improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to:

- A faster and more stable OMC model compiler. The 1.8.1 version flattens and simulates more models than the previous 1.8.0 version.
- Support for operator overloading (except Complex numbers).
- New ModelicaML version with support for value bindings in requirements-driven modeling and importing Modelica library models into ModelicaML models.
- Faster plotting in OMNotebook. The feature `sendData` has been removed from OpenModelica. As a result, the kernel no longer depends on Qt. The `plot3()` family of functions have now replaced to `plot()`, which in turn have been removed. The non-standard `visualize()` command has been removed in favour of more recent alternatives.
- Store OpenModelica documentation as Modelica Documentation annotations.
- Re-implementation of the simulation runtime using C instead of C++ (this was needed to export FMI source-based packages).
- FMI import/export bug fixes.
- Changed the internal representation of various structures to share more memory. This significantly improved the performance for very large models that use records.
- Faster model flattening, Improved simulation, some graphical API bug fixes.
- More robust and general initialization, but currently time-consuming.
- New initialization flags to `omc` and options to `simulate()`, to control whether fast or robust initialization is selected, or initialization from an external (.mat) data file.
- New options to API calls `list`, `loadFile`, and more.
- Enforce the restriction that input arguments of functions may not be assigned to.
- Improved the scripting environment. `cl := $TypeName(Modelica);getClassComment(cl)`; now works as expected. As does looping over lists of typenames and using reduction expressions.

- Beta version of Python scripting.
- Various bugfixes.
- NOTE: interactive simulation is not operational in this release. It will be put back again in the near future, first available as a nightly build. It is also available in the previous 1.8.0 release.

19.5.2 OpenModelica Notebook (OMNotebook)

Faster and more stable plotting.

19.5.3 OpenModelica Shell (OMShell)

No changes.

19.5.4 OpenModelica Eclipse Plug-in (MDT)

Small fixes and improvements.

19.5.5 OpenModelica Development Environment (OMDev)

No changes.

19.5.6 Graphic Editor OMEdit

Bug fixes.

19.5.7 New OMOptim Optimization Subsystem

Bug fixes.

19.5.8 FMI Support

Bug fixes.

19.6 OpenModelica 1.8, November 2011

The OpenModelica 1.8 release contains OMC flattening improvements for the Media library – it now flattens the whole library and simulates about 20% of its example models. Moreover, about half of the Fluid library models also flatten. This release also includes two new tool functionalities – the FMI for model exchange import and export, and a new efficient Eclipse-based debugger for Modelica/MetaModelica algorithmic code.

19.6.1 OpenModelica Compiler (OMC)

This release includes bug fixes and improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to:

- A faster and more stable OMC model compiler. The 1.8.1 version flattens and simulates more models than the previous 1.7.0 version.
- Flattening of the whole Media library, and about half of the Fluid library. Simulation of approximately 20% of the Media library example models.

- Functional Mockup Interface FMI 1.0 for model exchange, export and import, for the Windows platform.
- Bug fixes in the OpenModelica graphical model connection editor OMEdit, supporting easy-to-use graphical drag-and-drop modeling and MSL 3.1.
- Bug fixes in the OMOptim optimization subsystem.
- Beta version of compiler support for a new Eclipse-based very efficient algorithmic code debugger for functions in MetaModelica/Modelica, available in the development environment when using the bootstrapped OpenModelica compiler.
- Improvements in initialization of simulations.
- Improved index reduction with dynamic state selection, which improves simulation.
- Better error messages from several parts of the compiler, including a new API call for giving better error messages.
- Automatic partitioning of equation systems and multi-core parallel simulation of independent parts based on the shared-memory OpenMP model. This version is a preliminary experimental version without load balancing.

19.6.2 OpenModelica Notebook (OMNotebook)

No changes.

19.6.3 OpenModelica Shell (OMShell)

Small performance improvements.

19.6.4 OpenModelica Eclipse Plug-in (MDT)

Small fixes and improvements. MDT now also includes a beta version of a new Eclipse-based very efficient algorithmic code debugger for functions in MetaModelica/Modelica.

19.6.5 OpenModelica Development Environment (OMDev)

Third party binaries, including Qt libraries and executable Qt clients, are now part of the OMDev package. Also, now uses GCC 4.4.0 instead of the earlier GCC 3.4.5.

19.6.6 Graphic Editor OMEdit

Bug fixes. Access to FMI Import/Export through a pull-down menu. Improved configuration of library loading. A function to go to a specific line number. A button to cancel an on-going simulation. Support for some updated OMC API calls.

19.6.7 New OMOptim Optimization Subsystem

Bug fixes, especially in the Linux version.

19.6.8 FMI Support

The Functional Mockup Interface FMI 1.0 for model exchange import and export is supported by this release. The functionality is accessible via API calls as well as via pull-down menu commands in OMEdit.

19.7 OpenModelica 1.7, April 2011

The OpenModelica 1.7 release contains OMC flattening improvements for the Media library, better and faster event handling and simulation, and fast MetaModelica support in the compiler, enabling it to compile itself. This release also includes two interesting new tools – the OMOptim optimization subsystem, and a new performance profiler for equation-based Modelica models.

19.7.1 OpenModelica Compiler (OMC)

This release includes bug fixes and performance improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to:

- Flattening of the whole Modelica Standard Library 3.1 (MSL 3.1), except Media and Fluid.
- Progress in supporting the Media library, some models now flatten.
- Much faster simulation of many models through more efficient handling of alias variables, binary output format, and faster event handling.
- Faster and more stable simulation through new improved event handling, which is now default.
- Simulation result storage in binary .mat files, and plotting from such files.
- Support for Unicode characters in quoted Modelica identifiers, including Japanese and Chinese.
- Preliminary MetaModelica 2.0 support. (use `setCommandLineOptions({" +g=MetaModelica" })`). Execution is as fast as MetaModelica 1.0, except for garbage collection.
- Preliminary bootstrapped OpenModelica compiler: OMC now compiles itself, and the bootstrapped compiler passes the test suite. A garbage collector is still missing.
- Many bug fixes.

19.7.2 OpenModelica Notebook (OMNotebook)

Improved much faster and more stable 2D plotting through the new OMPlot module. Plotting from binary .mat files. Better integration between OMEdit and OMNotebook, copy/paste between them.

19.7.3 OpenModelica Shell (OMShell)

Same as previously, except the improved 2D plotting through OMPlot.

19.7.4 OpenModelica Eclipse Plug-in (MDT)

Same as previously.

19.7.5 OpenModelica Development Environment (OMDev)

No changes.

19.7.6 Graphic Editor OMEdit

Several enhancements of OMEdit are included in this release. Support for Icon editing is now available. There is also an improved much faster 2D plotting through the new OMPlot module. Better integration between OMEdit and OMNotebook, with copy/paste between them. Interactive on-line simulation is available in an easy-to-use way.

19.7.7 New OMOptim Optimization Subsystem

A new optimization subsystem called OMOptim has been added to OpenModelica. Currently, parameter optimization using genetic algorithms is supported in this version 0.9. Pareto front optimization is also supported.

19.7.8 New Performance Profiler

A new, low overhead, performance profiler for Modelica models has been developed.

19.8 OpenModelica 1.6, November 2010

The OpenModelica 1.6 release primarily contains flattening, simulation, and performance improvements regarding Modelica Standard Library 3.1 support, but also has an interesting new tool – the OMEdit graphic connection editor, and a new educational material called DrControl, and an improved ModelicaML UML/Modelica profile with better support for modeling and requirement handling.

19.8.1 OpenModelica Compiler (OMC)

This release includes bug fix and performance improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and some improvements of the backend, including, but not restricted to:

- Flattening of the whole Modelica Standard Library 3.1 (MSL 3.1), except Media and Fluid.
- Improved flattening speed of a factor of 5-20 compared to OpenModelica 1.5 for a number of models, especially in the MultiBody library.
- Reduced memory consumption by the OpenModelica compiler frontend, for certain large models a reduction of a factor 50.
- Reorganized, more modular OpenModelica compiler backend, can now handle approximately 30 000 equations, compared to previously approximately 10 000 equations.
- Better error messages from the compiler, especially regarding functions.
- Improved simulation coverage of MSL 3.1. Many models that did not simulate before are now simulating. However, there are still many models in certain sublibraries that do not simulate.
- Progress in supporting the Media library, but simulation is not yet possible.
- Improved support for enumerations, both in the frontend and the backend.
- Implementation of stream connectors.
- Support for linearization through symbolic Jacobians.
- Many bug fixes.

19.8.2 OpenModelica Notebook (OMNotebook)

A new DrControl electronic notebook for teaching control and modeling with Modelica.

19.8.3 OpenModelica Shell (OMShell)

Same as previously.

19.8.4 OpenModelica Eclipse Plug-in (MDT)

Same as previously.

19.8.5 OpenModelica Development Environment (OMDev)

Several enhancements. Support for match-expressions in addition to matchcontinue. Support for real if-then-else. Support for if-then without else-branches. Modelica Development Tooling 0.7.7 with small improvements such as more settings, improved error detection in console, etc.

19.8.6 New Graphic Editor OMEdit

A new improved open source graphic model connection editor called OMEdit, supporting 3.1 graphical annotations, which makes it possible to move models back and forth to other tools without problems. The editor has been implemented by students at Linköping University and is based on the C++ Qt library.

19.9 OpenModelica 1.5, July 2010

This OpenModelica 1.5 release has major improvements in the OpenModelica compiler frontend and some in the backend. A major improvement of this release is full flattening support for the MultiBody library as well as limited simulation support for MultiBody. Interesting new facilities are the interactive simulation and the integrated UML-Modelica modeling with ModelicaML. Approximately 4 person-years of additional effort have been invested in the compiler compared to the 1.4.5 version, e.g., in order to have a more complete coverage of Modelica 3.0, mainly focusing on improved flattening in the compiler frontend.

19.9.1 OpenModelica Compiler (OMC)

This release includes major improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and some improvements of the backend, including, but not restricted to:

- Improved flattening speed of at least a factor of 10 or more compared to the 1.4.5 release, primarily for larger models with inner-outer, but also speedup for other models, e.g. the robot model flattens in approximately 2 seconds.
- Flattening of all MultiBody models, including all elementary models, breaking connection graphs, world object, etc. Moreover, simulation is now possible for at least five MultiBody models: Pendulum, DoublePendulum, InitSpringConstant, World, PointGravityWithPointMasses.
- Progress in supporting the Media library, but simulation is not yet possible.
- Support for enumerations, both in the frontend and the backend.
- Support for expandable connectors.
- Support for the inline and late inline annotations in functions.
- Complete support for record constructors, also for records containing other records.
- Full support for iterators, including nested ones.
- Support for inferred iterator and for-loop ranges.
- Support for the function derivative annotation.
- Prototype of interactive simulation.
- Prototype of integrated UML-Modelica modeling and simulation with ModelicaML.
- A new bidirectional external Java interface for calling external Java functions, or for calling Modelica functions from Java.
- Complete implementation of replaceable model extends.
- Fixed problems involving arrays of unknown dimensions.
- Limited support for tearing.

- Improved error handling at division by zero.
- Support for Modelica 3.1 annotations.
- Support for all MetaModelica language constructs inside OpenModelica.
- OpenModelica works also under 64-bit Linux and Mac 64-bit OSX.
- Parallel builds and running test suites in parallel on multi-core platforms.
- New OpenModelica text template language for easier implementation of code generators, XML generators, etc.
- New OpenModelica code generators to C and C# using the text template language.
- Faster simulation result data file output optionally as comma-separated values.
- Many bug fixes.

It is now possible to graphically edit models using parts from the Modelica Standard Library 3.1, since the sim-Forge graphical editor (from Politecnico di Milano) that is used together with OpenModelica has been updated to version 0.9.0 with a important new functionality, including support for Modelica 3.1 and 3.0 annotations. The 1.6 and 2.2.1 Modelica graphical annotation versions are still supported.

19.9.2 OpenModelica Notebook (OMNotebook)

Improvements in platform availability.

- Support for 64-bit Linux.
- Support for Windows 7.
- Better support for MacOS, including 64-bit OSX.

19.9.3 OpenModelica Shell (OMShell)

Same as previously.

19.9.4 OpenModelica Eclipse Plug-in (MDT)

Minor bug fixes.

19.9.5 OpenModelica Development Environment (OMDev)

Minor bug fixes.

19.10 OpenModelica 1.4.5, January 2009

This release has several improvements, especially platform availability, less compiler memory usage, and supporting more aspects of Modelica 3.0.

19.10.1 OpenModelica Compiler (OMC)

This release includes small improvements and some bugfixes of the OpenModelica Compiler (OMC):

- Less memory consumption and better memory management over time. This also includes a better API supporting automatic memory management when calling C functions from within the compiler.
- Modelica 3.0 parsing support.

- Export of DAE to XML and MATLAB.
- Support for several platforms Linux, MacOS, Windows (2000, Xp, Vista).
- Support for record and strings as function arguments.
- Many bug fixes.
- (Not part of OMC): Additional free graphic editor SimForge can be used with OpenModelica.

19.10.2 OpenModelica Notebook (OMNotebook)

A number of improvements, primarily in the plotting functionality and platform availability.

- A number of improvements in the plotting functionality: scalable plots, zooming, logarithmic plots, grids, etc.
- Programmable plotting accessible through a Modelica API.
- Simple 3D visualization.
- Support for several platforms Linux, MacOS, Windows (2000, Xp, Vista).

19.10.3 OpenModelica Shell (OMShell)

Same as previously.

19.10.4 OpenModelica Eclipse Plug-in (MDT)

Minor bug fixes.

19.10.5 OpenModelica Development Environment (OMDev)

Same as previously.

19.11 OpenModelica 1.4.4, Feb 2008

This release is primarily a bug fix release, except for a preliminary version of new plotting functionality available both from the OMNotebook and separately through a Modelica API. This is also the first release under the open source license OSMC-PL (Open Source Modelica Consortium Public License), with support from the recently created Open Source Modelica Consortium. An integrated version handler, bug-, and issue tracker has also been added.

19.11.1 OpenModelica Compiler (OMC)

This release includes small improvements and some bugfixes of the OpenModelica Compiler (OMC):

- Better support for if-equations, also inside when.
- Better support for calling functions in parameter expressions and interactively through dynamic loading of functions.
- Less memory consumption during compilation and interactive evaluation.
- A number of bug-fixes.

19.11.2 OpenModelica Notebook (OMNotebook)

Test release of improvements, primarily in the plotting functionality and platform availability.

- Preliminary version of improvements in the plotting functionality: scalable plots, zooming, logarithmic plots, grids, etc., currently available in a preliminary version through the plot2 function.
- Programmable plotting accessible through a Modelica API.

19.11.3 OpenModelica Shell (OMShell)

Same as previously.

19.11.4 OpenModelica Eclipse Plug-in (MDT)

This release includes minor bugfixes of MDT and the associated MetaModelica debugger.

19.11.5 OpenModelica Development Environment (OMDev)

Extended test suite with a better structure. Version handling, bug tracking, issue tracking, etc. now available under the integrated Codebeamer

19.12 OpenModelica 1.4.3, June 2007

This release has a number of significant improvements of the OMC compiler, OMNotebook, the MDT plugin and the OMDev. Increased platform availability now also for Linux and Macintosh, in addition to Windows. OMShell is the same as previously, but now ported to Linux and Mac.

19.12.1 OpenModelica Compiler (OMC)

This release includes a number of improvements of the OpenModelica Compiler (OMC):

- Significantly increased compilation speed, especially with large models and many packages.
- Now available also for Linux and Macintosh platforms.
- Support for when-equations in algorithm sections, including elsewhen.
- Support for inner/outer prefixes of components (but without type error checking).
- Improved solution of nonlinear systems.
- Added ability to compile generated simulation code using Visual Studio compiler.
- Added “smart setting of fixed attribute to false. If initial equations, OMC instead has fixed=true as default for states due to allowing overdetermined initial equation systems.
- Better state select heuristics.
- New function getIncidenceMatrix(className) for dumping the incidence matrix.
- Builtin functions String(), product(), ndims(), implemented.
- Support for terminate() and assert() in equations.
- In emitted flat form: protected variables are now prefixed with protected when printing flat class.
- Some support for tables, using omcTableTimeIni instead of dymTableTimeIni2.
- Better support for empty arrays, and support for matrix operations like $a*[1,2;3,4]$.

- Improved `val()` function can now evaluate array elements and record fields, e.g. `val(x[n])`, `val(x.y)`.
- Support for `reinit` in algorithm sections.
- String support in external functions.
- Double precision floating point precision now also for interpreted expressions
- Better simulation error messages.
- Support for `der(expressions)`.
- Support for iterator expressions such as `{3*i for i in 1..10}`.
- More test cases in the test suite.
- A number of bug fixes, including sample and event handling bugs.

19.12.2 OpenModelica Notebook (OMNotebook)

A number of improvements, primarily in the platform availability.

- Available on the Linux and Macintosh platforms, in addition to Windows.
- Fixed cell copying bugs, plotting of derivatives now works, etc.

19.12.3 OpenModelica Shell (OMShell)

Now available also on the Macintosh platform.

19.12.4 OpenModelica Eclipse Plug-in (MDT)

This release includes major improvements of MDT and the associated MetaModelica debugger:

- Greatly improved browsing and code completion works both for standard Modelica and for MetaModelica.
- Hovering over identifiers displays type information.
- A new and greatly improved implementation of the debugger for MetaModelica algorithmic code, operational in Eclipse. Greatly improved performance – only approx 10% speed reduction even for 100 000 line programs. Greatly improved single stepping, step over, data structure browsing, etc.
- Many bug fixes.

19.12.5 OpenModelica Development Environment (OMDev)

Increased compilation speed for MetaModelica. Better if-expression support in MetaModelica.

19.13 OpenModelica 1.4.2, October 2006

This release has improvements and bug fixes of the OMC compiler, OMNotebook, the MDT plugin and the OMDev. OMShell is the same as previously.

19.13.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Improved initialization and index reduction.
- Support for integer arrays is now largely implemented.

- The `val(variable,time)` scripting function for accessing the value of a simulation result variable at a certain point in the simulated time.
- **Interactive evaluation of for-loops, while-loops, if-statements**, if-expressions, in the interactive scripting mode.
- Improved documentation and examples of calling the Model Query and Manipulation API.
- Many bug fixes.

19.13.2 OpenModelica Notebook (OMNotebook)

Search and replace functions have been added. The DrModelica tutorial (all files) has been updated, obsolete sections removed, and models which are not supported by the current implementation marked clearly. Automatic recognition of the `.onb` suffix (e.g. when double-clicking) in Windows makes it even more convenient to use.

19.13.3 OpenModelica Eclipse Plug-in (MDT)

Two major improvements are added in this release:

- Browsing and code completion works both for standard Modelica and for MetaModelica.
- The debugger for algorithmic code is now available and operational in Eclipse for debugging of MetaModelica programs.

19.13.4 OpenModelica Development Environment (OMDev)

Mostly the same as previously.

19.14 OpenModelica 1.4.1, June 2006

This release has only improvements and bug fixes of the OMC compiler, the MDT plugin and the OMDev components. The OMSHELL and OMNotebook are the same.

19.14.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Support for external objects.
- OMC now reports the version number (via command line switches or CORBA API `getVersion()`).
- Implemented caching for faster instantiation of large models.
- Many bug fixes.

19.14.2 OpenModelica Eclipse Plug-in (MDT)

Improvements of the error reporting when building the OMC compiler. The errors are now added to the problems view. The latest MDT release is version 0.6.6 (2006-06-06).

19.14.3 OpenModelica Development Environment (OMDev)

Small fixes in the MetaModelica compiler. MetaModelica Users Guide is now part of the OMDev release. The latest OMDev was release in 2006-06-06.

19.15 OpenModelica 1.4.0, May 2006

This release has a number of improvements described below. The most significant change is probably that OMC has now been translated to an extended subset of Modelica (MetaModelica), and that all development of the compiler is now done in this version..

19.15.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Partial support for mixed system of equations.
- New initialization routine, based on optimization (minimizing residuals of initial equations).
- Symbolic simplification of builtin operators for vectors and matrices.
- Improved code generation in simulation code to support e.g. Modelica functions.
- Support for classes extending basic types, e.g. connectors (support for MSL 2.2 block connectors).
- Support for parametric plotting via the plotParametric command.
- Many bug fixes.

19.15.2 OpenModelica Shell (OMShell)

Essentially the same OMShell as in 1.3.1. One difference is that now all error messages are sent to the command window instead of to a separate log window.

19.15.3 OpenModelica Notebook (OMNotebook)

Many significant improvements and bug fixes. This version supports graphic plots within the cells in the notebook. Improved cell handling and Modelica code syntax highlighting. Command completion of the most common OMC commands is now supported. The notebook has been used in several courses.

19.15.4 OpenModelica Eclipse Plug-in (MDT)

This is the first really useful version of MDT. Full browsing of Modelica code, e.g. the MSL 2.2, is now supported. (MetaModelica browsing is not yet fully supported). Full support for automatic indentation of Modelica code, including the MetaModelica extensions. Many bug fixes. The Eclipse plug-in is now in use for OpenModelica development at PELAB and MathCore Engineering AB since approximately one month.

19.15.5 OpenModelica Development Environment (OMDev)

The following mechanisms have been put in place to support OpenModelica development.

- A separate web page for OMDev (OpenModelica Development Environment).
- A pre-packaged OMDev zip-file with precompiled binaries for development under Windows using the mingw Gnu compiler from the Eclipse MDT plug-in. (Development is also possible using Visual Studio).
- All source code of the OpenModelica compiler has recently been translated to an extended subset of Modelica, currently called MetaModelica. The current size of OMC is approximately 100 000 lines. All development is now done in this version.
- A new tutorial and users guide for development in MetaModelica.
- Successful builds and tests of OMC under Linux and Solaris.

19.16 OpenModelica 1.3.1, November 2005

This release has several important highlights.

This is also the *first* release for which the New BSD (Berkeley) open-source license applies to the source code, including the whole compiler and run-time system. This makes it possible to use OpenModelica for both academic and commercial purposes without restrictions.

19.16.1 OpenModelica Compiler (OMC)

This release includes a significantly improved OpenModelica Compiler (OMC):

- Support for hybrid and discrete-event simulation (if-equations, if-expressions, when-equations; not yet if-statements and when-statements).
- Parsing of full Modelica 2.2
- Improved support for external functions.
- Vectorization of function arguments; each-modifiers, better implementation of replaceable, better handling of structural parameters, better support for vector and array operations, and many other improvements.
- Flattening of the Modelica Block library version 1.5 (except a few models), and simulation of most of these.
- Automatic index reduction (present also in previous release).
- Updated User's Guide including examples of hybrid simulation and external functions.

19.16.2 OpenModelica Shell (OMShell)

An improved window-based interactive command shell, now including command completion and better editing and font size support.

19.16.3 OpenModelica Notebook (OMNotebook)

A free implementation of an OpenModelica notebook (OMNotebook), for electronic books with course material, including the DrModelica interactive course material. It is possible to simulate and plot from this notebook.

19.16.4 OpenModelica Eclipse Plug-in (MDT)

An early alpha version of the first Eclipse plug-in (called MDT for Modelica Development Tooling) for Modelica Development. This version gives compilation support and partial support for browsing Modelica package hierarchies and classes.

19.16.5 OpenModelica Development Environment (OMDev)

The following mechanisms have been put in place to support OpenModelica development.

- Bugzilla support for OpenModelica bug tracking, accessible to anybody.
- A system for automatic regression testing of the compiler and simulator, (+ other system parts) usually run at check in time.
- Version handling is done using SVN, which is better than the previously used CVS system. For example, name change of modules is now possible within the version handling system.

CONTRIBUTORS TO OPENMODELICA

This Appendix lists the individuals who have made significant contributions to OpenModelica, in the form of software development, design, documentation, project leadership, tutorial material, promotion, etc. The individuals are listed for each year, from 1998 to the current year: the project leader and main author/editor of this document followed by main contributors followed by contributors in alphabetical order.

20.1 OpenModelica Contributors 2015

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Volker Waurich, TU Dresden, Dresden, Germany.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Robert Braun, IEL, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Bouskela, EDF, Paris, France.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Rüdiger Franke, ABB, Germany

Jens Frenkel, TU Dresden, Dresden, Germany.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.
Pavel Grozman, Equa AB, Stockholm, Sweden.
Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.
Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.
Alf Isaksson, ABB Corporate Research, Västerås, Sweden.
Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.
Tommi Karhela, VTT, Espoo, Finland.
Petter Krus, IEI, Linköping University, Linköping, Sweden.
Juha Kortelainen, VTT, Espoo, Finland.
Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.
Leonardo Laguna, Wolfram MathCore AB, Linköping, Sweden
Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.
Oliver Lenord, Siemens PLM, California, USA.
Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.
Henrik Magnusson, Linköping, Sweden.
Alachew Mengist, PELAB, Linköping University, Linköping, Sweden.
Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.
Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.
Tuomas Miettinen, VTT, Espoo, Finland.
Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.
Kannan Moudgalya, IIT Bombay, Mumbai, India
Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.
Hannu Niemistö, VTT, Espoo, Finland.
Peter Nordin, IEI, Linköping University, Linköping, Sweden.
Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.
Karl Pettersson, IEI, Linköping University, Linköping, Sweden.
Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.
Vitalij Ruge, Fachhochschule Bielefeld, Bielefeld, Germany.
Reino Ruusu, VTT, Espoo, Finland.
Per Sahlin, Equa Simulation AB, Stockholm, Sweden.
Wladimir Schamai, EADS, Hamburg, Germany.
Gerhard Schmitz, University of Hamburg, Hamburg, Germany.
Jan Šilar, Charles University, Prague, Czech Republic
Ingo Staack, IEI, Linköping University, Linköping, Sweden.
Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.
Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.
Bernhard Thiele, PELAB, Linköping University, Linköping, Sweden
Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.
Gustaf Thorslund, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Marcus Walther, TU Dresden, Dresden, Germany

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

20.2 OpenModelica Contributors 2014

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEL, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.
Juha Kortelainen, VTT, Espoo, Finland.
Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.
Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.
Oliver Lenord, Siemens PLM, California, USA.
Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.
Henrik Magnusson, Linköping, Sweden.
Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.
Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.
Tuomas Miettinen, VTT, Espoo, Finland.
Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.
Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.
Hannu Niemistö, VTT, Espoo, Finland.
Peter Nordin, IEI, Linköping University, Linköping, Sweden.
Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.
Karl Pettersson, IEI, Linköping University, Linköping, Sweden.
Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.
Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.
Reino Ruusu, VTT, Espoo, Finland.
Per Sahlin, Equa Simulation AB, Stockholm, Sweden.
Wladimir Schamai, EADS, Hamburg, Germany.
Gerhard Schmitz, University of Hamburg, Hamburg, Germany.
Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.
Anton Sodja, University of Ljubljana, Ljubljana, Slovenia
Ingo Staack, IEI, Linköping University, Linköping, Sweden.
Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.
Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.
Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.
Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.
Parham Vasaiely, EADS, Hamburg, Germany.
Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.
Robert Wotzlaw, Goettingen, Germany.
Azam Zia, PELAB, Linköping University, Linköping, Sweden.

20.3 OpenModelica Contributors 2013

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

20.4 OpenModelica Contributors 2012

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.
David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.
Mikael Axin, IEI, Linköping University, Linköping, Sweden.
Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.
Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.
Robert Braun, IEI, Linköping University, Linköping, Sweden.
David Broman, PELAB, Linköping University, Linköping, Sweden.
Stefan Brus, PELAB, Linköping University, Linköping, Sweden.
Francesco Casella, Politecnico di Milano, Milan, Italy.
Filippo Donida, Politecnico di Milano, Milan, Italy.
Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.
Pavel Grozman, Equa AB, Stockholm, Sweden.
Michael Hanke, NADA, KTH, Stockholm.
Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.
Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.
Alf Isaksson, ABB Corporate Research, Västerås, Sweden.
Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.
Tommi Karhela, VTT, Espoo, Finland.
Petter Krus, IEI, Linköping University, Linköping, Sweden.
Juha Kortelainen, VTT, Espoo, Finland.
Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.
Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.
Oliver Lenord, Siemens PLM, California, USA.
Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.
Henrik Magnusson, Linköping, Sweden.
Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.
Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.
Tuomas Miettinen, VTT, Espoo, Finland.
Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.
Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.
Hannu Niemistö, VTT, Espoo, Finland.
Peter Nordin, IEI, Linköping University, Linköping, Sweden.
Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.
Karl Pettersson, IEI, Linköping University, Linköping, Sweden.
Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.
Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.
Reino Ruusu, VTT, Espoo, Finland.
Per Sahlin, Equa Simulation AB, Stockholm, Sweden.
Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.
Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.
Anton Sodja, University of Ljubljana, Ljubljana, Slovenia
Ingo Staack, IEI, Linköping University, Linköping, Sweden.
Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.
Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.
Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.
Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.
Parham Vasaiely, EADS, Hamburg, Germany.
Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.
Robert Wotzlaw, Goettingen, Germany.
Azam Zia, PELAB, Linköping University, Linköping, Sweden.

20.5 OpenModelica Contributors 2011

Peter Fritzon, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.
Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.
Jens Frenkel, TU Dresden, Dresden, Germany.
Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.
Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.
Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.
David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.
Mikael Axin, IEI, Linköping University, Linköping, Sweden.
Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.
Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.
Robert Braun, IEI, Linköping University, Linköping, Sweden.
David Broman, PELAB, Linköping University, Linköping, Sweden.
Stefan Brus, PELAB, Linköping University, Linköping, Sweden.
Francesco Casella, Politecnico di Milano, Milan, Italy.
Filippo Donida, Politecnico di Milano, Milan, Italy.
Anand Ganeson, PELAB, Linköping University, Linköping, Sweden.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Rickard Lindberg, PELAB, Linköping University, Linköping, Sweden

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

20.6 OpenModelica Contributors 2010

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEL, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Jens Frenkel, TU Dresden, Dresden, Germany.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.
Kim Jansson, PELAB, Linköping University, Linköping, Sweden.
Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.
Tommi Karhela, VTT, Espoo, Finland.
Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.
Petter Krus, IEI, Linköping University, Linköping, Sweden.
Juha Kortelainen, VTT, Espoo, Finland.
Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.
Magnus Leksell, Linköping, Sweden.
Oliver Lenord, Bosch-Rexroth, Lohr am Main, Germany.
Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.
Rickard Lindberg, PELAB, Linköping University, Linköping, Sweden
Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.
Henrik Magnusson, Linköping, Sweden.
Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.
Hannu Niemistö, VTT, Espoo, Finland.
Peter Nordin, IEI, Linköping University, Linköping, Sweden.
Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.
Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.
Atanas Pavlov, Munich, Germany.
Karl Pettersson, IEI, Linköping University, Linköping, Sweden.
Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.
Reino Ruusu, VTT, Espoo, Finland.
Per Sahlin, Equa Simulation AB, Stockholm, Sweden.
Wladimir Schamai, EADS, Hamburg, Germany.
Gerhard Schmitz, University of Hamburg, Hamburg, Germany.
Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.
Anton Sodja, University of Ljubljana, Ljubljana, Slovenia
Ingo Staack, IEI, Linköping University, Linköping, Sweden.
Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.
Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.
Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.
Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.
Robert Wotzlaw, Goettingen, Germany.
Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

20.7 OpenModelica Contributors 2009

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy

Filippo Donida, Politecnico di Milano, Milan, Italy

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Jens Frenkel, TU Dresden, Dresden, Germany.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden

Magnus Leksell, Linköping, Sweden

Oliver Lenord, Bosch-Rexroth, Lohr am Main, Germany

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Hannu Niemistö, VTT, Espoo, Finland

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Atanas Pavlov, Munich, Germany.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany

Robert Wotzlaw, Goettingen, Germany

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden

20.8 OpenModelica Contributors 2008

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Simon Bjorklén, PELAB, Linköping University, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

20.9 OpenModelica Contributors 2007

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Ola Leifler, IDA, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

William Spinelli, Politecnico di Milano, Milano, Italy

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Stefan Vorkoetter, MapleSoft, Waterloo, Canada.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

20.10 OpenModelica Contributors 2006

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Elmir Jagudin, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Andreas Remar, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

20.11 OpenModelica Contributors 2005

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, PELAB, Linköping University and MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Ingemar Axelsson, PELAB, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

20.12 OpenModelica Contributors 2004

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.
Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.
Peter Bunus, PELAB, Linköping University, Linköping, Sweden.
Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.
Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.
Emma Larsdotter Nilsson, PELAB, Linköping University, Linköping, Sweden.
Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.
Adrian Pop, PELAB, Linköping University, Linköping, Sweden.
Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.
Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

20.13 OpenModelica Contributors 2003

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.
Peter Aronsson, Linköping University, Linköping, Sweden.
Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Bunus, PELAB, Linköping University, Linköping, Sweden.
Vadim Engelson, PELAB, Linköping University, Linköping, Sweden.
Daniel Hedberg, Linköping University, Linköping, Sweden.
Eva-Lena Lengquist-Sandelin, PELAB, Linköping University, Linköping, Sweden.
Susanna Monemar, PELAB, Linköping University, Linköping, Sweden.
Adrian Pop, PELAB, Linköping University, Linköping, Sweden.
Erik Svensson, MathCore Engineering AB, Linköping, Sweden.

20.14 OpenModelica Contributors 2002

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.
Daniel Hedberg, Linköping University, Linköping, Sweden.

Henrik Johansson, PELAB, Linköping University, Linköping, Sweden

Andreas Karström, PELAB, Linköping University, Linköping, Sweden

20.15 OpenModelica Contributors 2001

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

20.16 OpenModelica Contributors 2000

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

20.17 OpenModelica Contributors 1999

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden

Peter Rönnquist, PELAB, Linköping University, Linköping, Sweden.

20.18 OpenModelica Contributors 1998

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

David Kågedal, PELAB, Linköping University, Linköping, Sweden.

Vadim Engelson, PELAB, Linköping University, Linköping, Sweden.

BIBLIOGRAPHY

- [Axe05] Ingemar Axelsson. OpenModelica Notebook for interactive structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, oct 2005. LITH-IDA-EX-05/080-SE.
- [Fernstrom06] Anders Fernström. Extending OpenModelica Notebook – an interactive notebook for structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, sep 2006. LITH-IDA-EX-06/057-SE.
- [Fri04] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, feb 2004.
- [Knu84] Donald E. Knuth. Literate programming. *The Computer Journal*, 27:97–111, 1984.
- [Wol96] Stephen Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge University Press, third edition, 1996.
- [BOR+12] Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson. Parallel multiple-shooting and collocation Optimization with OpenModelica. In Martin Otter and Dirk Zimmer, editors, *Proceedings of the 9th International Modelica Conference*. Linköping University Electronic Press, sep 2012. doi:10.3384/ecp12076659.
- [RBB+14] Vitalij Ruge, Willi Braun, Bernhard Bachmann, Andrea Walther, and Kshitij Kulshreshtha. Efficient implementation of collocation methods for optimization using openmodelica and adol-c. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*. Modelica Association and Linköping University Electronic Press, mar 2014. doi:10.3384/ecp140961017.