OpenModelica Users Guide

Version 2014-10-24
for OpenModelica 1.9.1

October 2014

Peter Fritzson
Adrian Pop, Adeel Asghar, Willi Braun, Jens Frenkel,
Lennart Ochel, Martin Sjölund, Per Östlund, Peter Aronsson,
Mikael Axin, Bernhard Bachmann, Vasile Baluta, Robert Braun, Lena Buffoni,
David Broman, Stefan Brus, Francesco Casella, Filippo Donida, Atiyah Elsheikh,
Anand Ganeson, Mahder Gebremedhin, Pavel Grozman, Daniel Hedberg, Michael
Hanke, Alf Isaksson, Kim Jansson, Daniel Kanth, Tommi Karhela, Juha
Kortelainen, Abhinn Kothari, Petter Krus, Alexey Lebedev, Oliver Lenord, Ariel
Liebman, Rickard Lindberg, Håkan Lundvall, Abhir Raj Metkar, Eric Meyers,
Tuomas Miettinen, Afshin Moghadam, Kenneth Nealy, Nemer, Hannu Niemistö,
Peter Nordin, Kristoffer Norling, Arunkumar Palanisamy, Karl Pettersson, Pavol
Privitzer, Jhansi Reddy, Reino Ruusu, Per Sahlin,Wladimir Schamai, Gerhard
Schmitz, Alachew Shitahun, Anton Sodja, Ingo Staack, Kristian Stavåker, Sonia
Tariq, Mohsen Torabzadeh-Tari, Parham Vasaiely, Marcus Walter, Volker
Waurich, Niklas Worschech, Robert Wotzlaw, Azam Zia

Table of Contents

Preface

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

Chapter 1

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:

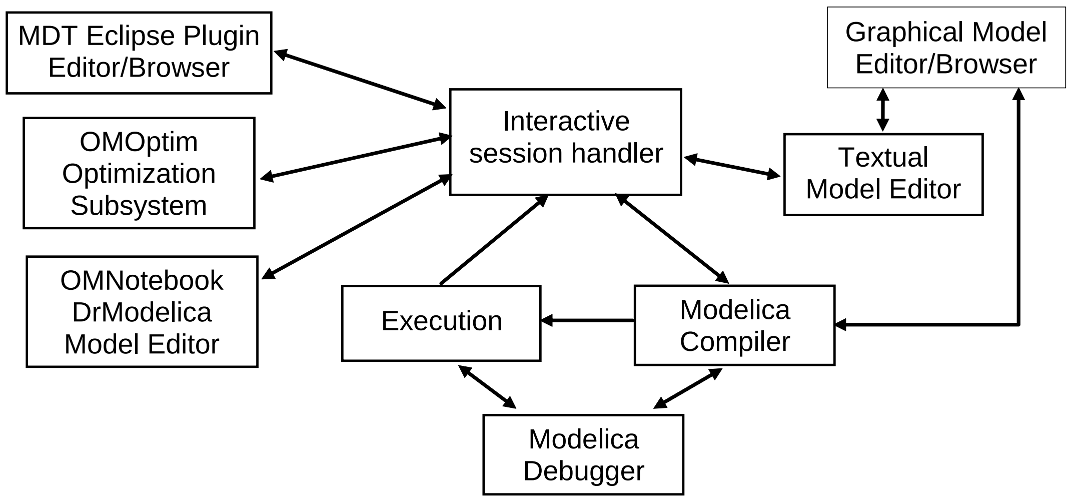
- Development of a *complete formal specification* of Modelica, including both static and dynamic semantics. Such a specification can be used to assist current and future Modelica implementers by providing a semantic reference, as a kind of reference implementation.
- *Language design,* e.g. to further *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 1-1-1 below.


MDT Eclipse Plugin Editor/Browser
OMOptim Optimization Subsystem
OMNotebook DrModelica Model Editor
Interactive session handler
Graphical Model Editor/Browser
Textual Model Editor
Execution
Modelica Compiler
Modelica Debugger

•

Figure 1-1-1. 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.

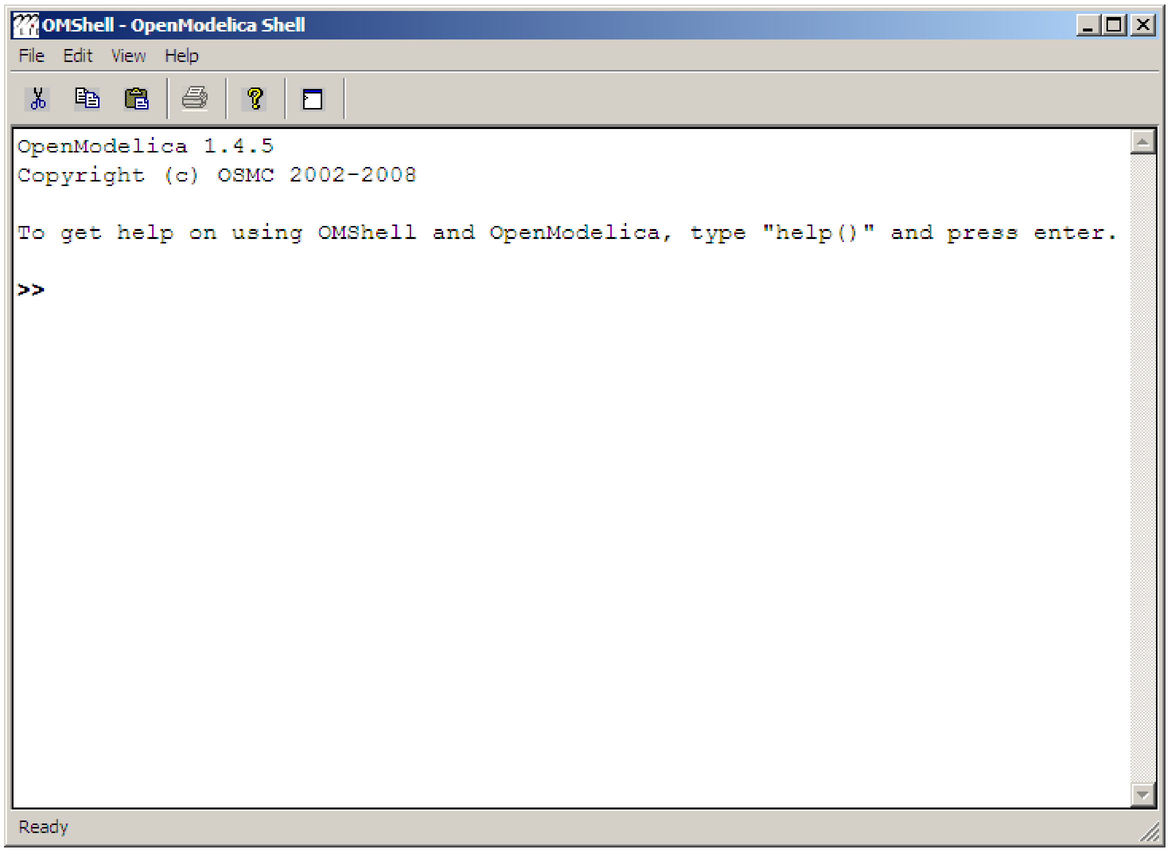
- *Eclipse plugin editor/browser*. The Eclipse plugin called MDT (Modelica Development Tooling) provides file and class hierarchy browsing and text editing capabilities, rather analogous to previously described Emacs editor/browser. Some syntax highlighting facilities are also included. The Eclipse framework 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 OpenModelica notebook `UsersGuideExamples.onb` in the `testmodels` (C:/OpenModelica/share/doc/omc/testmodels/) directory, see also Chapter 4.

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:



```
OpenModelica 1.4.5
Copyright (c) OSMC 2002-2008

To get help on using OMShell and OpenModelica, type "help()" and press enter.

>>
```

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 Compiler Debug Trace Flags in Interactive Mode

When running OMC in interactive mode (for instance using OMShell) one can make use of some of the compiler debug trace flags defined in section 2.1.2 in the System Documentation. Here we give a few example sessions.

Example Session 1

```
OpenModelica 1.9.0
Copyright (c) OSMC 2002-2013
To get help on using OMShell and OpenModelica, type "help()" and press enter.

>> setDebugFlags("failtrace")
true
```

```
>> model A Integer t = 1.5; end A;  //The type is Integer but 1.5 is of Real Type
{A}

>> instantiateModel(A)
"/*- CevalScript.cevalGenerateFunctionDAEs failed( instantiateModel )*/
/*- CevalScript.cevalGenerateFunction failed(instantiateModel)*/
- Inst.makeBinding failed
- Inst.instElement failed: COMPONENT(t in/out: mod: = 1.5 tp: Integer var :VAR,
baseClass: <nothing>)
Scope: A
- Inst.instClassdef failed
class :A
- Inst.instClass: A failed
Inst.instClassInProgram failed
"
Error: Type mismatch in modifier, expected type Integer, got modifier =1.5 of type Real
Error: Error occured while flattening model A
Error: Type mismatch in modifier, expected type Integer, got modifier =1.5 of type Real
Error: Error occured while flattening model A
```

Example Session 2

```
OpenModelica 1.9.1
Copyright (c) OSMC 2002-2014
To get help on using OMShell and OpenModelica, type "help()" and press enter.

>> setDebugFlags("dump")
true
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("setDebugFlags", []),
FUNCTIONARGS(Absyn.STRING("dump"), )))
---/DEBUG(dump)---
"
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("getErrorString", []), FUNCTIONARGS(, )))
---/DEBUG(dump)—

>> model B Integer k = 10; end B;
{B}
---DEBUG(dump)---
Absyn.PROGRAM([
Absyn.CLASS("B", false, false, false, Absyn.R_MODEL,
Absyn.PARTS([Absyn.PUBLIC([Absyn.ELEMENTITEM(Absyn.ELEMENT(false, _, Absyn.UNSPECIFIED
, "component", Absyn.COMPONENTS(Absyn.ATTR(false, false, Absyn.VAR, Absyn.BIDIR,
[]),Integer,[Absyn.COMPONENTITEM(Absyn.COMPONENT("k",[], SOME(Absyn.CLASSMOD([],
SOME(Absyn.INTEGER(10))))), NONE())]), Absyn.INFO("", false, 1, 9, 1, 23)), NONE))])],
NONE()), Absyn.INFO("", false, 1, 1, 1, 30))
],Absyn.TOP)
---/DEBUG(dump)---
"
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("getErrorString", []), FUNCTIONARGS(, )))
---/DEBUG(dump)—

>> instantiateModel(B)
"fclass B
Integer k = 10;
end B;
```

```
"
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("instantiateModel", []),
FUNCTIONARGS(Absyn.CREF(Absyn.CREF_IDENT("B", [])), )))
---/DEBUG(dump)---
"
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("getErrorString", []), FUNCTIONARGS(, )))
---/DEBUG(dump)—

>> simulate(B, startTime=0, stopTime=1, numberOfIntervals=500, tolerance=1e-4)
record SimulationResult
resultFile = "B_res.plt"
end SimulationResult;
---DEBUG(dump)---
#ifdef __cplusplus
extern "C" {
#endif
#ifdef __cplusplus
}
#endif
IEXP(Absyn.CALL(Absyn.CREF_IDENT("simulate", []),
FUNCTIONARGS(Absyn.CREF(Absyn.CREF_IDENT("B", [])), startTime = Absyn.INTEGER(0),
stopTime = Absyn.INTEGER(1), numberOfIntervals = Absyn.INTEGER(500), tolerance =
Absyn.REAL(0.0001))))
---/DEBUG(dump)---
"
---DEBUG(dump)---
IEXP(Absyn.CALL(Absyn.CREF_IDENT("getErrorString", []), FUNCTIONARGS(, )))
---/DEBUG(dump)--
```

Example Session 3

```
OpenModelica 1.9.1
Copyright (c) OSMC 2002-2014
To get help on using OMShell and OpenModelica, type "help()" and press enter.

>> setDebugFlags("failtrace")
true

>> model C Integer a; Real b; equation der(a) = b; der(b) = 12.0; end C;
{C}

>> instantiateModel(C)
"/*- CevalScript.cevalGenerateFunctionDAEs failed( instantiateModel )*/
/*- CevalScript.cevalGenerateFunction failed(instantiateModel)*/
- Static.elabCall failed
function: der posargs: a
- Static.elabExp failed: der(a)
Scope: C
- instEquationCommon failed for eqn: der(a) = b; in scope:C
- instEquation failed eqn:der(a) = b;
- Inst.instClassdef failed
class :C
- Inst.instClass: C failed
Inst.instClassInProgram failed
"
```

```
Error: Illegal derivative. der(a) where a is of type Integer, which is not a subtype of
Real
Error: Wrong type or wrong number of arguments to der(a)'.
Error: Error occured while flattening model C
Error: Illegal derivative. der(a) where a is of type Integer, which is not a subtype of
Real
Error: Wrong type or wrong number of arguments to der(a)'.
Error: Error occured 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("C:/OpenModelica1.9.1/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}
```

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. However, the cat command does not boldface Modelica keywords – this improvement has been done by hand for readability.

```
>> cd("C:/OpenModelica1.9.1/share/doc/omc/testmodels/")
>> system("cat 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;
```

1.2.4 Trying the system and cd Commands

Note: Under Windows the output emitted into stdout by `system` commands is put into the `winmosh` console windows, not into the `winmosh` interaction windows. Thus the text emitted by the above cat command would not be returned. Only a success code (0 = success, 1 = failure) is returned to the `winmosh` window. For example:

```
>> system("dir")
0

>> system("Non-existing command")
1
```

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

```
>> cd()
" C:/OpenModelica1.9.1/share/doc/omc/testmodels/"

>> cd("..")
" C:/OpenModelica1.9.1/share/doc/omc/"

>> cd("C:/OpenModelica1.9.1/share/doc/omc/testmodels/")
" C:/OpenModelica1.9.1/share/doc/omc/testmodels/"
```

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("C:/OpenModelica1.9.1/share/doc/omc/testmodels/dcmotor.mo")
true
```

It is simulated:

```
>> simulate(dcmotor,startTime=0.0,stopTime=10.0)

record
    resultFile = "dcmotor_res.plt"
end record
```

We list the source code of the model:

```
>> list(dcmotor)

"model dcmotor
  Modelica.Electrical.Analog.Basic.Resistor r1(R=10);
  Modelica.Electrical.Analog.Basic.Inductor i1;
  Modelica.Electrical.Analog.Basic.EMF emf1;
  Modelica.Mechanics.Rotational.Inertia load;
  Modelica.Electrical.Analog.Basic.Ground g;
  Modelica.Electrical.Analog.Sources.ConstantVoltage v;

equation
  connect(v.p,r1.p);
  connect(v.n,g.p);
```

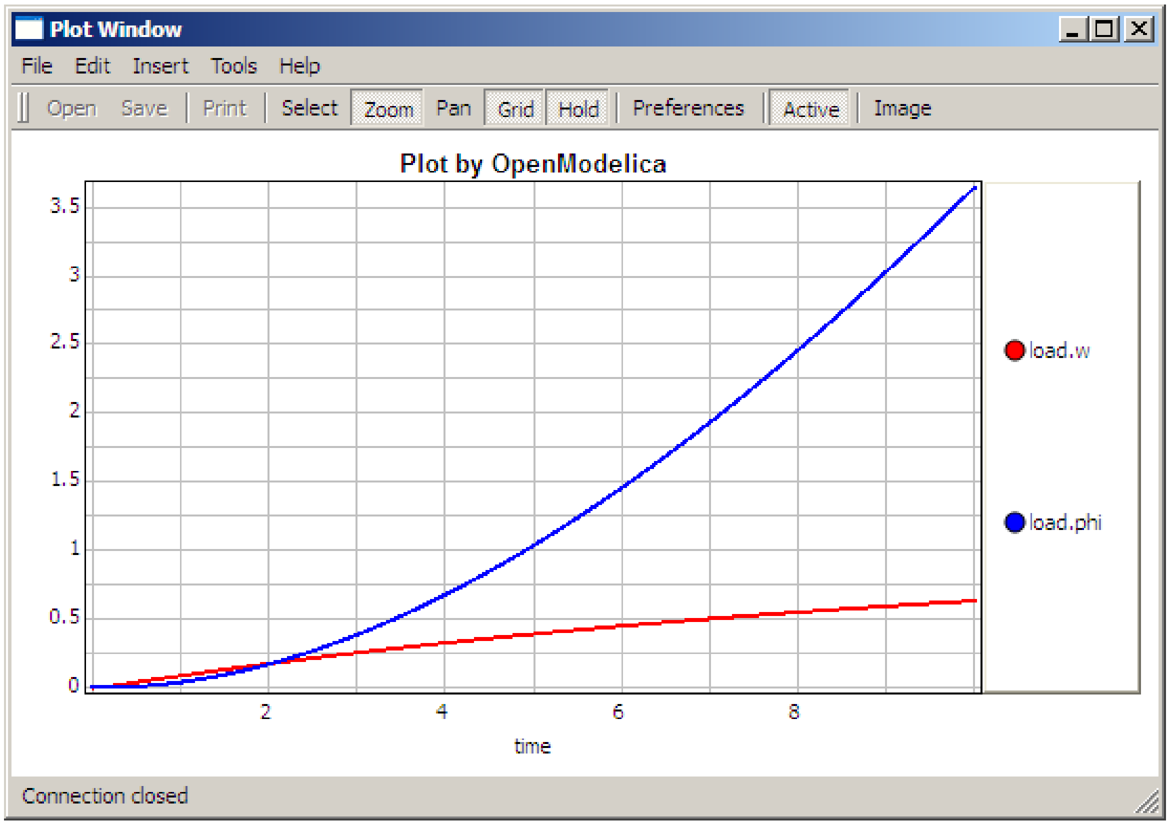
```
    connect(r1.n,i1.p);
    connect(i1.n,emf1.p);
    connect(emf1.n,g.p);
    connect(emf1.flange_b,load.flange_a);
  end dcmotor;
  "
```

We test code instantiation of the model to flat code:

```
>> instantiateModel(dcmotor)

"fclass dcmotor
Real r1.v "Voltage drop between the two pins (= p.v - n.v)";
Real r1.i "Current flowing from pin p to pin n";
Real r1.p.v "Potential at the pin";
Real r1.p.i "Current flowing into the pin";
Real r1.n.v "Potential at the pin";
Real r1.n.i "Current flowing into the pin";
parameter Real r1.R = 10 "Resistance";
Real i1.v "Voltage drop between the two pins (= p.v - n.v)";
Real i1.i "Current flowing from pin p to pin n";
Real i1.p.v "Potential at the pin";
Real i1.p.i "Current flowing into the pin";
Real i1.n.v "Potential at the pin";
Real i1.n.i "Current flowing into the pin";
parameter Real i1.L = 1 "Inductance";
parameter Real emf1.k = 1 "Transformation coefficient";
Real emf1.v "Voltage drop between the two pins";
Real emf1.i "Current flowing from positive to negative pin";
Real emf1.w "Angular velocity of flange_b";
Real emf1.p.v "Potential at the pin";
Real emf1.p.i "Current flowing into the pin";
Real emf1.n.v "Potential at the pin";
Real emf1.n.i "Current flowing into the pin";
Real emf1.flange_b.phi "Absolute rotation angle of flange";
Real emf1.flange_b.tau "Cut torque in the flange";
Real load.phi "Absolute rotation angle of component (= flange_a.phi = flange_b.phi)";
Real load.flange_a.phi "Absolute rotation angle of flange";
Real load.flange_a.tau "Cut torque in the flange";
Real load.flange_b.phi "Absolute rotation angle of flange";
Real load.flange_b.tau "Cut torque in the flange";
parameter Real load.J = 1 "Moment of inertia";
Real load.w "Absolute angular velocity of component";
Real load.a "Absolute angular acceleration of component";
Real g.p.v "Potential at the pin";
Real g.p.i "Current flowing into the pin";
Real v.v "Voltage drop between the two pins (= p.v - n.v)";
Real v.i "Current flowing from pin p to pin n";
Real v.p.v "Potential at the pin";
Real v.p.i "Current flowing into the pin";
Real v.n.v "Potential at the pin";
Real v.n.i "Current flowing into the pin";
parameter Real v.V = 1 "Value of constant voltage";
equation
  r1.R * r1.i = r1.v;
  r1.v = r1.p.v - r1.n.v;
  0.0 = r1.p.i + r1.n.i;
  r1.i = r1.p.i;
  i1.L * der(i1.i) = i1.v;
  i1.v = i1.p.v - i1.n.v;
  0.0 = i1.p.i + i1.n.i;
  i1.i = i1.p.i;
  emf1.v = emf1.p.v - emf1.n.v;
  0.0 = emf1.p.i + emf1.n.i;
  emf1.i = emf1.p.i;
  emf1.w = der(emf1.flange_b.phi);
  emf1.k * emf1.w = emf1.v;
  emf1.flange_b.tau = -(emf1.k * emf1.i);
  load.w = der(load.phi);
  load.a = der(load.w);
  load.J * load.a = load.flange_a.tau + load.flange_b.tau;
  load.flange_a.phi = load.phi;
  load.flange_b.phi = load.phi;
  g.p.v = 0.0;
  v.v = v.V;
  v.v = v.p.v - v.n.v;
  0.0 = v.p.i + v.n.i;
  v.i = v.p.i;
  emf1.flange_b.tau + load.flange_a.tau = 0.0;
  emf1.flange_b.phi = load.flange_a.phi;
  emf1.n.i + v.n.i + g.p.i = 0.0;
  emf1.n.v = v.n.v;
  v.n.v = g.p.v;
  i1.n.i + emf1.p.i = 0.0;
  i1.n.v = emf1.p.v;
  r1.n.i + i1.p.i = 0.0;
  r1.n.v = i1.p.v;
  v.p.i + r1.p.i = 0.0;
  v.p.v = r1.p.v;
  load.flange_b.tau = 0.0;
end dcmotor;
"
```

We plot part of the simulated result:

```
>> plot({load.w,load.phi})
true
```



1.2.6 The val() function

The `val`(*variableName*,*time*) scription 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.

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):

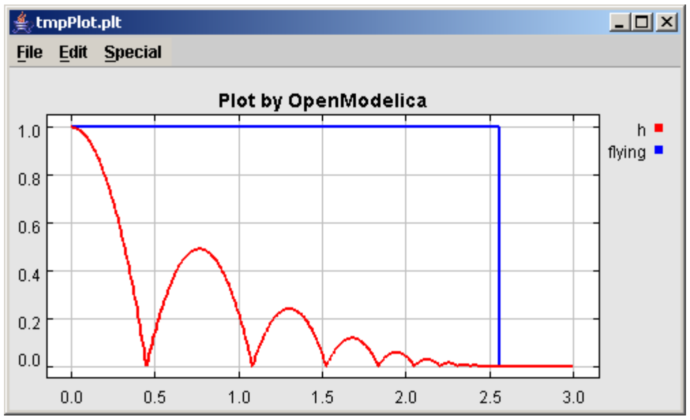
```
>> loadFile("C:/OpenModelica1.9.1/share/doc/omc/testmodels/BouncingBall.mo")
true
>> list(BouncingBall)
"model BouncingBall
  parameter Real e=0.7 "coefficient of restitution";
  parameter Real g=9.81 "gravity acceleration";
  Real h(start=1) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start=true) "true, if ball is flying";
  Boolean impact;
  Real v_new;
equation
  impact=h <= 0.0;
  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:

```
loadFile("BouncingBall.mo");
simulate(BouncingBall, stopTime=3.0);
plot({h,flying});
```

The `runScript` command:

```
>> runScript("sim_BouncingBall.mos")
"true
record
    resultFile = "BouncingBall_res.plt"
end record
true
true"
```



We enter a switch model, to test if-equations (e.g. copy and paste from another file and push enter):

```
>> model Switch
  Real v;
  Real i;
  Real i1;
  Real itot;
  Boolean open;
equation
  itot = i + i1;

  if open then
    v = 0;
  else
    i = 0;
  end if;
   1 - i1 = 0;
   1 - v - i = 0;
   open = time >= 0.5;
end Switch;
```

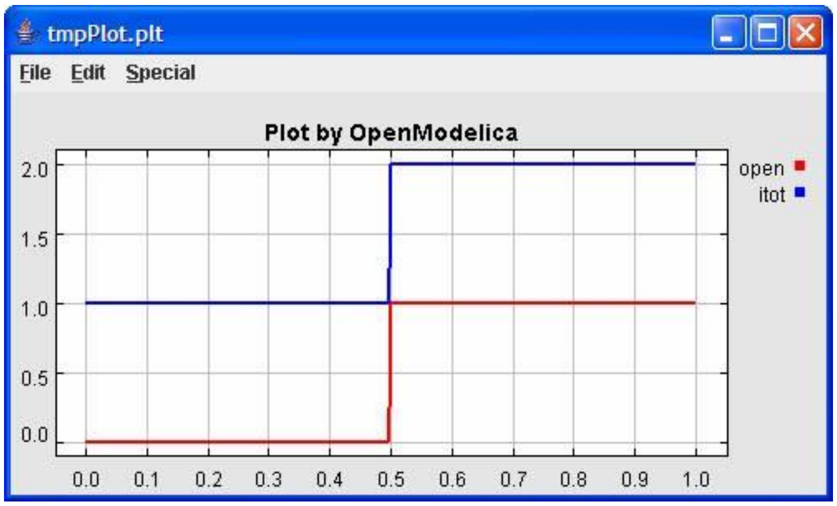
```
Ok
>> simulate(Switch, startTime=0, stopTime=1);
```

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

```
>> val(itot,0)
1
```

Plot `itot` and `open`:

```
>> plot({itot,open})
true
```



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:

```
>> 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`):

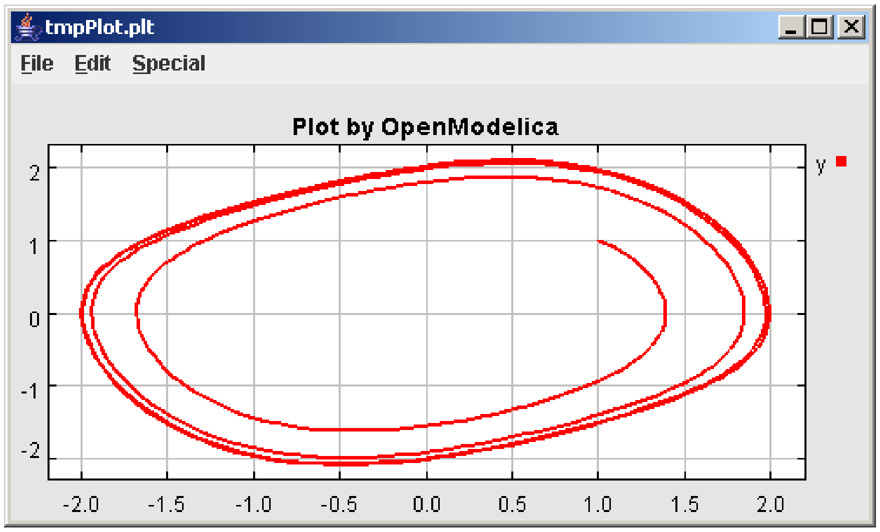
```
>> loadFile("C:/OpenModelica1.9.1/share/doc/omc/testmodels/VanDerPol.mo"))
true
```

It is simulated:

```
>> simulate(VanDerPol)
record
    resultFile = "VanDerPol_res.plt"
end record
```

It is plotted:

```
plotParametric(x,y);
```

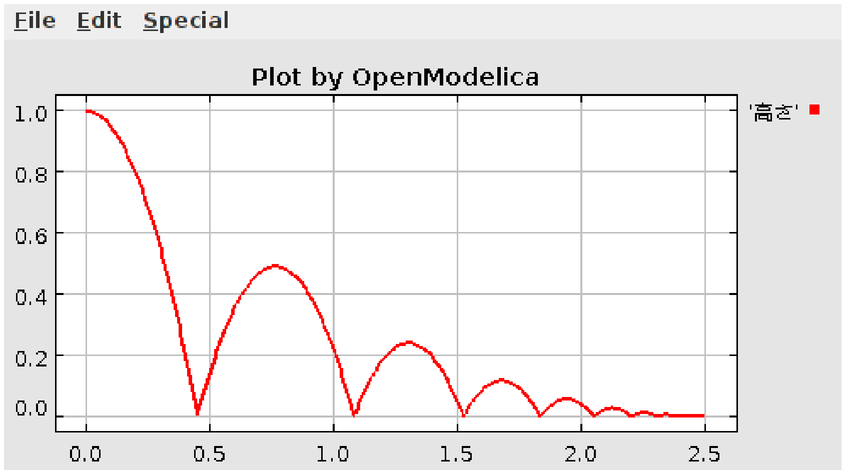


Perform code instantiation to flat form of the `VanDerPol` model:

```
>> instantiateModel(VanDerPol)

"fclass VanDerPol
Real x(start=1.0);
Real y(start=1.0);
parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda * (1.0 - x * x) * y;
end VanDerPol;
"
```

1.2.10 Using Japanese or Chinese Characters

Japenese, 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;
Ok
```

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[]"
```

Retrieve the type of `b`:

```
>> typeOf(b)
"Real"
```

What is the type of `MySqr`? Cannot currently be handled.

```
>> typeOf(MySqr)
Error evaluating expr.
```

List the available variables:

```
>> listVariables()
{currentSimulationResult, a, b}
```

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 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 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 an external scripts or tools like `gnuplot` to generate plots or process data. The `mat` format can be post-processed in MATLAB[1] or Octave[2].

```
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[3]. 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]*\\]")
```

[1] http://www.mathworks.com/products/matlab/
[2] http://www.gnu.org/software/octave/
[3] http://en.wikipedia.org/wiki/Regular_expression

```
// match x or y or z
simulate(... , variableFilter="x|y|z")
```

1.2.15 Using External Functions

See Chapter 13 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 versiosn 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;
```

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("C:/OpenModelica1.9.1/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;
equation
  impact=h <= 0.0;
  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},{"writable",1,1,18,17}}
```

```
>>isFunction(BouncingBall)
false
```

```
>>existClass(BouncingBall)
true
```

```
>>getComponents(BouncingBall)
{{Real,e,"coefficient of restitution", "public", false, false, false,
"parameter", "none", "unspecified"},
{Real,g,"gravity acceleration",
"public", false, false, false, "parameter", "none", "unspecified"},
{Real,h,"height of ball", "public", false, false, false,
"unspecified", "none", "unspecified"},
{Real,v,"velocity of ball",
"public", false, false, false, "unspecified", "none", "unspecified"},
{Boolean,flying,"true, if ball is flying", "public", false, false,
false, "unspecified", "none", "unspecified"},
{Boolean,impact,"",
"public", false, false, false, "unspecified", "none", "unspecified"},
{Real,v_new,"", "public", false, false, false, "unspecified", "none",
"unspecified"}}
```

```
>>getConnectionCount(BouncingBall)
0
```

```
>>getInheritanceCount(BouncingBall)
0
```

```
>>getComponentModifierValue(BouncingBall,e)
0.7
```

```
>>getComponentModifierNames(BouncingBall,e)
```

```
{}
```

```
>>getClassRestriction(BouncingBall)
"model"
```

```
>>getVersion() // Version of the currently running OMC
"1.6"
```

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 export 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:

```
$ cat daequery.mos
loadFile("BouncingBall.mo");
exportDAEtoMatlab(BouncingBall);
readFile("BouncingBall_imatrix.m");

$ omc daequery.mos
true
"The equation system was dumped to Matlab file:BouncingBall_imatrix.m"
"
% Incidence Matrix
% ====================================
% number of rows: 6
IM={[3,-6],[1,{'if', 'true','==' {3},{},}],[2,{'if', 'edge(impact)'
{3},{5},}],[4,2],[5,{'if', 'true','==' {4},{},}],[6,-5]};
VL = {'foo','v_new','impact','flying','v','h'};
```

```
EqStr = {'impact = h <= 0.0;','foo = if impact then 1 else 2;','when {h <= 0.0 AND v
<= 0.0,impact} then v_new = if edge(impact) then (-e) * pre(v) else 0.0; end
when;','when {h <= 0.0 AND v <= 0.0,impact} then flying = v_new > 0.0; end
when;','der(v) = if flying then -g else 0.0;','der(h) = v;'};

OldEqStr={'fclass 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;',''};"
```

1.3 Summary of Commands for the Interactive Session Handler

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

`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 chapter 1.2.14).
`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 References

Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe, 44/45, December 2005. See also: http://www.openmodelica.org.

Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.

The Modelica Association. The Modelica Language Specification Version 3.0, Sept 2007. http://www.modelica.org.

Chapter 2

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

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. A splash screen similar to the one shown in Figure 2-1 will appear indicating that it is starting OMEdit.

2.1.1 Linux

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

2.1.2 Mac OS X

?? fill in

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.

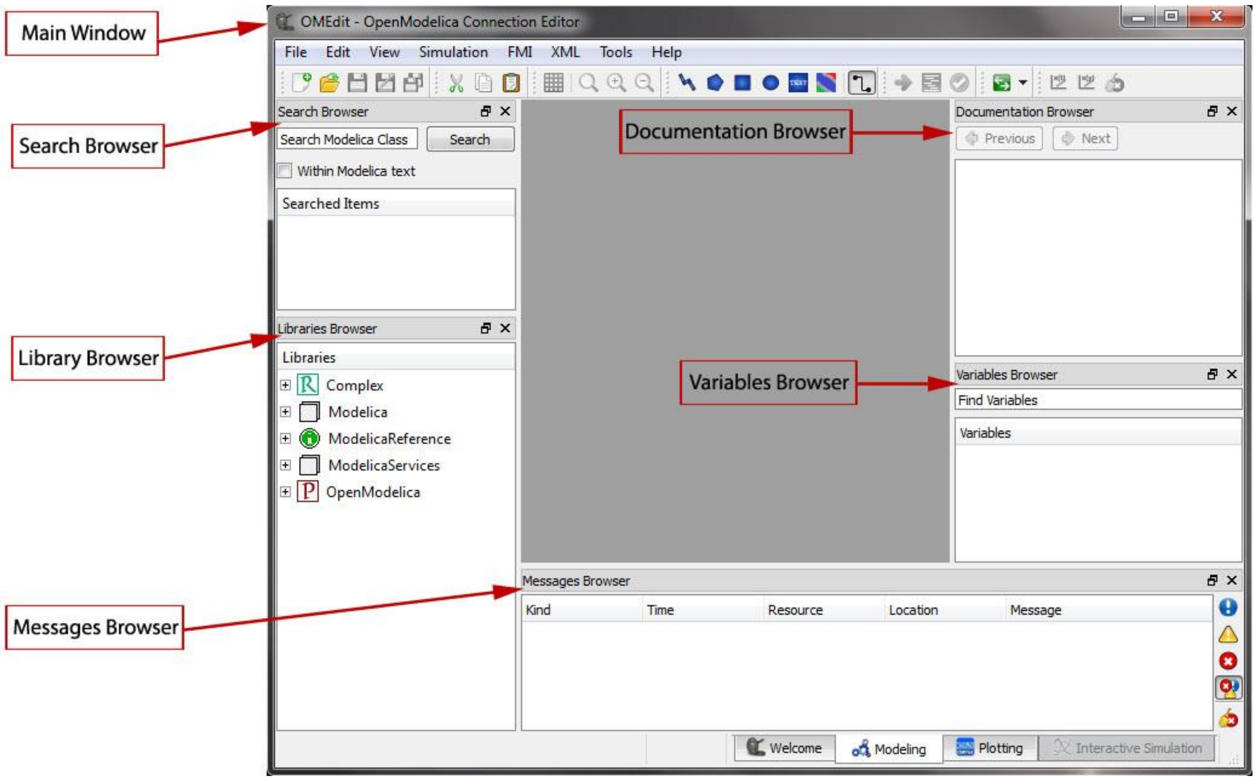
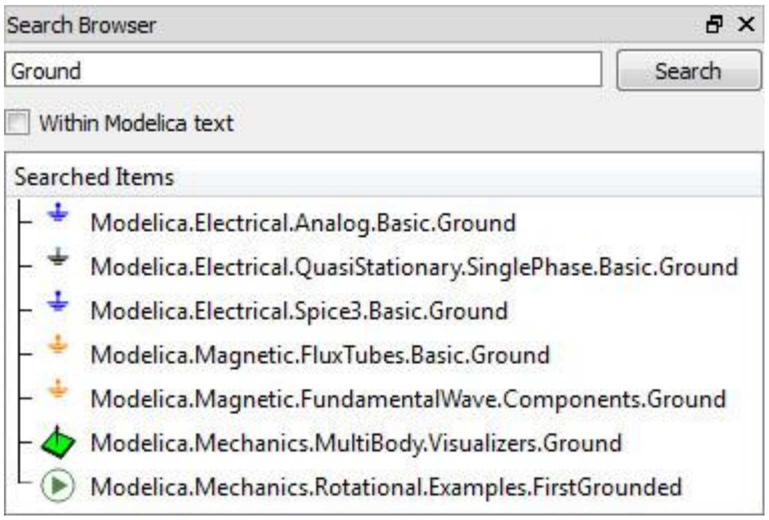


Figure 2-2: OMEdit MainWindow and Browsers.

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 2.9.1.

2.2.1 Search Browser



Modelica.Electrical.Analog.Basic.Ground
Modelica.Electrical.QuasiStationary.SinglePhase.Basic.Ground
Modelica.Electrical.Spice3.Basic.Ground
Modelica.Magnetic.FluxTubes.Basic.Ground
Modelica.Magnetic.FundamentalWave.Components.Ground
Modelica.Mechanics.MultiBody.Visualizers.Ground
Modelica.Mechanics.Rotational.Examples.FirstGrounded

Figure 2-3: 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.

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 2.9.1.

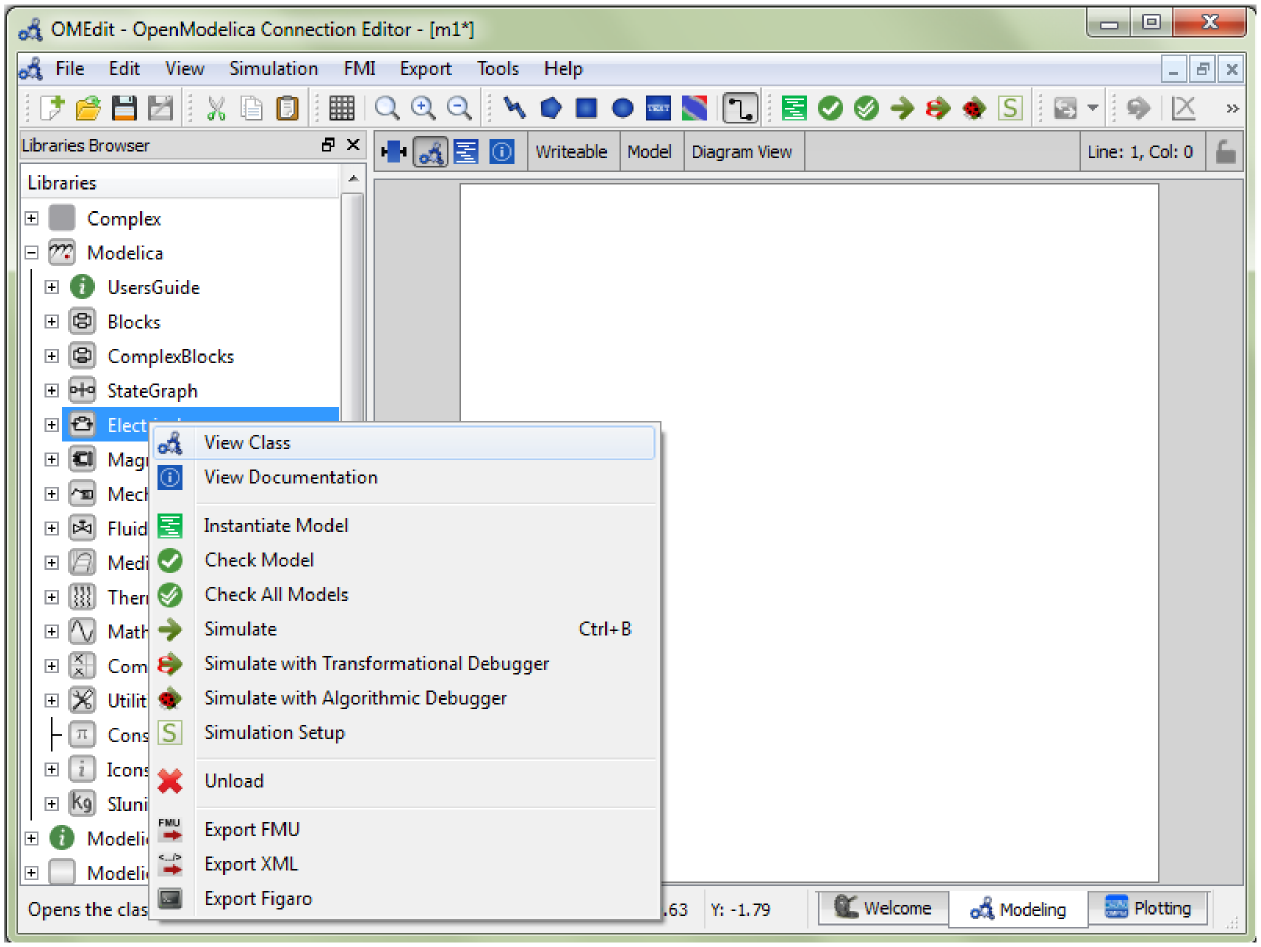
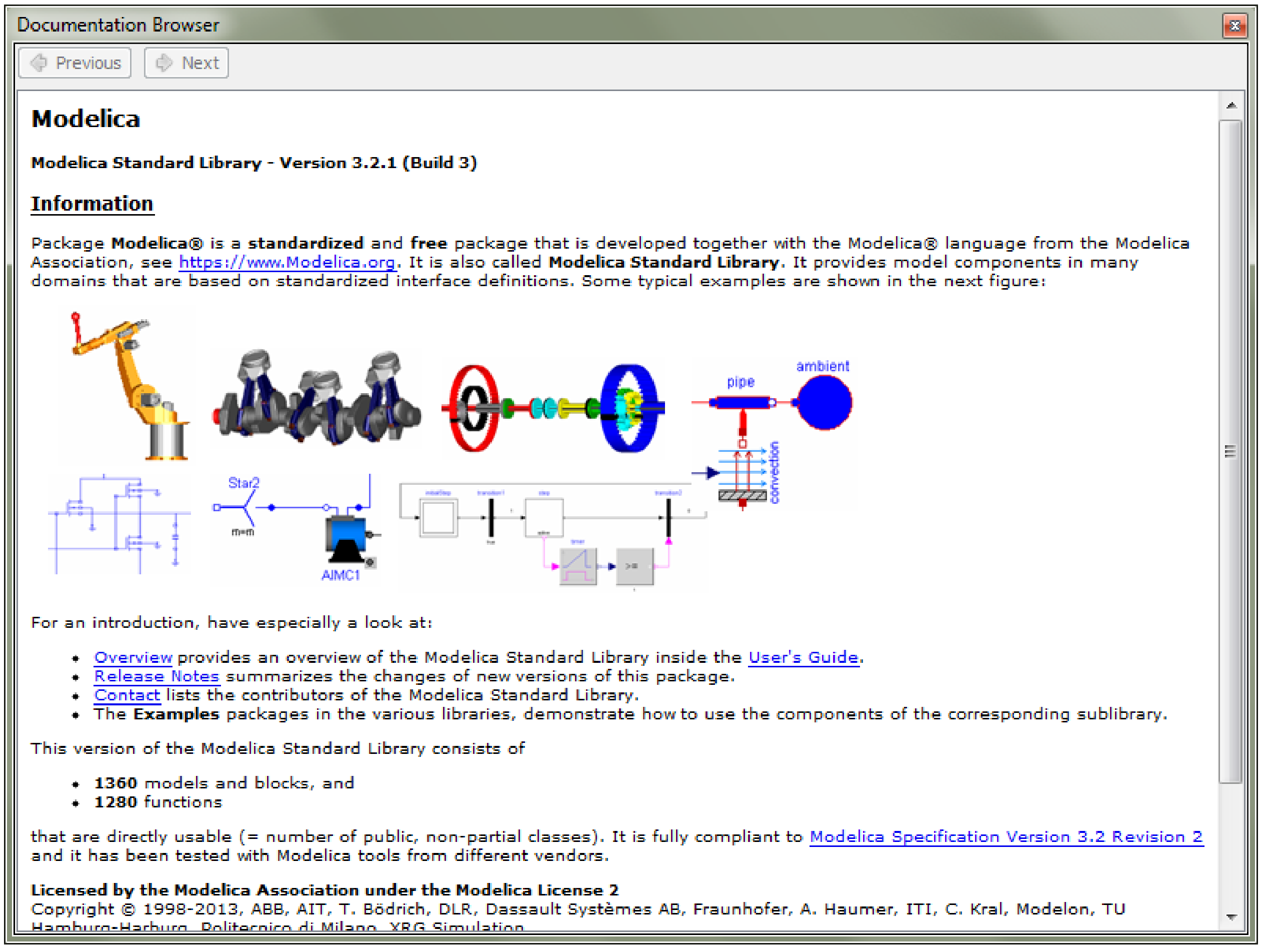


Figure 2-4: Libraries Browser.

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`.



Modelica

Modelica Standard Library - Version 3.2.1 (Build 3)

Information

Package **Modelica®** is a **standardized** and **free** package that is developed together with the Modelica® language from the Modelica Association, see https://www.Modelica.org. It is also called **Modelica Standard Library**. It provides model components in many domains that are based on standardized interface definitions. Some typical examples are shown in the next figure:

For an introduction, have especially a look at:

- Overview provides an overview of the Modelica Standard Library inside the User's Guide.
- Release Notes summarizes the changes of new versions of this package.
- Contact lists the contributors of the Modelica Standard Library.
- The **Examples** packages in the various libraries, demonstrate how to use the components of the corresponding sublibrary.

This version of the Modelica Standard Library consists of

- **1360** models and blocks, and
- **1280** functions

that are directly usable (= number of public, non-partial classes). It is fully compliant to Modelica Specification Version 3.2 Revision 2 and it has been tested with Modelica tools from different vendors.

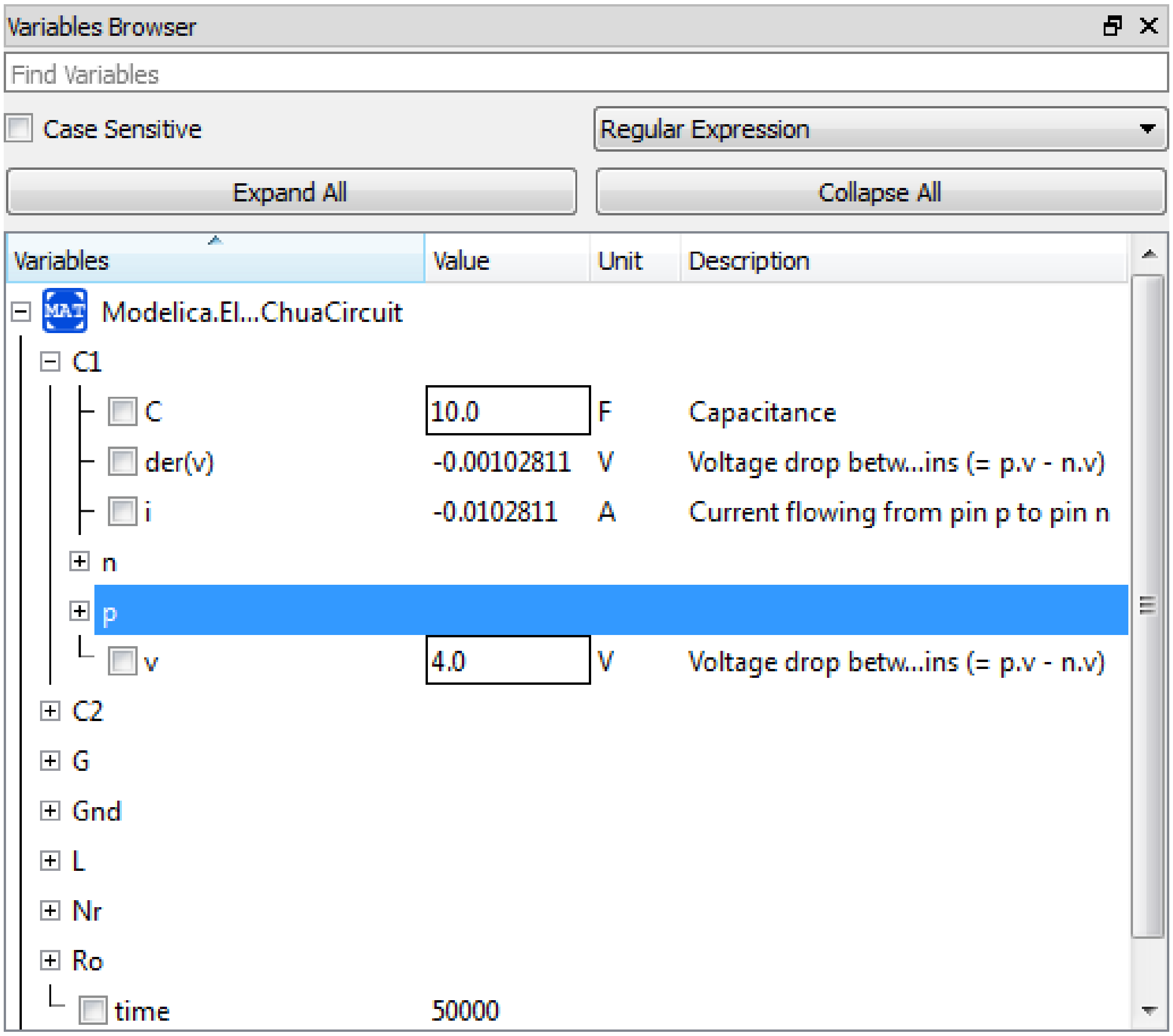
Licensed by the Modelica Association under the Modelica License 2
Copyright © 1998-2013, ABB, AIT, T. Bödrich, DLR, Dassault Systèmes AB, Fraunhofer, A. Haumer, ITI, C. Kral, Modelon, TU Hamburg-Harburg, Politecnico di Milano, XRG Simulation

Figure 2-5: Documentation Browser.

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-simulation. See section **Error! Reference source not found.**. It also displays the unit and description of the variable.



Variables	Value	Unit	Description
Modelica.El...ChuaCircuit			
C1			
C	10.0	F	Capacitance
der(v)	-0.00102811	V	Voltage drop betw...ins (= p.v - n.v)
i	-0.0102811	A	Current flowing from pin p to pin n
n			
p			
v	4.0	V	Voltage drop betw...ins (= p.v - n.v)
C2			
G			
Gnd			
L			
Nr			
Ro			
time	50000		

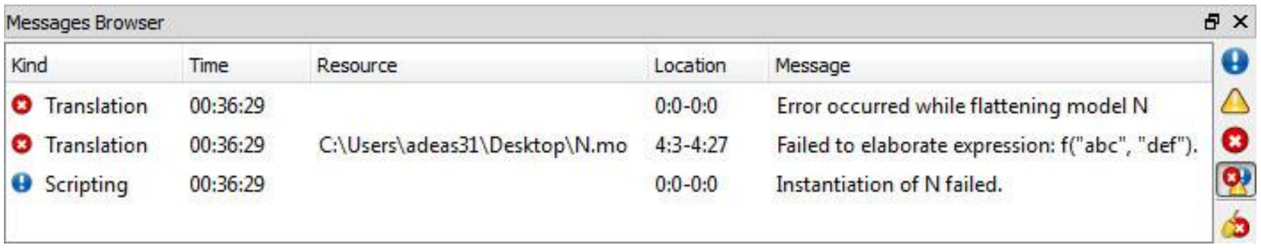
Figure 2-6: Variables Browser.

2.2.5 Messages Browser

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

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

`Messages Browser` contains some manipulation buttons on the right side which allows filtering of the error messages. The browser has a right click menu for copying and deleting the error messages.



Kind	Time	Resource	Location	Message
Translation	00:36:29		0:0-0:0	Error occurred while flattening model N
Translation	00:36:29	C:\Users\adeas31\Desktop\N.mo	4:3-4:27	Failed to elaborate expression: f("abc", "def").
Scripting	00:36:29		0:0-0:0	Instantiation of N failed.

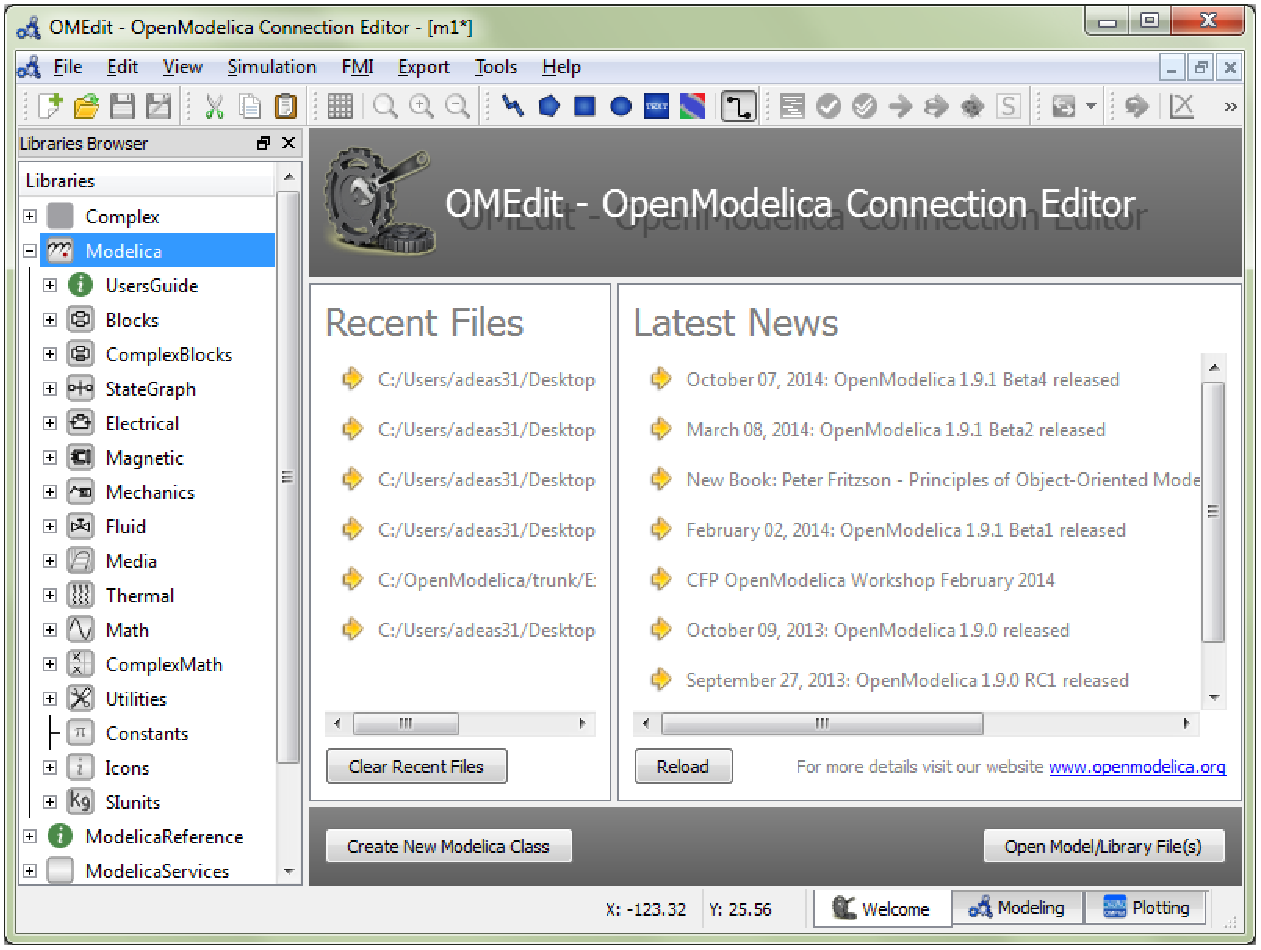
Figure 2-7: Messages Browser.

2.3 Perspectives

The perspective tabs are loacted at the bottom right of the `MainWindow`,

- Welcome Perspective
- Modeling Perspective
- Plotting Perspective

2.3.1 Welcome Perspective



OMEdit - OpenModelica Connection Editor - [m1*]
File Edit View Simulation FMI Export Tools Help
Libraries Browser
Libraries
Complex
Modelica
UsersGuide
Blocks
ComplexBlocks
StateGraph
Electrical
Magnetic
Mechanics
Fluid
Media
Thermal
Math
ComplexMath
Utilities
Constants
Icons
SIunits
ModelicaReference
ModelicaServices
OMEdit - OpenModelica Connection Editor
Recent Files
C:/Users/adeas31/Desktop
C:/Users/adeas31/Desktop
C:/Users/adeas31/Desktop
C:/Users/adeas31/Desktop
C:/OpenModelica/trunk/E:
C:/Users/adeas31/Desktop
Clear Recent Files
Latest News
October 07, 2014: OpenModelica 1.9.1 Beta4 released
March 08, 2014: OpenModelica 1.9.1 Beta2 released
New Book: Peter Fritzson - Principles of Object-Oriented Mode
February 02, 2014: OpenModelica 1.9.1 Beta1 released
CFP OpenModelica Workshop February 2014
October 09, 2013: OpenModelica 1.9.0 released
September 27, 2013: OpenModelica 1.9.0 RC1 released
Reload
For more details visit our website www.openmodelica.org
Create New Modelica Class
Open Model/Library File(s)
X: -123.32 Y: 25.56
Welcome
Modeling
Plotting

Figure 2-8: OMEdit Welcome Perspective.

The `Welcome Perspective` shows the list of recent files and the list of latest news from openmodelica.org. See Figure 2-8. The orientation of recent files and latest news can be horizontal or vertical. User is allowed to show/hide the latest news. See section 2.9.1.

2.3.2 Modeling Perspective

The `Modeling Perpective` provides the interface where user can create and design their models. See Figure 2-9.

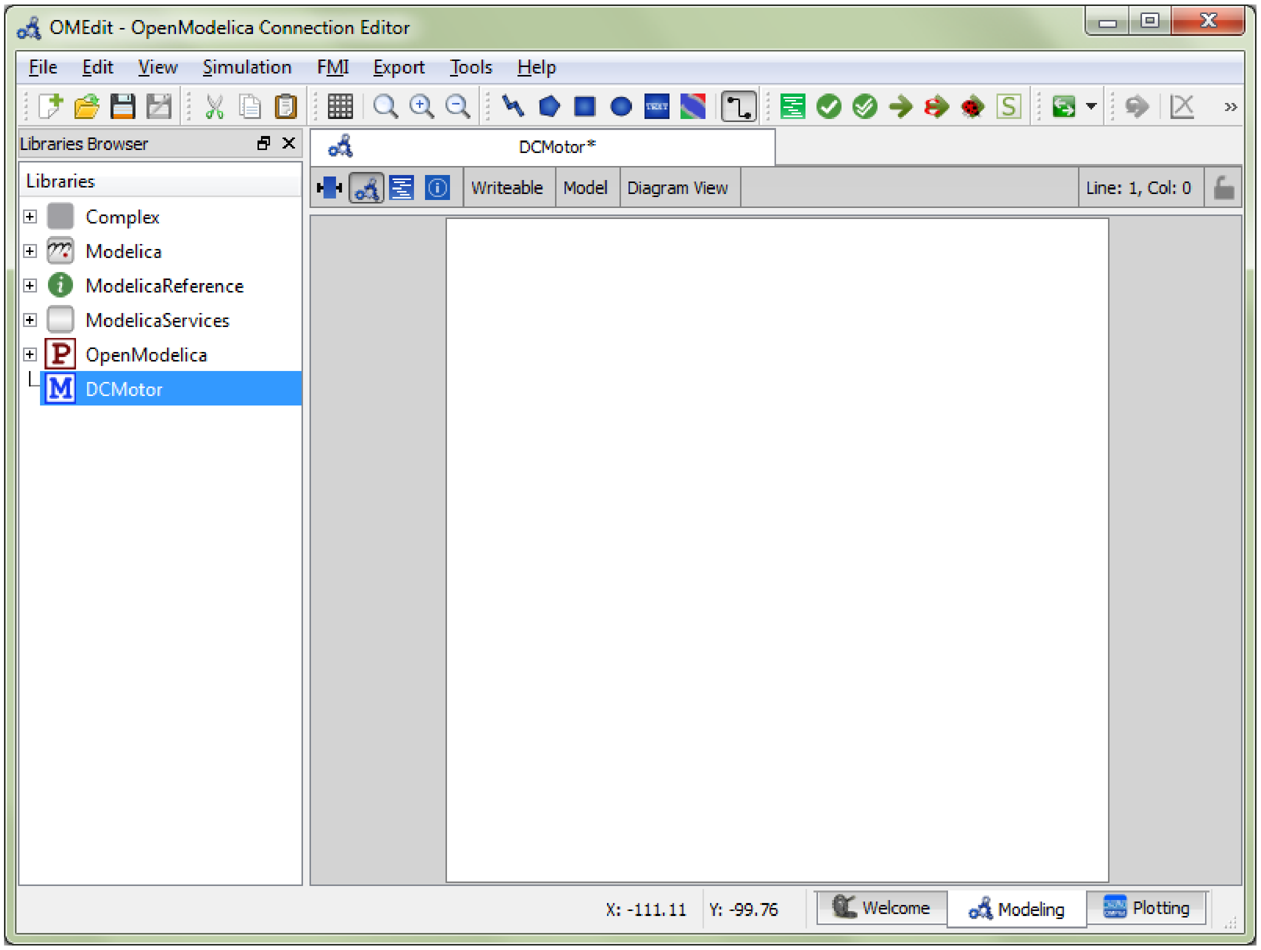


Figure 2-9: OMEdit Modeling Perspective.

The `Modeling Perspective` interface can be viewed in two different modes, the `tabbed view` and `subwindow view`, see section 2.9.1.

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 2.9.1.

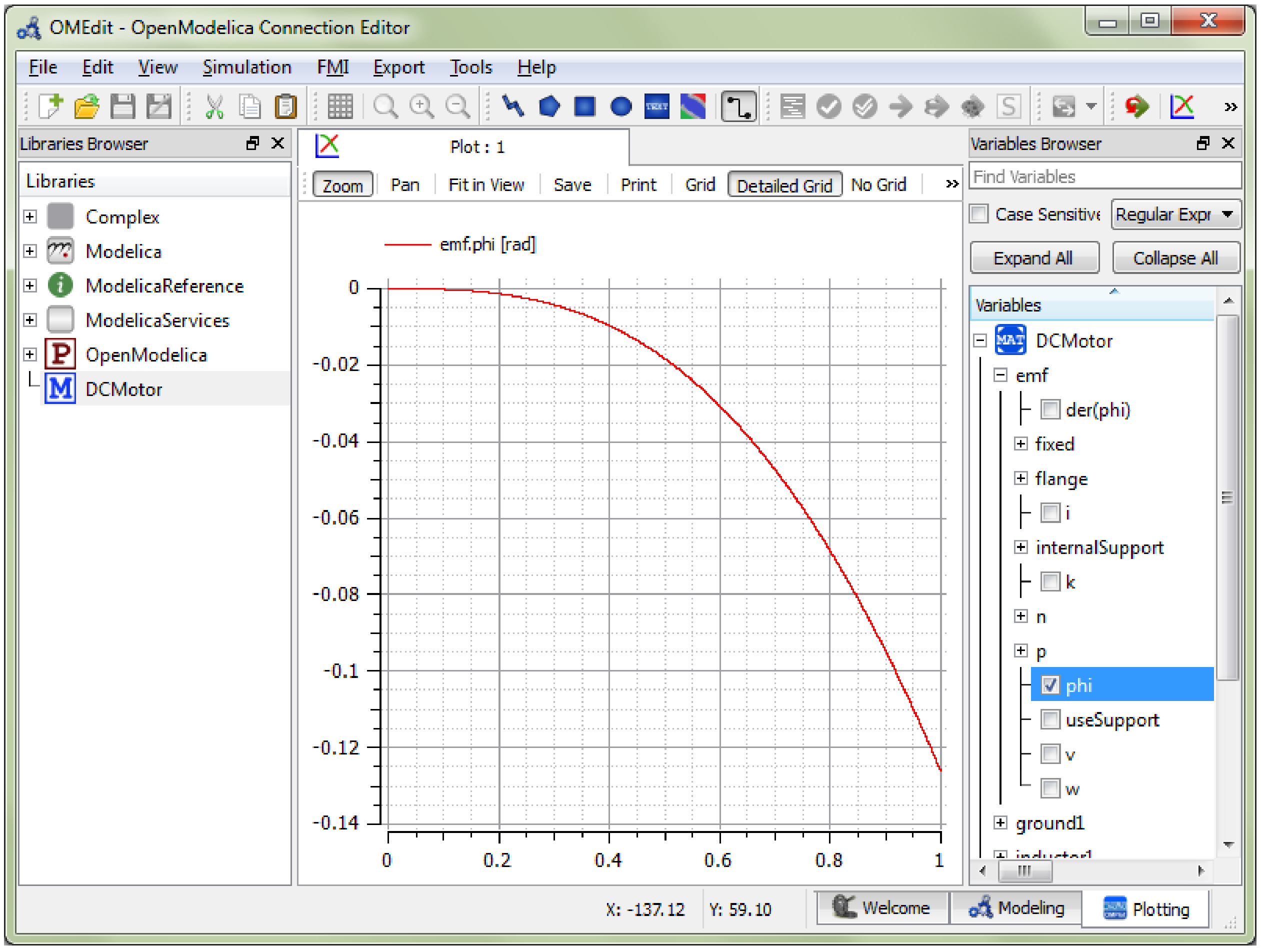


Figure 2-10: OMEdit Plotting Perspective.

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`.
- 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 2-11.

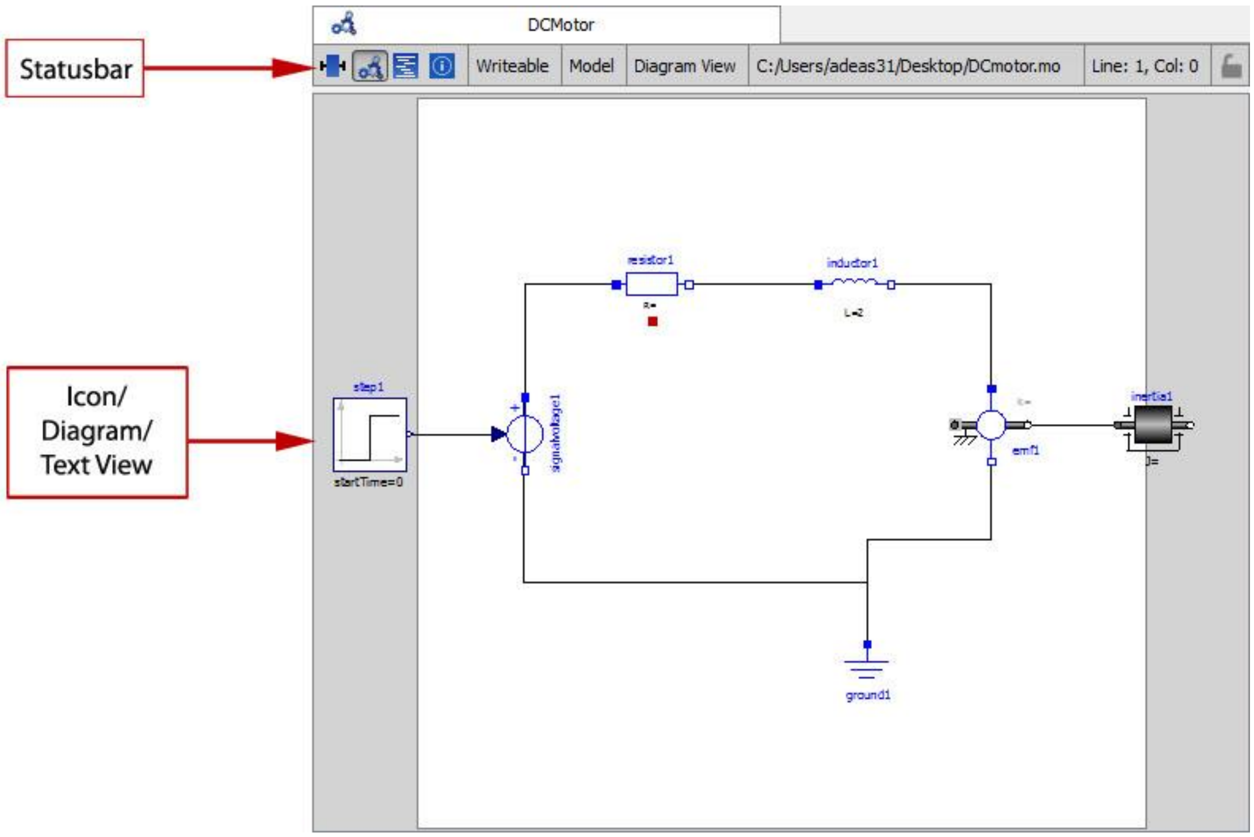


Figure 2-11: Model Widget showing the Diagram View.

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. See Figure 2-12.

Figure 2-12: Connect/Unconnect Mode toolbar button.

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 Appendix C 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).*
- *Protecetd 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.
- *Optimization Method (Optional)* – specifies the initialization optimization 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 2-10.

2.6.1 Types of Plotting

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

2.6.1.1 Time Plot

Plots the variable over the simulation time. You can have multiple `Time Plot` windows by clicking on New Plot Window toolbar button. See Figure 2-13.

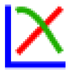


Figure 2-13: New Plot Window toolbar button.

2.6.1.2 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. See Figure 2-14.

Figure 2-14: New Plot Parametric toolbar button.

2.7 Re-simulating a Model

The `Variables Browser` allows manipulation of changeable parameters for re-simulation as shown in Figure 2-6. 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.

Figure 2-15: Re-simulate toolbar button.

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 2-16.

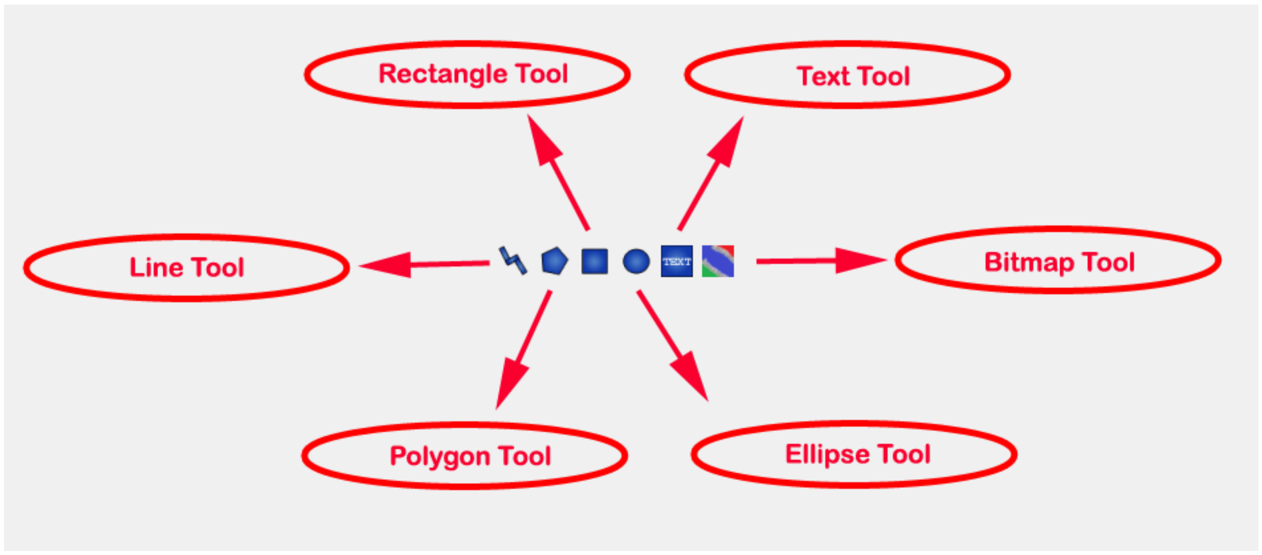


Figure 2-16: User defined shapes.

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}, pattern = LinePattern.Solid, fillPattern = FillPattern.None, lineThickness
= 0.25, extent = {{ -64.5,88},{63, -22.5}}),Polygon(points = {{ -47.5, -29.5},{52.5, -
29.5},{4.5, -86},{ -47.5, -29.5}}, rotation = 0, lineColor = {0,0,255}, fillColor =
{0,0,255}, pattern = LinePattern.Solid, fillPattern = FillPattern.None, lineThickness
= 0.25)}));
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.

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.
  - *Preserve User's GUI Customizations* – If true then OMEdit will remember its windows and toolbars positions and sizes.
  - *Global Precision Value* – Sets the global precision value for all the floating point spin boxes.
- Libraries Browser
  - *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 ture then always saves the class before running the simulation.

2.9.6 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.

2.9.7 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.

2.9.8 Fill Style

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

2.9.9 Curve Style

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

2.9.10 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.

2.10 The Equation-based Debugger

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

2.10.1 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 2.9.5) OR alternatively click on the checkbox *Generate operations in the info xml* in `Tools->Options->Debugger` (see section 2.9.10) which performs the same thing.

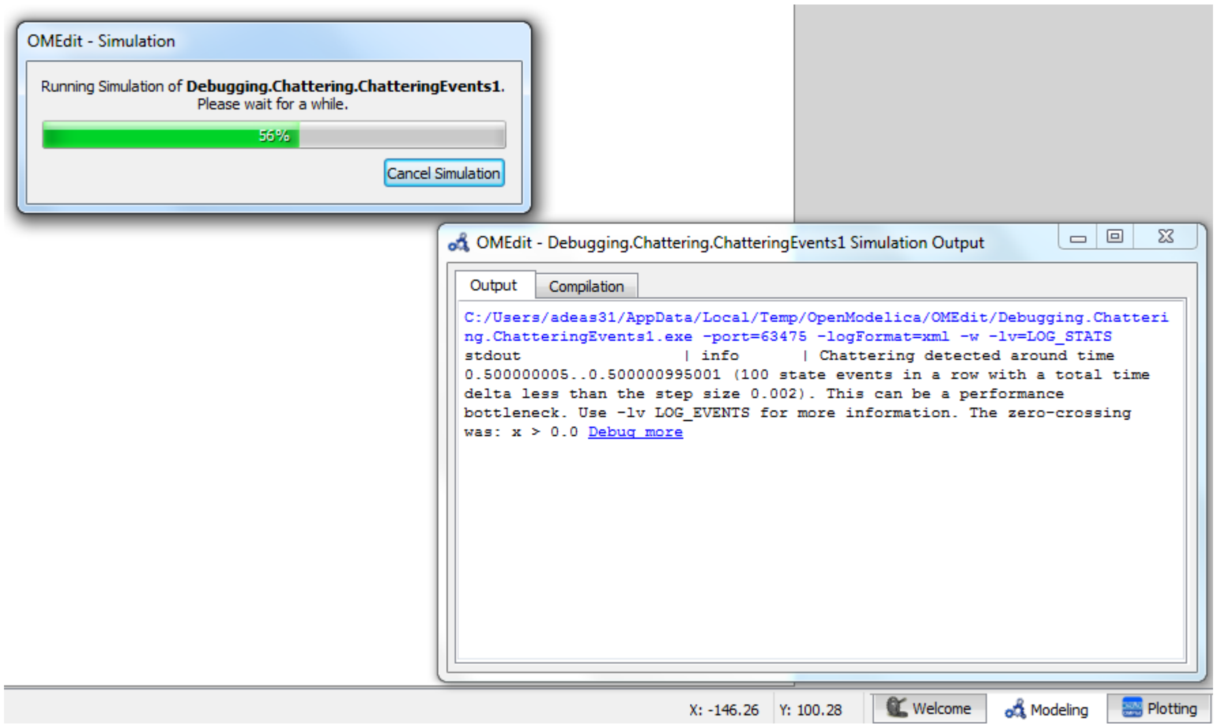
This adds all the transformations performed by OpenModelica on the equations and variables stored in the `model_info.xml` file. This is necessary for the debugger to be able to show the whole path from the source equation(s) to the position of the bug.

2.10.2 Load a Model to Debug

Load an interesting model. We will use the package https://openmodelica.org/svn/OpenModelica/trunk/testsuite/openmodelica/debugging/Debugging.mo since it contains suitable, broken models to demonstrate common errors.

2.10.3 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.


OMEdit - Simulation
Running Simulation of Debugging.Chattering.ChatteringEvents1.
Please wait for a while.
56%
Cancel Simulation
OMEdit - Debugging.Chattering.ChatteringEvents1 Simulation Output
Output
Compilation
X: -146.26
Y: 100.28
Welcome
Modeling
Plotting

```
C:/Users/adeas31/AppData/Local/Temp/OpenModelica/OMEdit/Debugging.Chatteri
ng.ChatteringEvents1.exe -port=63475 -logFormat=xml -w -lv=LOG_STATS
stdout                    | info        | Chattering detected around time
0.500000005..0.500000995001 (100 state events in a row with a total time
delta less than the step size 0.002). This can be a performance
bottleneck. Use -lv LOG_EVENTS for more information. The zero-crossing
was: x > 0.0 Debug more
```

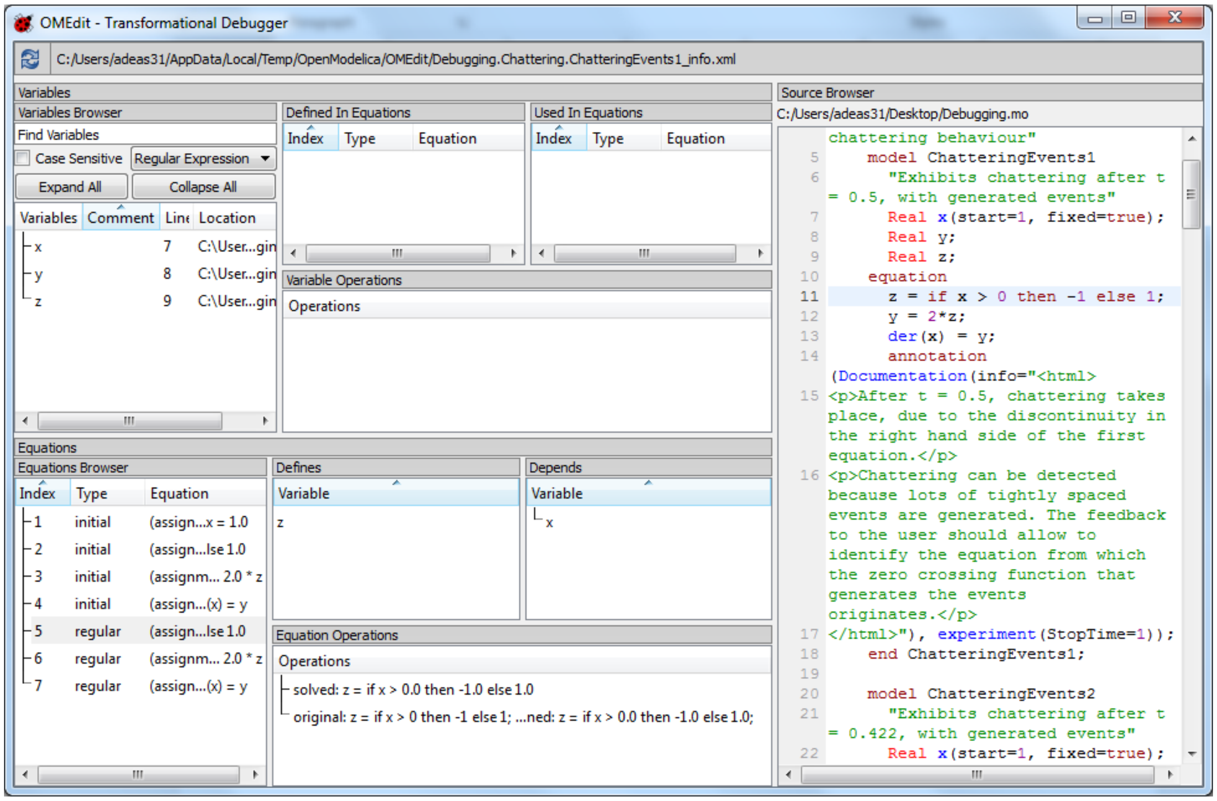
Figure 2-17. Simulating the model.

2.10.4 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.xml)`.


OMEdit - Transformational Debugger

C:/Users/adeas31/AppData/Local/Temp/OpenModelica/OMEdit/Debugging.Chattering.ChatteringEvents1_info.xml

Variables — Variables Browser

Find Variables

Case Sensitive · Regular Expression · Expand All · Collapse All

Variables	Comment	Line	Location
x		7	C:\User...gin
y		8	C:\User...gin
z		9	C:\User...gin

Defined In Equations

Index	Type	Equation

Used In Equations

Index	Type	Equation

Variable Operations

Operations

Equations — Equations Browser

Index	Type	Equation
1	initial	(assign...x = 1.0
2	initial	(assign...lse 1.0
3	initial	(assignm... 2.0 * z
4	initial	(assign...(x) = y
5	regular	(assign...lse 1.0
6	regular	(assignm... 2.0 * z
7	regular	(assign...(x) = y

Defines

Variable
z

Depends

Variable
x

Equation Operations

Operations

solved: z = if x > 0.0 then -1.0 else 1.0

original: z = if x > 0 then -1 else 1; ...ned: z = if x > 0.0 then -1.0 else 1.0;

Source Browser

C:/Users/adeas31/Desktop/Debugging.mo

```
chattering behaviour"
    model ChatteringEvents1
      "Exhibits chattering after t
= 0.5, with generated events"
      Real x(start=1, fixed=true);
      Real y;
      Real z;
    equation
      z = if x > 0 then -1 else 1;
      y = 2*z;
      der(x) = y;
      annotation
(Documentation(info="<html>
<p>After t = 0.5, chattering takes
place, due to the discontinuity in
the right hand side of the first
equation.</p>
<p>Chattering can be detected
because lots of tightly spaced
events are generated. The feedback
to the user should allow to
identify the equation from which
the zero crossing function that
generates the events
originates.</p>
</html>"), experiment(StopTime=1));
    end ChatteringEvents1;

    model ChatteringEvents2
      "Exhibits chattering after t
= 0.422, with generated events"
      Real x(start=1, fixed=true);
```

Figure 2-18. Transfomation Debugger.

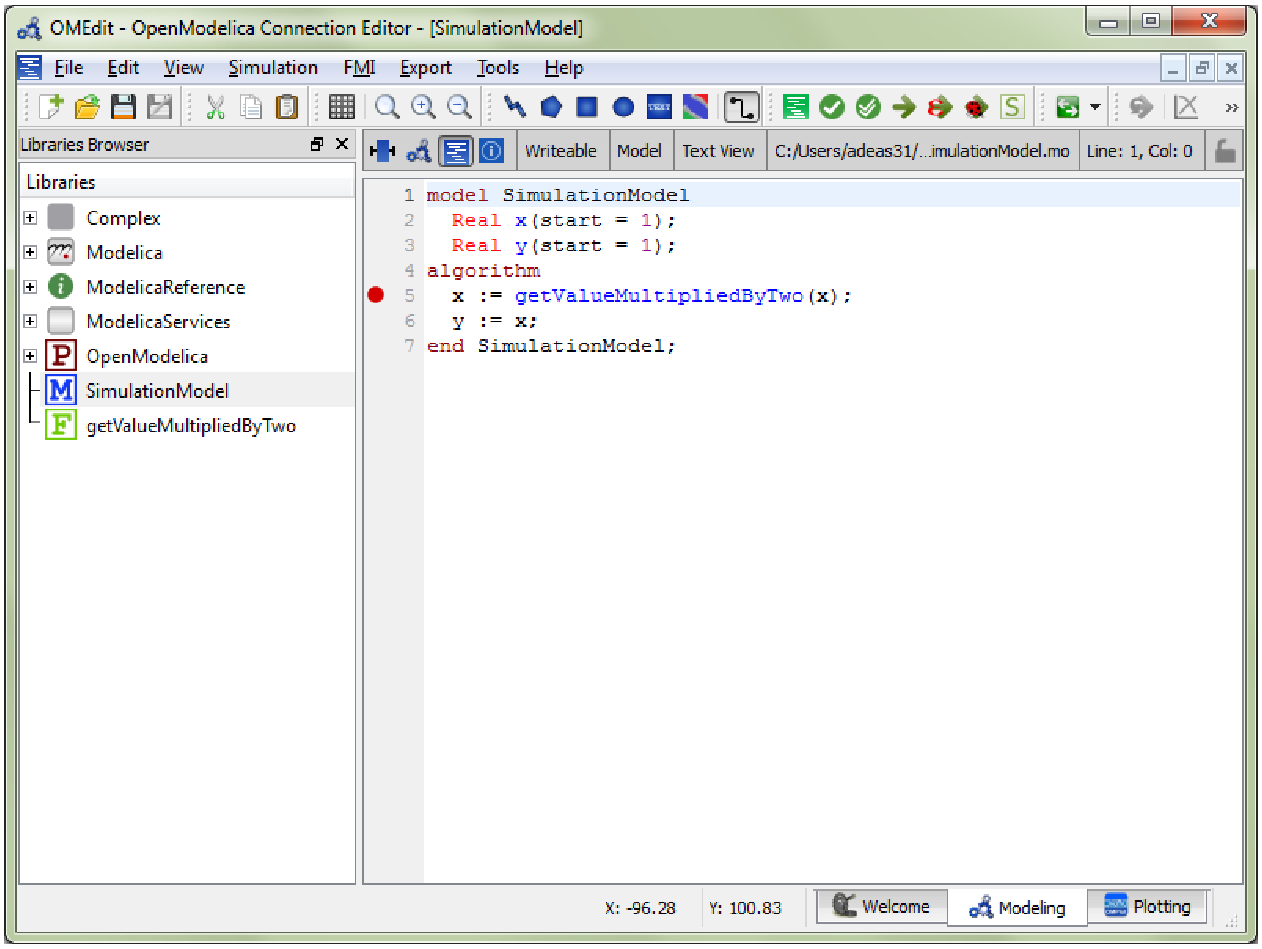
2.11 The Algorithmic Debugger

This section gives a short description how to get started using the algorithmic debugger in OMEdit. See section 2.9.10 for further details of debugger options/settings. The `Algorithmic Debugger` window can be launched from `Tools->Windows->Algorithmic Debugger`.

2.11.1 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 2-19.
- Open the `Algorithmic Debugger` window and add a breakpoint using the right click menu of `Breakpoints Browser` window.


OMEdit - OpenModelica Connection Editor - [SimulationModel]
File Edit View Simulation FMI Export Tools Help
Libraries Browser
Libraries
Complex
Modelica
ModelicaReference
ModelicaServices
OpenModelica
SimulationModel
getValueMultipliedByTwo
Writeable Model Text View C:/Users/adeas31/...imulationModel.mo Line: 1, Col: 0
X: -96.28 Y: 100.83
Welcome Modeling Plotting

```
model SimulationModel
  Real x(start = 1);
  Real y(start = 1);
algorithm
  x := getValueMultipliedByTwo(x);
  y := x;
end SimulationModel;
```

Figure 2-19: Adding breakpoint in Text View.

2.11.2 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.

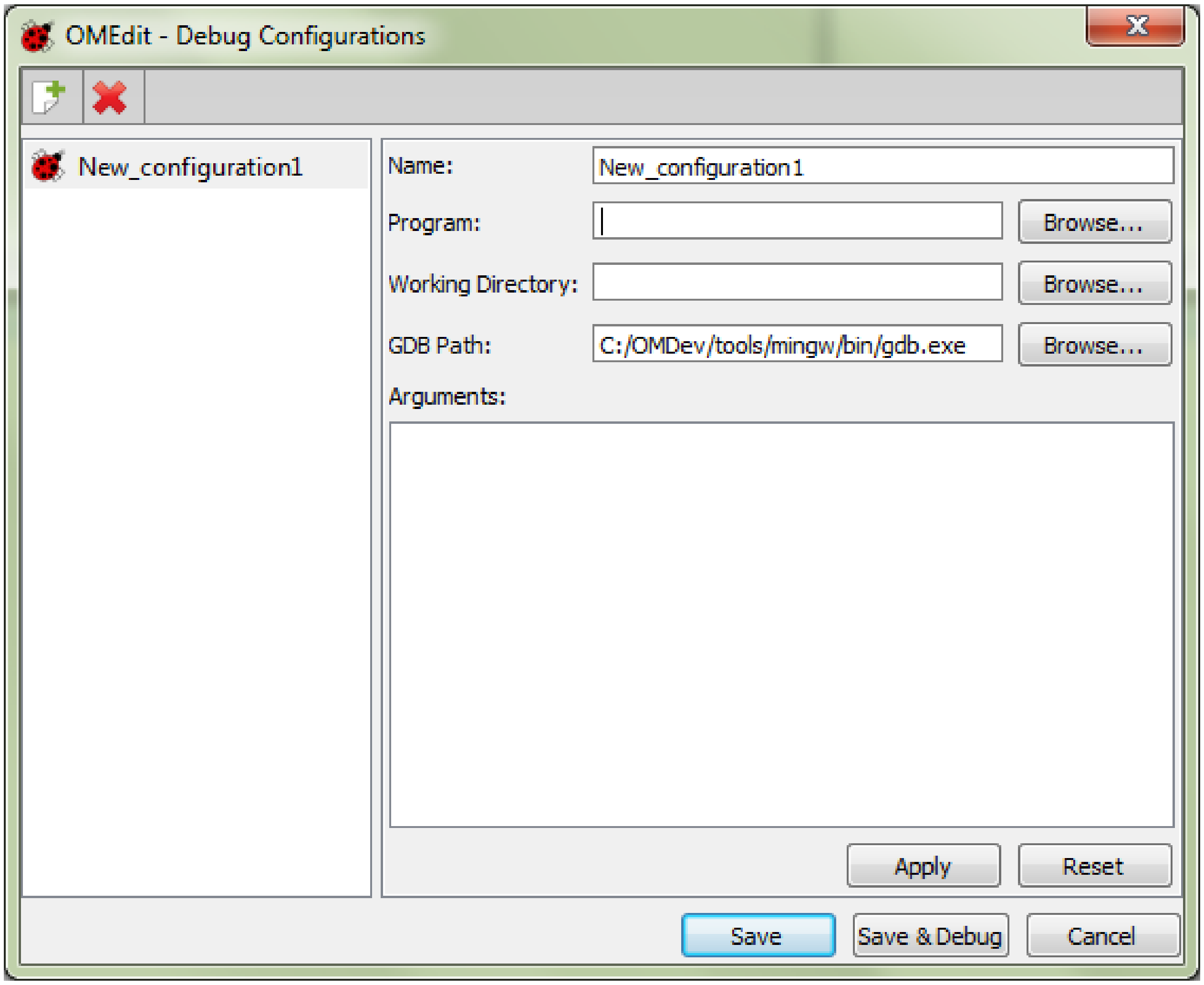
There are four ways to start the debugger,

- 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` (see section 2.11.3).
- Open the `Algorithmic Debugger` window and from menu select `Debug->Attach to Running Process` (see section 2.11.4).

2.11.3 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.



Name: New_configuration1

Program:

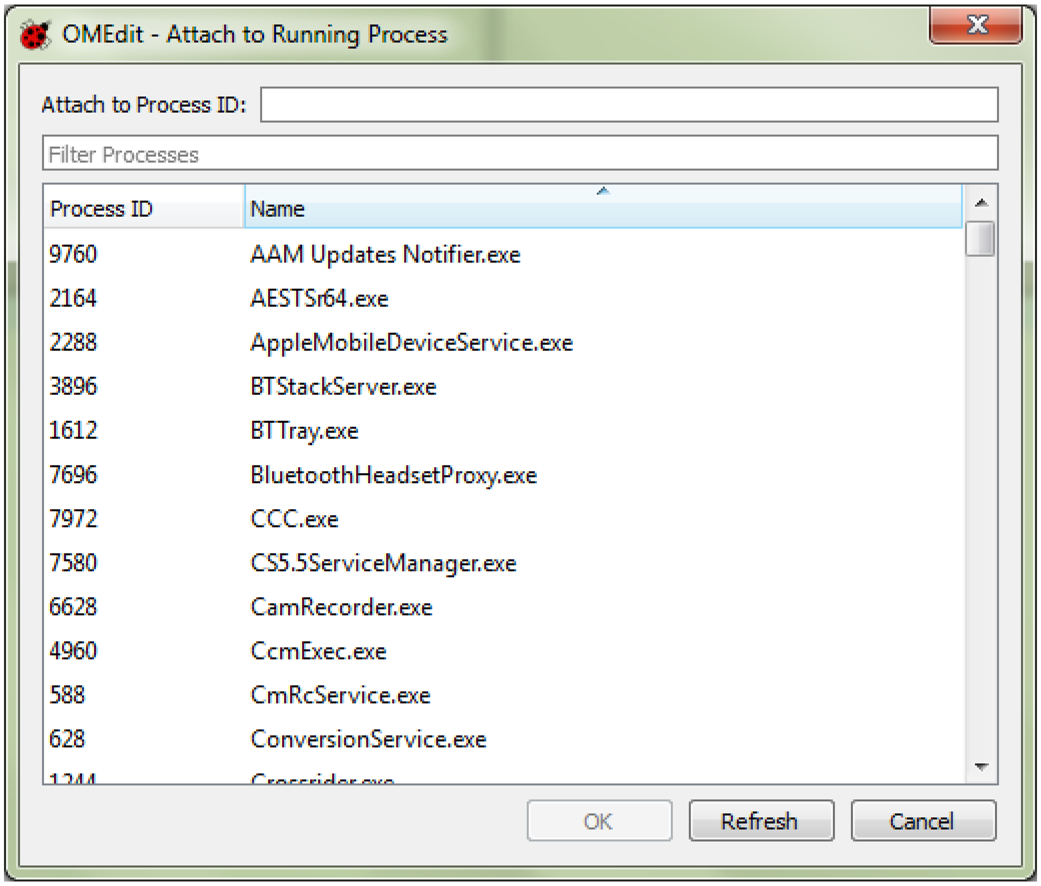
Working Directory:

GDB Path: C:/OMDev/tools/mingw/bin/gdb.exe

Figure 2-20: Debug Configurations.

2.11.4 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 2-21 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.



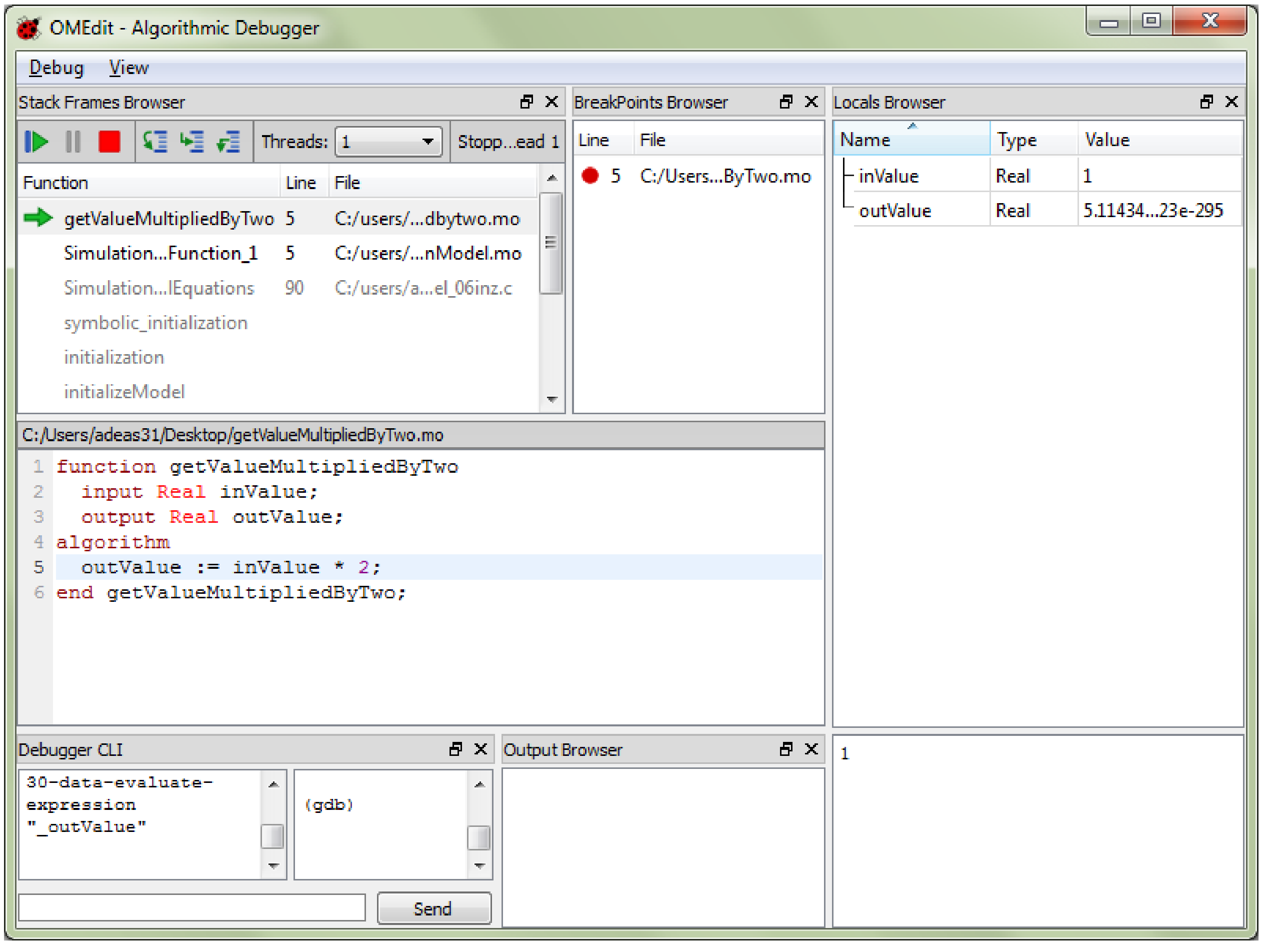
Process ID	Name
9760	AAM Updates Notifier.exe
2164	AESTSr64.exe
2288	AppleMobileDeviceService.exe
3896	BTStackServer.exe
1612	BTTray.exe
7696	BluetoothHeadsetProxy.exe
7972	CCC.exe
7580	CS5.5ServiceManager.exe
6628	CamRecorder.exe
4960	CcmExec.exe
588	CmRcService.exe
628	ConversionService.exe

Figure 2-21: Attach to Running Process.

2.11.5 Using the Algorithmic Debugger Window

Figure 2-22 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.



Function	Line	File
getValueMultipliedByTwo	5	C:/users/...dbytwo.mo
Simulation...Function_1	5	C:/users/...nModel.mo
Simulation...lEquations	90	C:/users/a...el_06inz.c
symbolic_initialization		
initialization		
initializeModel		

Line	File
5	C:/Users...ByTwo.mo

Name	Type	Value
inValue	Real	1
outValue	Real	5.11434...23e-295

C:/Users/adeas31/Desktop/getValueMultipliedByTwo.mo

```
function getValueMultipliedByTwo
  input Real inValue;
  output Real outValue;
algorithm
  outValue := inValue * 2;
end getValueMultipliedByTwo;
```

```
30-data-evaluate-
expression
"_outValue"
```

```
(gdb)
```

```
1
```

Figure 2-22: Algorithmic Debugger.

Chapter 3

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);
  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=10`. The simulation is started with the command below.

```
simulate(HelloWorld, startTime=0, stopTime=4, numberOfIntervals=10);
```

When the simulation is finished the file `HelloWorld res.plt` contains the simulation data. The contents of the file is the following (some formatting has been applied).

```
0                      1
4.440892098500626e-013 0.9999999999995559
0.4444444444444444     0.6411803884299349
0.8888888888888888     0.411112290507163
1.333333333333333      0.2635971381157249
1.777777777777778      0.1690133154060587
2.222222222222222      0.1083680232218813
2.666666666666667      0.06948345122279623
3.111111111111112      0.04455142624447787
3.555555555555556      0.02856550078454138
4                      0.01831563888872685
```

Diagrams are now created with the new `OMPlot` program by using the following command.

```
plot(x);
```

seems to correspond well with the data.

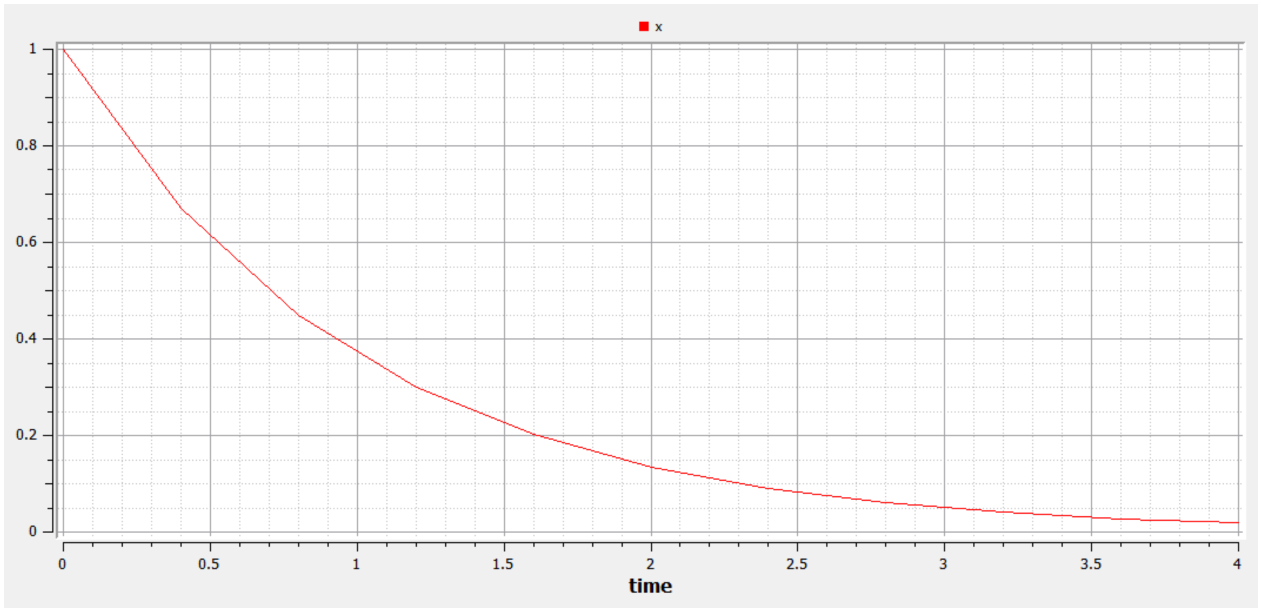


Figure 3-1: Simple 2D plot of the HelloWorld example.

By re-simulating and saving results at many more points, e.g. using the default 500 intervals, a much smoother plot can be obtained.

```
simulate(HelloWorld, startTime=0, stopTime=4, numberOfIntervals=500);
```

```
plot(x);
```

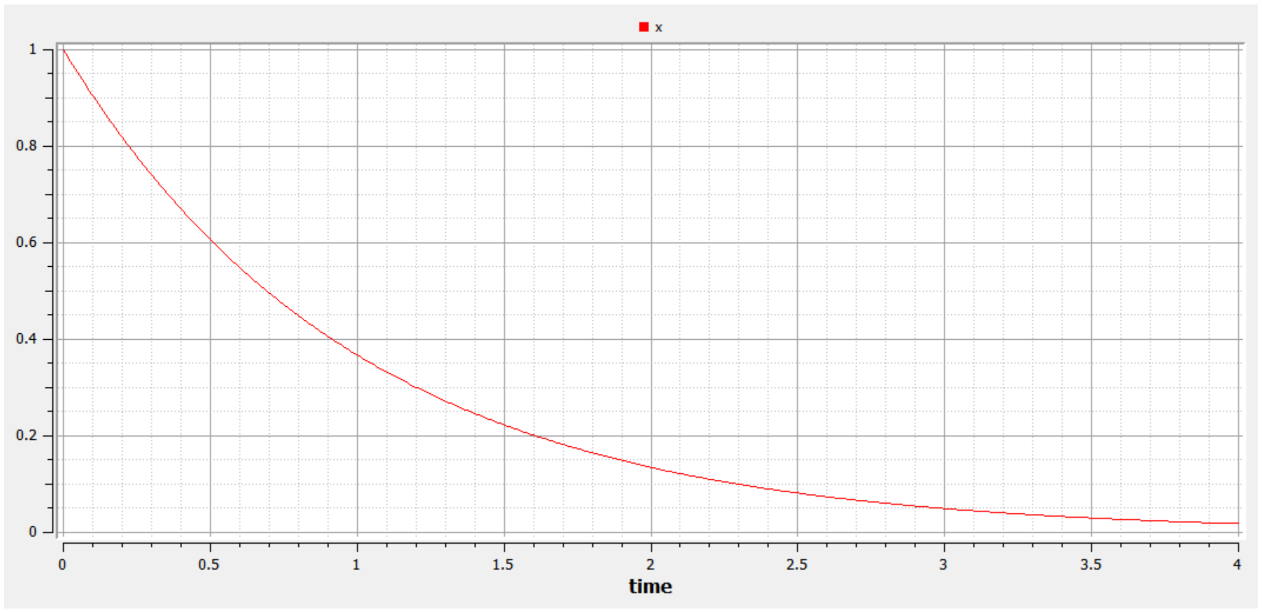


Figure 3-2: Simple 2D plot of the HelloWorld example with larger number of points.

3.2 Plotting Commands and their Options

Command	**Description**
`plot(x)`	Creates a diagram with data from the last simulation that had a variable named x.
`plot({x,y,..., z})`	Like the previous command, but with several variables.
`plotParametric(x, y)`	Creates a parametric diagram with data from the last simulated variables named x and y.
`plotParametric(x, {y1,y2})`	Like the previous command, but with several variables.
`plotAll()`	Creates a diagram with all variables from the last simulated model as functions of time.

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.
`legendPosition`	`"top"`	Defines the position of the legend in the diagram. Possible values are left, right, top, bottom and none.
`externalWindow`	`false`	Opens a new `OMPlot` window if set to true otherwise update the current opened window.

Chapter 4

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.

4.1 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.

4.1.1 Mathematica Notebooks

Literate Programming (Knuth 1984) is a form of programming where programs are integrated with documentation in the same document. Mathematica notebooks (Wolfram 1997) 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, e.g. see Figure 4-1below and Chapter 19 in (Fritzson 2004)

4.1.2 OMNotebook

The OMNotebook software (Axelsson 2005, Fernström 2006) is a new open source free software that gives an interactive WYSIWYG (What-You-See-Is-What-You-Get) 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 (What-You-See-Is-What-You-Get) 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 4-1.

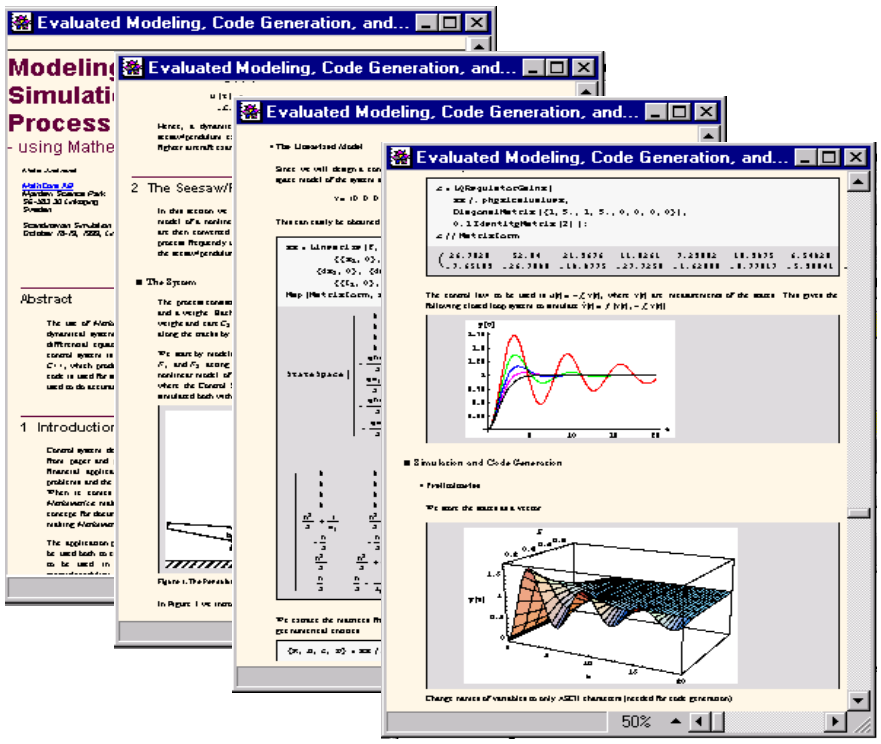

Evaluated Modeling, Code Generation, and...
Evaluated Modeling, Code Generation, and...
Evaluated Modeling, Code Generation, and...
Evaluated Modeling, Code Generation, and...
Modelin
Simulati
Process
- using Mathe
Abstract
1 Introductio
2 The Seesaw/I
50%

Figure 4-1. Examples of Mathematica notebooks in the MathModelica modeling and simulation environment.

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.

4.2 DrModelica Tutoring System – an Application of OMNotebook

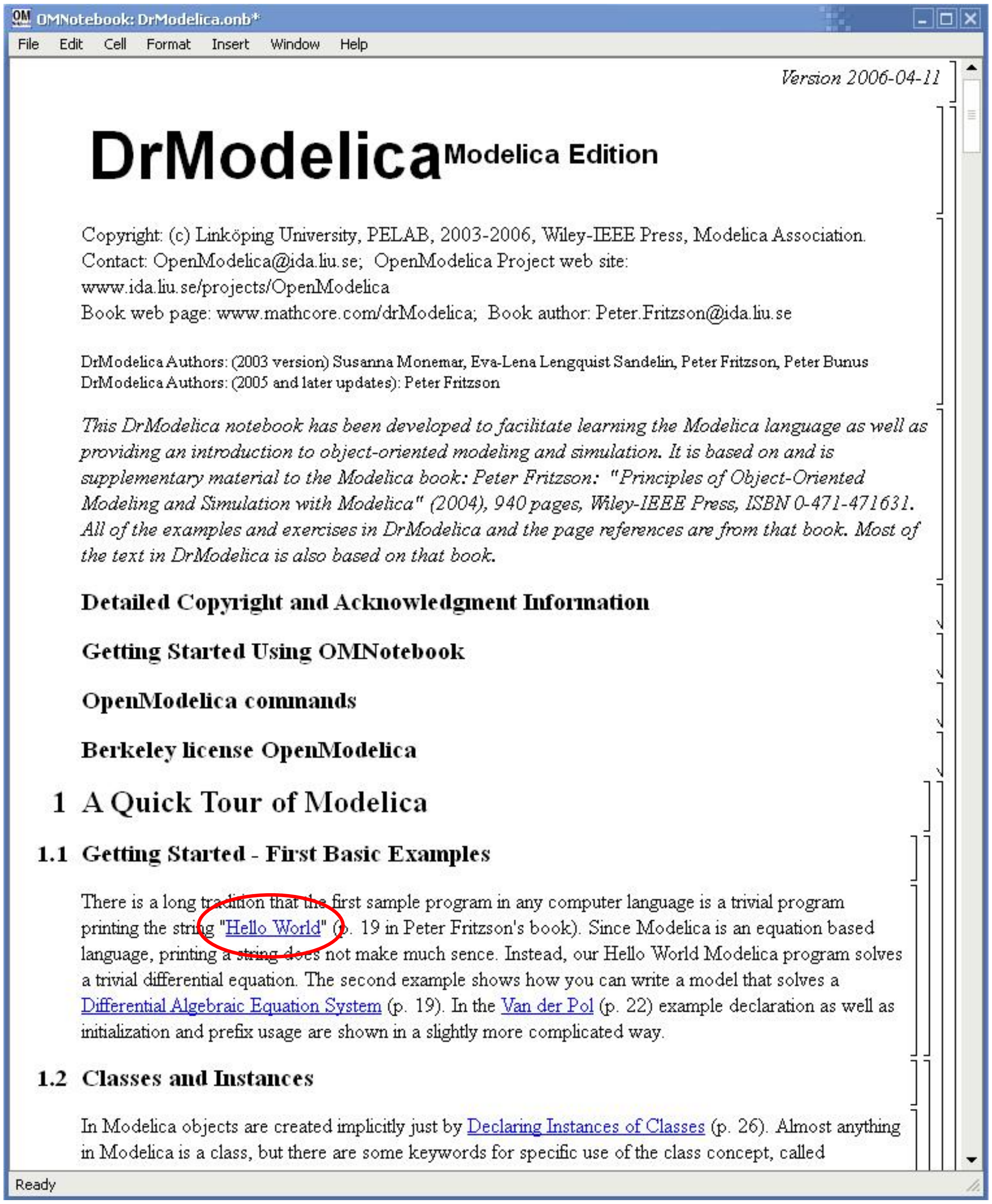
Understanding programs is hard, especially code written by someone else. For educational purposes it is essential to be able to show the source code and to give an explanation of it at the same time.

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 4-2).


OMNotebook: DrModelica.onb*
File Edit Cell Format Insert Window Help
Ready

Version 2006-04-11

DrModelica **Modelica Edition**

Copyright: (c) Linköping University, PELAB, 2003-2006, Wiley-IEEE Press, Modelica Association.
Contact: OpenModelica@ida.liu.se; OpenModelica Project web site:
www.ida.liu.se/projects/OpenModelica
Book web page: www.mathcore.com/drModelica; Book author: Peter.Fritzson@ida.liu.se

DrModelica Authors: (2003 version) Susanna Monemar, Eva-Lena Lengquist Sandelin, Peter Fritzson, Peter Bunus
DrModelica Authors: (2005 and later updates): Peter Fritzson

This DrModelica notebook has been developed to facilitate learning the Modelica language as well as providing an introduction to object-oriented modeling and simulation. It is based on and is supplementary material to the Modelica book: Peter Fritzson: "Principles of Object-Oriented Modeling and Simulation with Modelica" (2004), 940 pages, Wiley-IEEE Press, ISBN 0-471-471631. All of the examples and exercises in DrModelica and the page references are from that book. Most of the text in DrModelica is also based on that book.

Detailed Copyright and Acknowledgment Information

Getting Started Using OMNotebook

OpenModelica commands

Berkeley license OpenModelica

1 A Quick Tour of Modelica

1.1 Getting Started - First Basic Examples

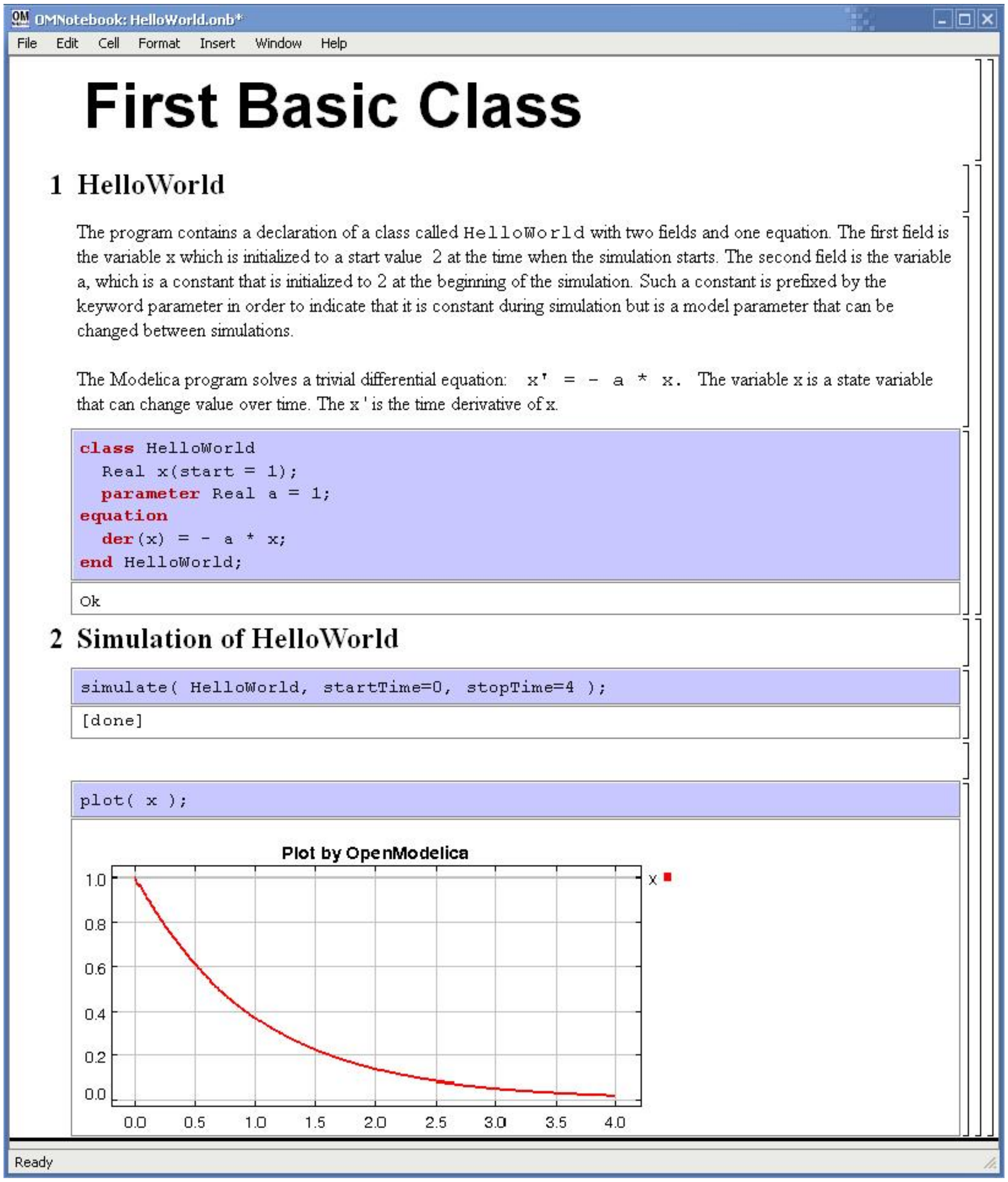
There is a long tradition that the first sample program in any computer language is a trivial program printing the string "Hello World" (p. 19 in Peter Fritzson's book). Since Modelica is an equation based language, printing a string does not make much sence. Instead, our Hello World Modelica program solves a trivial differential equation. The second example shows how you can write a model that solves a Differential Algebraic Equation System (p. 19). In the Van der Pol (p. 22) example declaration as well as initialization and prefix usage are shown in a slightly more complicated way.

1.2 Classes and Instances

In Modelica objects are created implicitly just by Declaring Instances of Classes (p. 26). Almost anything in Modelica is a class, but there are some keywords for specific use of the class concept, called

Figure 4-2. The front-page notebook of the OMNotebook version of the DrModelica tutoring system.

In each chapter of DrModelica the user is presented a short summary of the corresponding chapter of the book "Principles of Object-Oriented Modeling and Simulation with Modelica 2.1" by Peter Fritzson. The summary introduces some *keywords*, being hyperlinks that will lead the user to other notebooks describing the keywords in detail.


OMNotebook: HelloWorld.onb*
File Edit Cell Format Insert Window Help
Plot by OpenModelica
x
Ready

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);
  parameter Real a = 1;
equation
  der(x) = - a * x;
end HelloWorld;
```

```
Ok
```

2 Simulation of HelloWorld

```
simulate( HelloWorld, startTime=0, stopTime=4 );
```

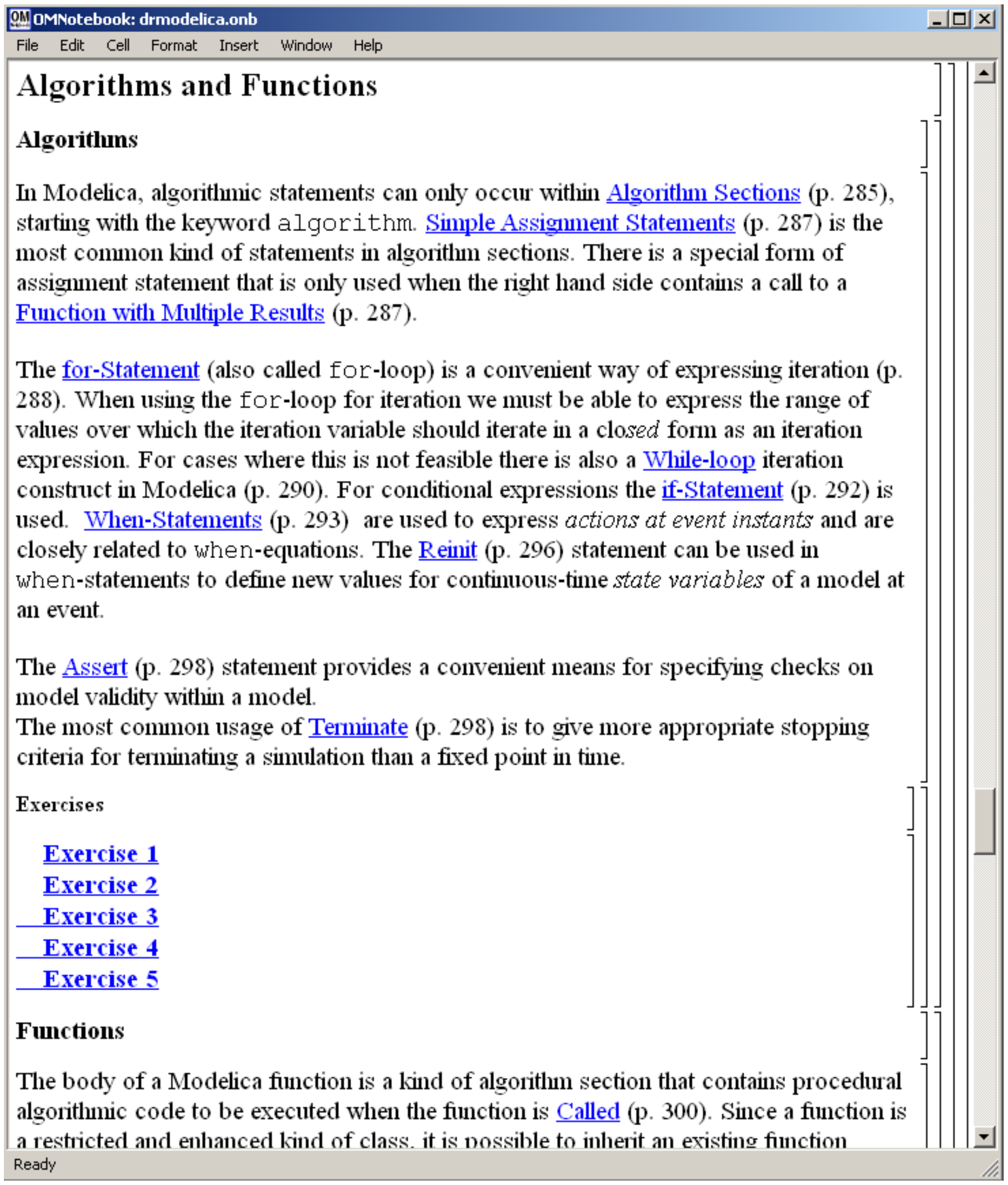
```
[done]
```

```
plot( x );
```

Figure 4-3. The **`HelloWorld`** class simulated and plotted using the OMNotebook version of DrModelica.

Now, let us consider that the link "*HelloWorld*" in DrModelica Section is clicked by the user. The new HelloWorld notebook (see Figure 4-3), 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.


OMNotebook: drmodelica.onb
File Edit Cell Format Insert Window Help
Ready

Algorithms and Functions

Algorithms

In Modelica, algorithmic statements can only occur within Algorithm Sections (p. 285), starting with the keyword `algorithm`. Simple Assignment Statements (p. 287) is the most common kind of statements in algorithm sections. There is a special form of assignment statement that is only used when the right hand side contains a call to a Function with Multiple Results (p. 287).

The for-Statement (also called `for`-loop) is a convenient way of expressing iteration (p. 288). When using the `for`-loop for iteration we must be able to express the range of values over which the iteration variable should iterate in a clo*sed* form as an iteration expression. For cases where this is not feasible there is also a While-loop iteration construct in Modelica (p. 290). For conditional expressions the if-Statement (p. 292) is used. When-Statements (p. 293) are used to express *actions at event instants* and are closely related to `when`-equations. The Reinit (p. 296) statement can be used in `when`-statements to define new values for continuous-time *state variables* of a model at an event.

The Assert (p. 298) statement provides a convenient means for specifying checks on model validity within a model.
The most common usage of Terminate (p. 298) is to give more appropriate stopping criteria for terminating a simulation than a fixed point in time.

Exercises

Exercise 1
Exercise 2
Exercise 3
Exercise 4
Exercise 5

Functions

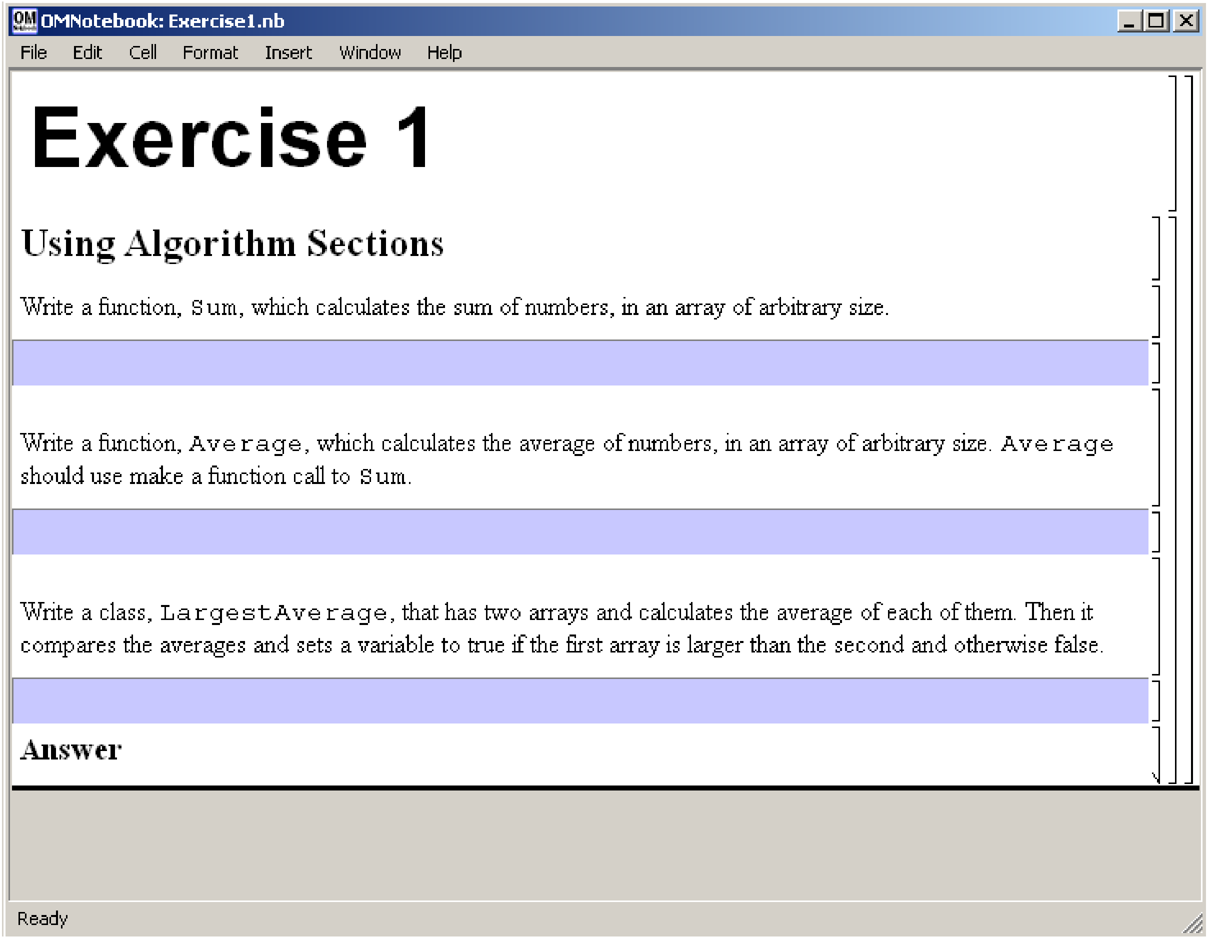
The body of a Modelica function is a kind of algorithm section that contains procedural algorithmic code to be executed when the function is Called (p. 300). Since a function is a restricted and enhanced kind of class, it is possible to inherit an existing function

Figure 4-4. DrModelica Chapter on Algorithms and Functions in the main page of the OMNotebook version of DrModelica.

When a class has been successfully evaluated the user can simulate and plot the result, as previously depicted in Figure 4-3 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 4-4 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

Using Algorithm Sections

Write a function, `Sum`, which calculates the sum of numbers, in an array of arbitrary size.

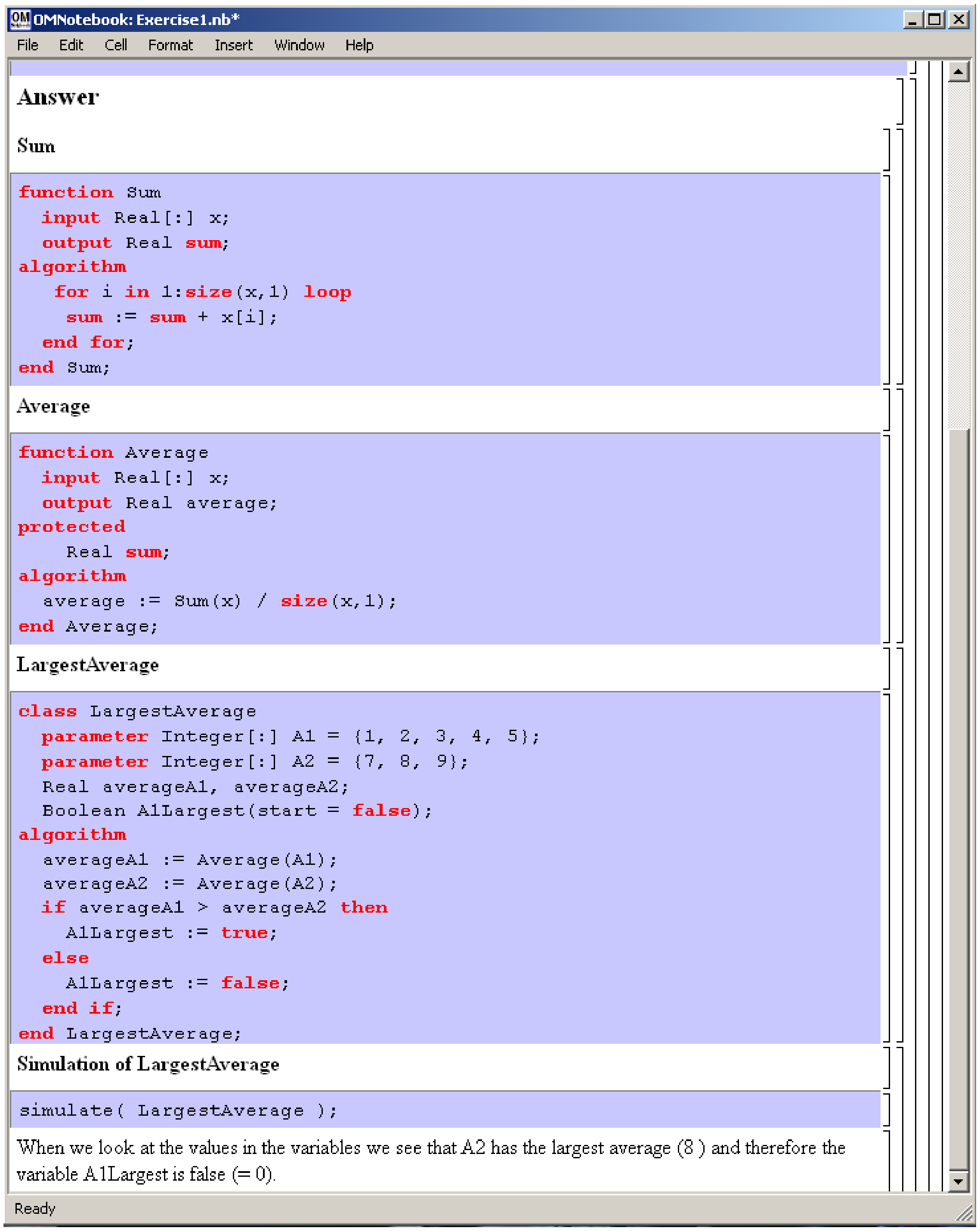
Write a function, `Average`, which calculates the average of numbers, in an array of arbitrary size. `Average` should use make a function call to `Sum`.

Write a class, `LargestAverage`, that has two arrays and calculates the average of each of them. Then it compares the averages and sets a variable to true if the first array is larger than the second and otherwise false.

Answer

Figure 4-5. Exercise 1 in Chapter 9 of DrModelica

Exercise 1 from Chapter 9 is shown in Figure 4-5. 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 visible until the *Answer* section is expanded. The answer is shown in Figure 4-6.


OMNotebook: Exercise1.nb*
File Edit Cell Format Insert Window Help
Ready

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).

Figure 4-6. The answer section to Exercise 1 in Chapter 9 of DrModelica.

4.3 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 `OpenModelica1.9.1/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:

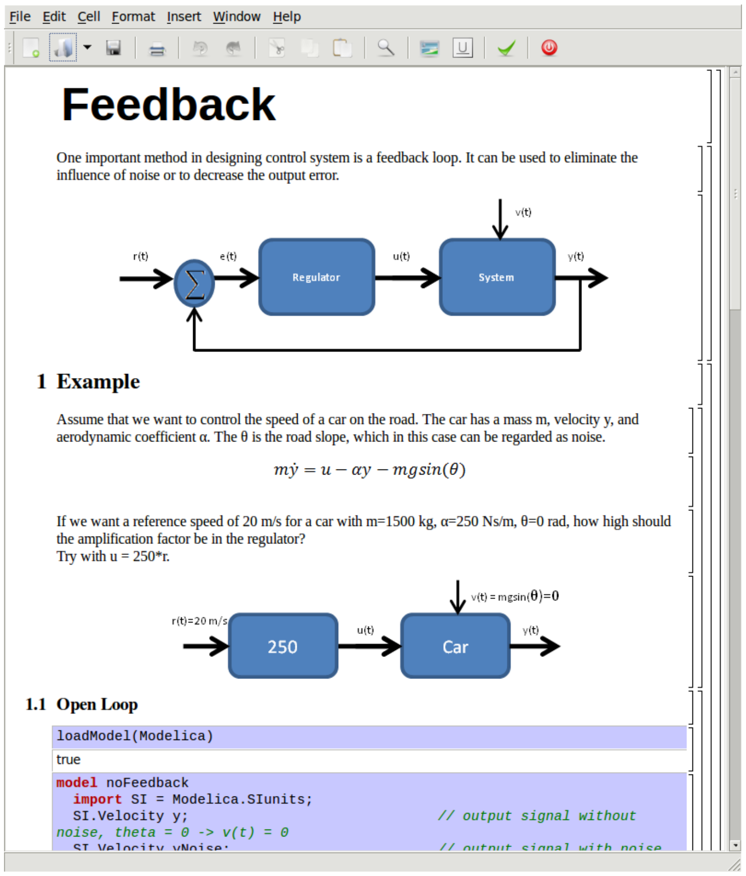
- Getting started
- The control problem in ordinary life
- Feedback loop
- Mathematical modeling
- Transfer function
- Stability
- Example of controlling a DC-motor
- Feedforward compensation
- State-space form
- State observation
- Closed loop control system.
- Reconstructed system
- Linear quadratic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

4.3.1 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 4-7, 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.



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.

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

If we want a reference speed of 20 m/s for a car with m=1500 kg, α=250 Ns/m, θ=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
  SI.Velocity yNoise;                     // output signal with noise
```

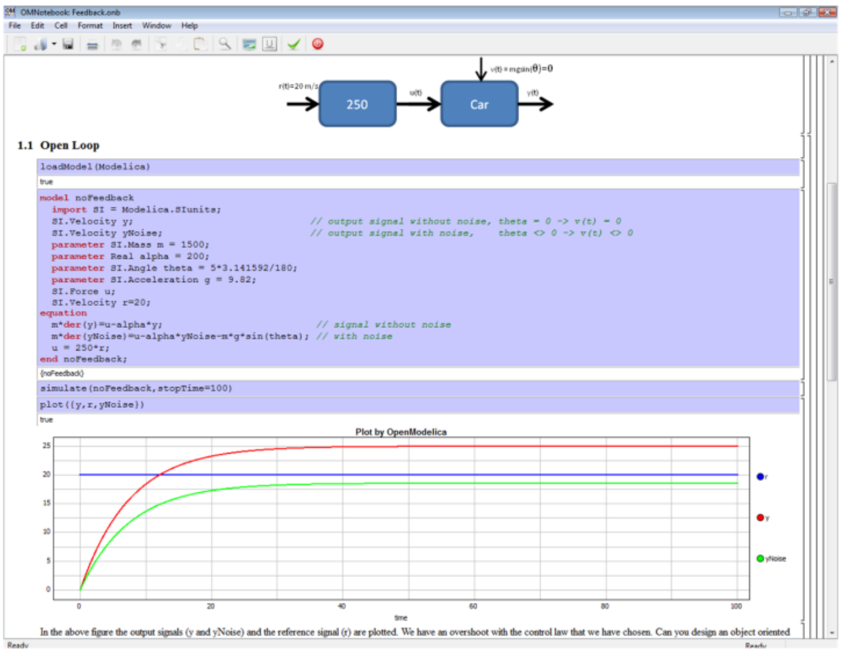
Figure 4-7. Feedback loop

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

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

```
model NoFeedback
   import SI = Modelica.SIunits;
   SI.Velocity y          "No noise";
   SI.Velocity yNoise     "With noise";
   parameter SI.Mass m = 1500;
   parameter Real alpha = 200;
   parameter SI. ngle theta = 5*3.14/180;
   parameter SI.Acceleration g = 9.82;
   SI.Force u;
   SI.Velocity r = 20 "Reference signal";
equation
   m*der(y)=u - alpha*y;
   m*der(yNoise)= u - alpha*yNoise -
      m*g*sin(theta);
   u = 250A*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 4-8 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.



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
  SI.Velocity yNoise;                   // output signal with noise,    theta <> 0 -> v(t) <> 0
  parameter SI.Mass m = 1500;
  parameter Real alpha = 200;
  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;
```

{noFeedback}

```
simulate(noFeedback,stopTime=100)
```

```
plot({y,r,yNoise})
```

true

In the above figure the output signals (y and yNoise) and the reference signal (r) are plotted. We have an overshoot with the control law that we have chosen. Can you design an object oriented

Figure 4-8. Open loop control example.

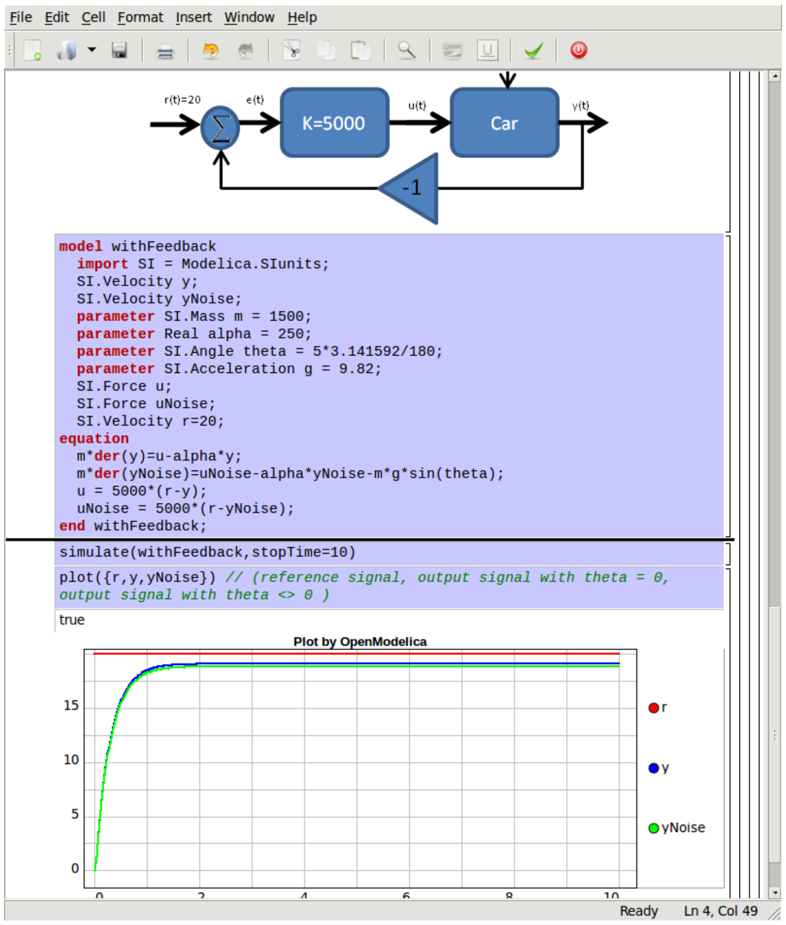
The closed car model with a proportional regulator is shown below:

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

```
model WithFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y        "Output, No noise";
  SI.Velocity yNoise   "Output With noise";
  parameter SI.Mass m = 1500;
  parameter Real alpha = 250;
  parameter SI.Angle theta = 5*3.14/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Force uNoise;
  SI.Velocity r = 20   "Reference signal";
equation
  m*der(y) = u - alpha*y;
  m*der(yNoise) = uNoise - alpha*yNois –
    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 4-9.

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.



```
model withFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y;
  SI.Velocity yNoise;
  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;
```

```
simulate(withFeedback,stopTime=10)
```

```
plot({r,y,yNoise}) // (reference signal, output signal with theta = 0,
output signal with theta <> 0 )
```

```
true
```

Figure 4-9. Closed loop control example.

4.3.2 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 4-10.

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

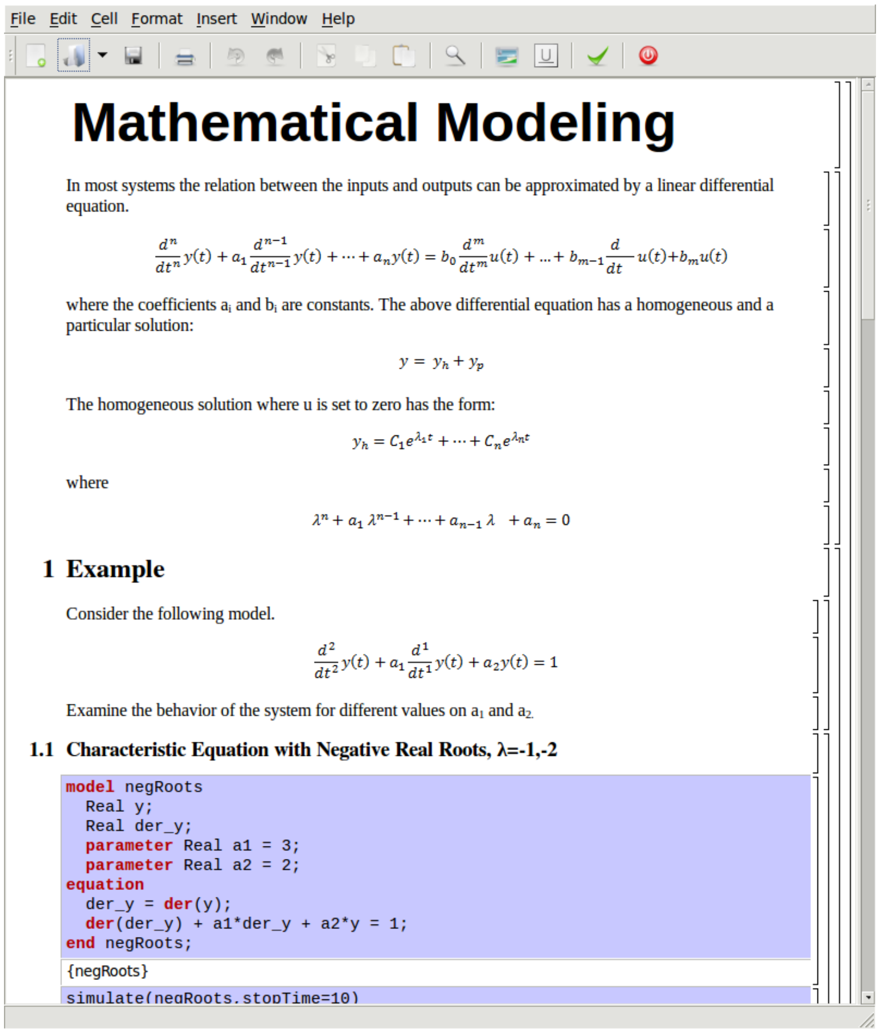
Now let us examine a second order system:

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

```
model NegRoots
  Real y;
  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 4-11 and Figure 4-12.



Mathematical Modeling

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

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

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:

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

where

$$\lambda^n + a_1\,\lambda^{n-1} + \cdots + a_{n-1}\,\lambda \;\; + a_n = 0$$

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, λ=-1,-2

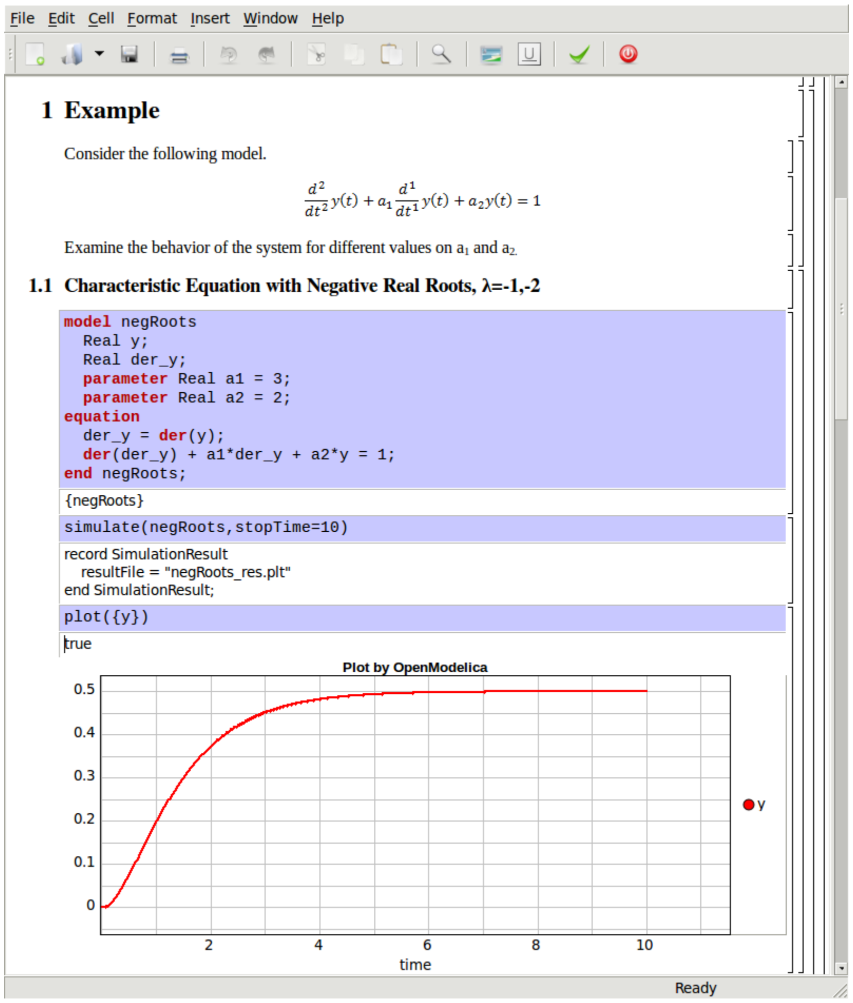
```
model negRoots
  Real y;
  Real der_y;
  parameter Real a1 = 3;
  parameter Real a2 = 2;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end negRoots;
```

```
{negRoots}
```

```
simulate(negRoots,stopTime=10)
```

Figure 4-10. Mathematical modeling with characteristic equation.

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 4-11.



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, λ=-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)
```

```
record SimulationResult
   resultFile = "negRoots_res.plt"
end SimulationResult;
```

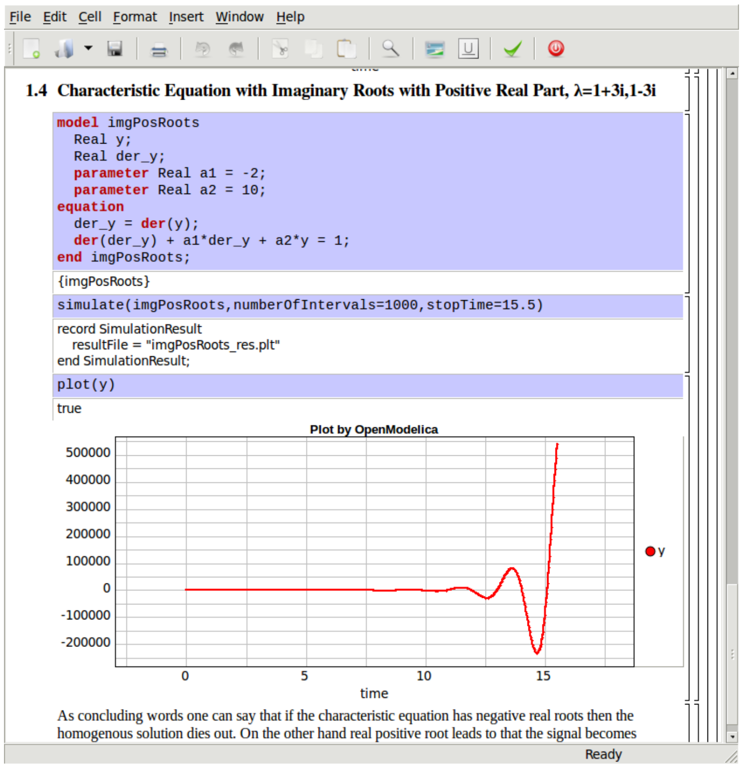
```
plot({y})
```

```
true
```

Figure 4-11. Characteristic eq. with real negative roots.

The importance of the sign of the roots in the characteristic equation is illustrated in Figure 4-11 and Figure 4-12, e.g., a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

```
model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;
```



1.4 Characteristic Equation with Imaginary Roots with Positive Real Part, λ=1+3i,1-3i

```
model imgPosRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end imgPosRoots;
```

```
{imgPosRoots}
```

```
simulate(imgPosRoots,numberOfIntervals=1000,stopTime=15.5)
```

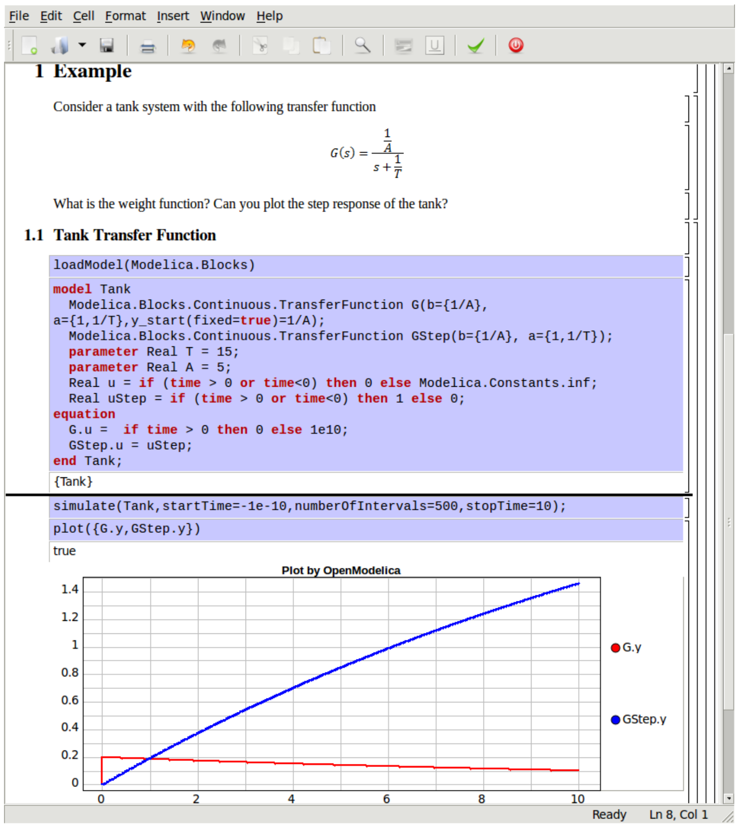
```
record SimulationResult
   resultFile = "imgPosRoots_res.plt"
end SimulationResult;
```

```
plot(y)
```

```
true
```

As concluding words one can say that if the characteristic equation has negative real roots then the homogenous solution dies out. On the other hand real positive root leads to that the signal becomes

Figure 4-12. Characteristic eq. with positive imaginary roots.



1 Example

Consider a tank system with the following transfer function

$$G(s) = \frac{\frac{1}{A}}{s + \frac{1}{T}}$$

What is the weight function? Can you plot the step response of the tank?

1.1 Tank Transfer Function

```
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}
```

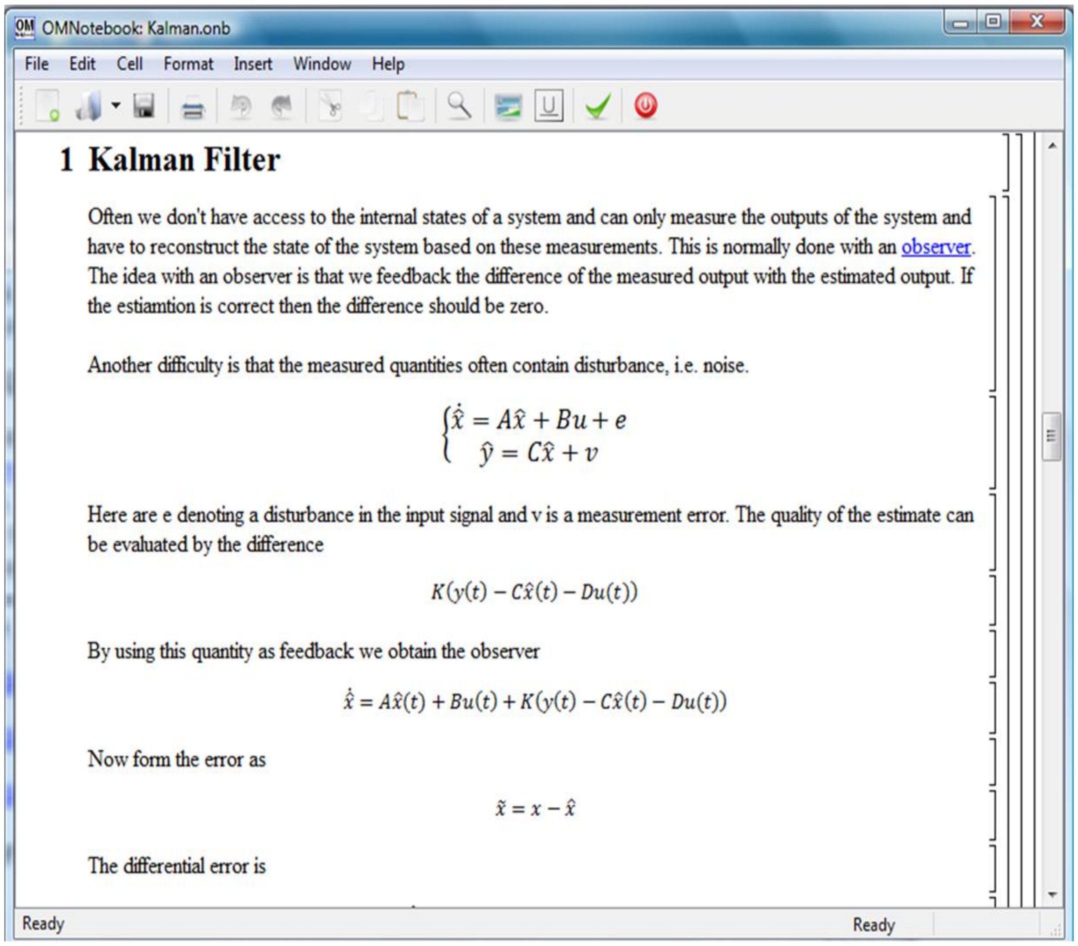
```
simulate(Tank,startTime=-1e-10,numberOfIntervals=500,stopTime=10);
```

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

```
true
```

Figure 4-13. Step and pulse (weight function) response.

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



1 Kalman Filter

Often we don't have access to the internal states of a system and can only measure the outputs of the system and have to reconstruct the state of the system based on these measurements. This is normally done with an observer. The idea with an observer is that we feedback the difference of the measured output with the estimated output. If the estiamtion is correct then the difference should be zero.

Another difficulty is that the measured quantities often contain disturbance, i.e. noise.

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + Bu + e \\ \hat{y} = C\hat{x} + v \end{cases}$$

Here are e denoting a disturbance in the input signal and v is a measurement error. The quality of the estimate can be evaluated by the difference

$$K(y(t) - C\hat{x}(t) - Du(t))$$

By using this quantity as feedback we obtain the observer

$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t) - Du(t))$$

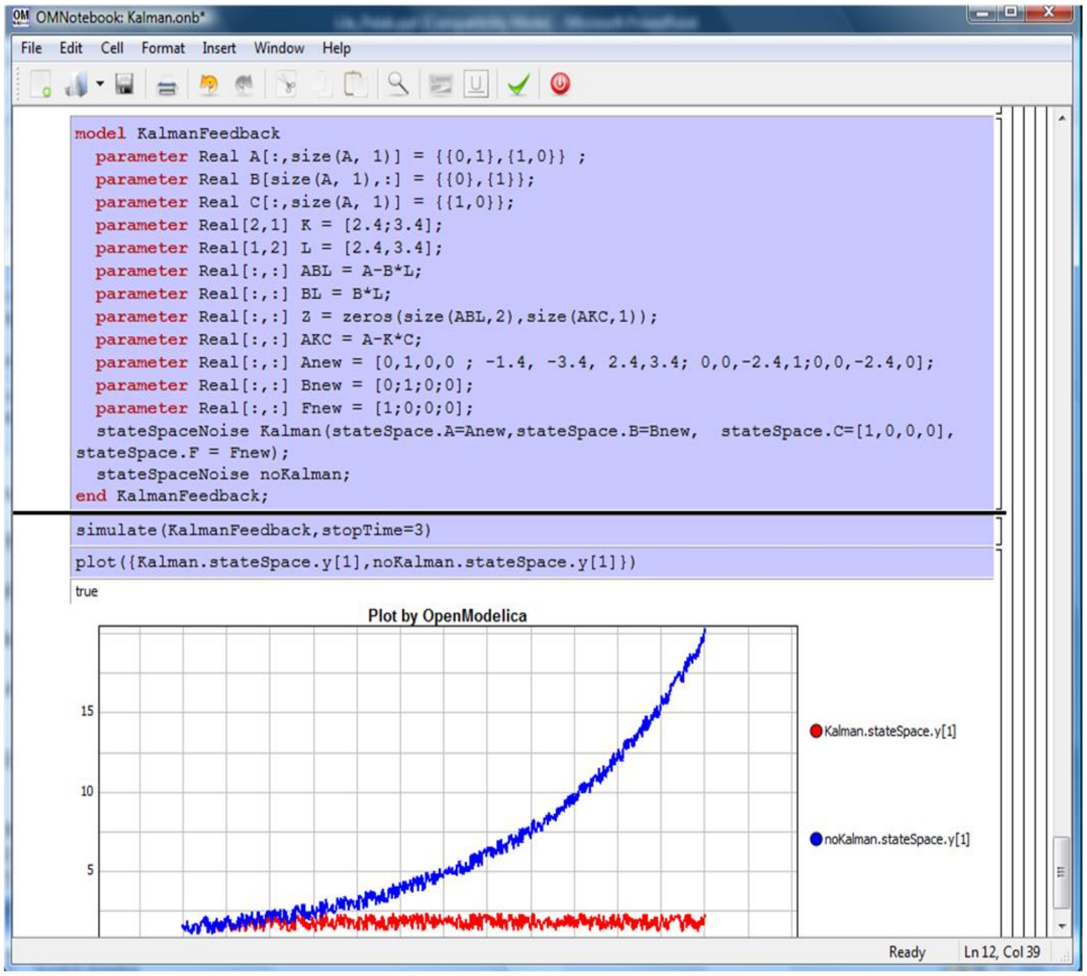
Now form the error as

$$\tilde{x} = x - \hat{x}$$

The differential error is

Figure 4-14. Theory background about Kalman filter.

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.



```
model KalmanFeedback
  parameter Real A[:,size(A, 1)] = {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] = {{0},{1}};
  parameter Real C[:,size(A, 1)] = {{1,0}};
  parameter Real[2,1] K = [2.4;3.4];
  parameter Real[1,2] L = [2.4,3.4];
  parameter Real[:,:] ABL = A-B*L;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z = zeros(size(ABL,2),size(AKC,1));
  parameter Real[:,:] AKC = A-K*C;
  parameter Real[:,:] Anew = [0,1,0,0 ; -1.4, -3.4, 2.4,3.4; 0,0,-2.4,1;0,0,-2.4,0];
  parameter Real[:,:] Bnew = [0;1;0;0];
  parameter Real[:,:] Fnew = [1;0;0;0];
  stateSpaceNoise Kalman(stateSpace.A=Anew,stateSpace.B=Bnew,  stateSpace.C=[1,0,0,0],
stateSpace.F = Fnew);
  stateSpaceNoise noKalman;
end KalmanFeedback;
```

```
simulate(KalmanFeedback,stopTime=3)
```

```
plot({Kalman.stateSpace.y[1],noKalman.stateSpace.y[1]})
```

```
true
```

Figure 4-15. Comparison of a noisy system with feedback link **i**n DrControl.

4.4 OpenModelica Notebook Commands

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

4.4.1 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 `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.
- *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.

4.4.2 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.
- *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).

4.4.3 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 positioin 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.

4.4.4 File Menu

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

- *Create a new noteboo*k – 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 Crtl+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.

4.4.5 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. Pasteing 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.

4.4.6 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 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* – Spliting a cell is done with the menu item `Cell->Split cell` or the key combination Ctrl+Shift+P. The cell is splited 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.

4.4.7 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)

4.4.8 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.

4.4.9 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.

4.4.10 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.

4.4.11 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 several cells* – Several inputcells 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 groupcell is selected all inputcells in that groupcell are evaluated, in the order they are located in the groupcell.
- *Command completion* – Inputcells 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 Ptplot to create a plot. From Ptplot OMNotebook gets an image of the plot and automatically adds that image to the output part of an inputcell. 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 <chapterLevel> 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 choosen 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.

4.5 References

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).

Ingemar Axelsson. OpenModelica Notebook for Interactive Structured Modelica Documents. Final thesis, LITH-IDA-EX–05/080–SE, Linköping University, Linköping, Sweden, October 21, 2005.

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.

Anders Fernström. Extending OMNotebook – An Interactive Notebook for Structured Modelica Documents. Final thesis, LITH-IDA-EX--06/057—SE, Dept. Computer and Information Science, Linköping University, Sweden, September 4, 2006.

Peter Fritzson. Principles of Object Oriented Modeling and Simulation with Modelica 2.1, 940 pages, ISBN 0-471-471631, Wiley-IEEE Press. Feb. 2004.

Knuth, Donald E. Literate Programming. The Computer Journal, NO27(2), pp. 97–111, May 1984.

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.

The Modelica Association. The Modelica Language Specification Version 3.0, Sept 2007. http://www.modelica.org.

Stephen Wolfram. The Mathematica Book. Wolfram Media Inc, 1997.

Chapter 5

Interactive Simulation

In order to offer a user-interactive and time synchronous simulation, OpenModelica has an additional subsystem to fulfill general requirements on such simulations.

This module is part of the simulation runtime core and is called “OpenModelica Interactive” (OMI). OMI will result in an executable simulation application, such as the non interactive simulation. The executable file will be generated by the OMC, which contains the full Modelica model as C/C++ code with all required equations, conditions and different solvers to simulate a whole system or a single system component. This executable file offers a non-interactive and an interactive simulation runtime.

The following are some general functionalities of an interactive simulation runtime:

- The user will be able to stimulate the system during a running system simulation and to observe its’ reaction immediately.
- Simulation runtime behavior will be controllable and adaptable to offer an interaction with a user.
- A user will receive simulation results during a simulation synchronous to the real-time. Since network process time and some other factors like scheduling of processes from the operation system this is not given at any time.
- In order to offer a stable simulation, a runtime will inform a user interface of errors and consequential simulation aborts.
- Simulation results will not under-run or exceed a tolerance compared to a thoroughly reliable value, for a correct simulation.
- Communication between a simulation runtime and a user interface will use a well defined interface and be base on a common technology, in this case network communication.

Note that OMI is available in an easy-to-use way from OMEdit, see Section **Error! Reference source not found.**.

5.1 OpenModelica Interactive

5.1.1 Interactively Changeable Parameters

An important modification/addition to the semantics of the Modelica language during interactive simulation is the fact that parameters are changeable while simulating interactively using OMI. All properties using the prefix “parameter” can be changed during an interactive simulation. The fully qualified name is used as a unique identifier, so a parameter value can be found and changed regardless of its hierarchical position in the model.

5.1.2 OpenModelica Interactive Components description

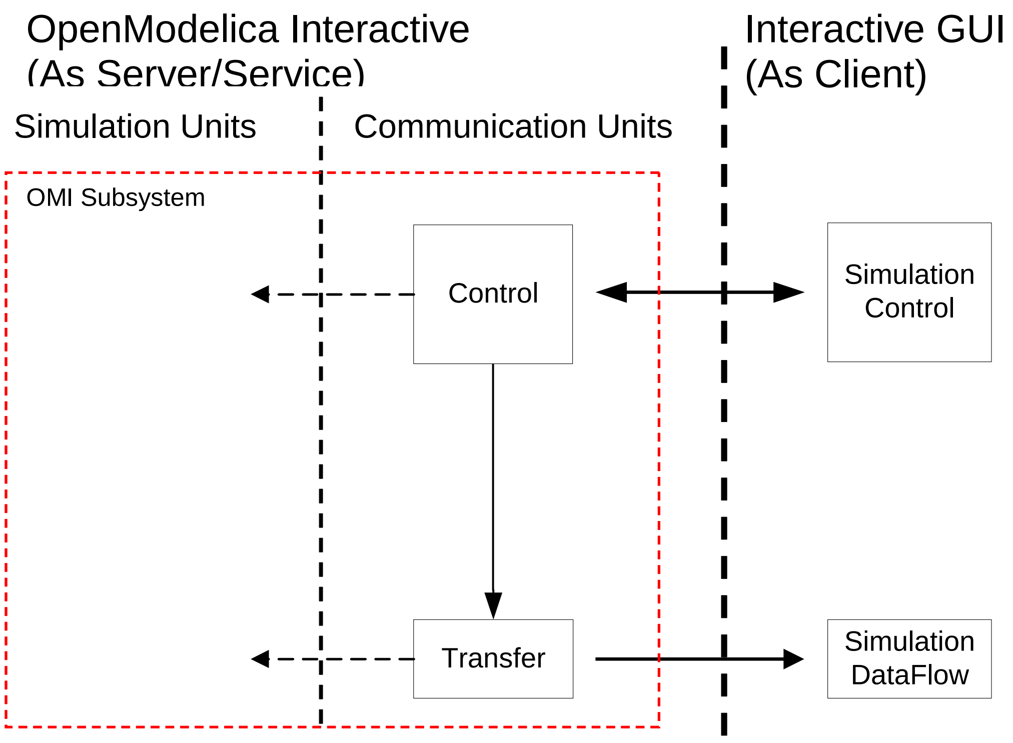


Figure 5-1. OpenModelica interactive communication architecture..

The OpenModelica Interactive subsystem is also separated into different modules, following are important for the user to communicate with:

- Control: The “Control” module is the interface between OMI and a UI. It is implemented as a single thread to support parallel tasks and independent reactivity. As the main controlling and communication instance at simulation initialization phase and while simulation is running it manages simulation properties and also behavior. A client can permanently send operations as messages to the “Control” unit, it can react at any time to feedback from the other internal OMI components and it also sends messages to a client, for example error or status messages.
- Transfer: Similar to a consumer, the “Transfer” thread tries to get simulation results from a result manager and sends them to the UI immediately after starting a simulation. If the communication takes longer than a calculation step, it is also possible to create more than one consumer. The “Transfer” uses a property filter mask containing all property names whose result values are important for the UI. The UI must set this mask using the “setfilter” operation from chapter 2.1.3.2, otherwise the transfer sends only the actual simulation time. This is very useful for increasing the communication speed while sending results to the UI.

5.1.3 Communication Interface

The network communication technology “TCP/IPv4” (later IPv6) will be used to send and receive messages. Each system has its own server and client implementations to receive and send messages respectively. The Control and Transfer are the OMI components which are designated for a communication over TCP/IP.

5.1.4 Network configuration Settings

Name	Description	URL
Control Server	Waits for requests from the UI	By Default, waits for connection on: 127.0.0.1:10501
Control Client	Replies to the UI and sends other synchronization messages to it	By Default, tries to connect on: 127.0.0.1:10500
Transfer Client	Sends simulation results to a UI	By Default, tries to connect on: 127.0.0.1:10502

OMI server and client components: Communication behaviour and configuration by default

Name	Description	URL
Control Client	Requests to the OMI Control Server	By Default, tries to connect on: 127.0.0.1:10501
Control Server	Waits for information from the OMI Control Client	By Default, waits for connection on: 127.0.0.1:10500
Transfer Server	Waits for simulation results from the OMI Transfer Client	By Default, waits for connection on: 127.0.0.1:10502

UI server and client components: Suggested configuration by default

5.1.4.1 Operation Messages

To use messages parsing there is a need to specify a communications protocol.

A string message begins with a specified prefix and ends with a specified suffix.

The prefix describes the request type, for example an operation. Depending on the request type, some additional information and parameters can append on it. The suffix is to check if the message has been received correctly and if the sender has created it correctly. All parts should be separated with “#”.

A sequence number is helpful to manage operation request and reply, a UI has to send a sequence number combined with an operation.

The following are all available message strings between a UI and the OMI system:

Request from UI to Control

UI Request	Description	OMI::Control Reply
start#SEQ#end	Starts or continues the simulation	done#SEQ#end
pause#SEQ#end	Pauses the running simulation	done#SEQ#end
stop#SEQ#end	Stops the running simulation and resets all values to the beginning	done#SEQ#end
shutdown#SEQ#end	Shuts the simulation down	done#SEQ#end
setfilter#SEQ# var1:var2# par1:par2#	Sets the filter for variables and parameters which should send from OMI to the client UI	done#SEQ#end

end		
useindex#SEQ#end	Uses indexes as attribute names. The index will be used at transmitting results to a client. This will cause much less data to transmit. (??Not implemented yet)	done#SEQ#end
setcontrolclienturl#SEQ# ip#port# end	Changes the IP and port of the Control Server. Otherwise the default configuration will be used.	done#SEQ#end
settransferclienturl#SEQ# ip#port# end	Changes the IP and port of the Control Server. Otherwise the default configuration will be used.	done#SEQ#end
changetime#SEQ#Tn#end	Changes the simulation time and goes back to a specific time step	done#SEQ#end
changevalue#SEQ#Tn# par1=2.3:par2=33.3# end	Changes the value of the appended parameters and stets the simulation time back to the point where the user clicked in the UI	done#SEQ#end
error#TYPE#end	Error handling not implemented yet	Error: *

Table 5-1 Available messages from a UI to OMI (Request-Reply)

Messages from Control to UI

OMI::Control	Description	UI
Error: MESSAGE	If an error occurs the OMI::Control generates an error messages and sends the messages with the prefix “Error:” to the UI (not implemented yet)	Up to the UI developers

Table 5-2 Available messages from OMI::Control to UI

Messages from Transfer to UI

OMI::Transfer	Description	UI
result#ID#Tn# var1=Val:var2=Val# par1=Val:par2=Val# end	Sends the simulation result for a time step Tn to the client UI, using the property names as identifier. Maybe a result ID is important to identify the results which are obsolete (not implemented yet).	None
result#ID#Tn# 1=Val:2=Val# 1=Val:2=Val# end	Sends the simulation result for a time step Tn to the client UI, using an index as identifier. This requires a convention about the used index mask. Transfer optimization. NOTE: Operation from UI needed, Mask creation using the standard array index is recommended. Maybe a result ID is important to identify the results which are	None

	obsolete (not implemented yet).	

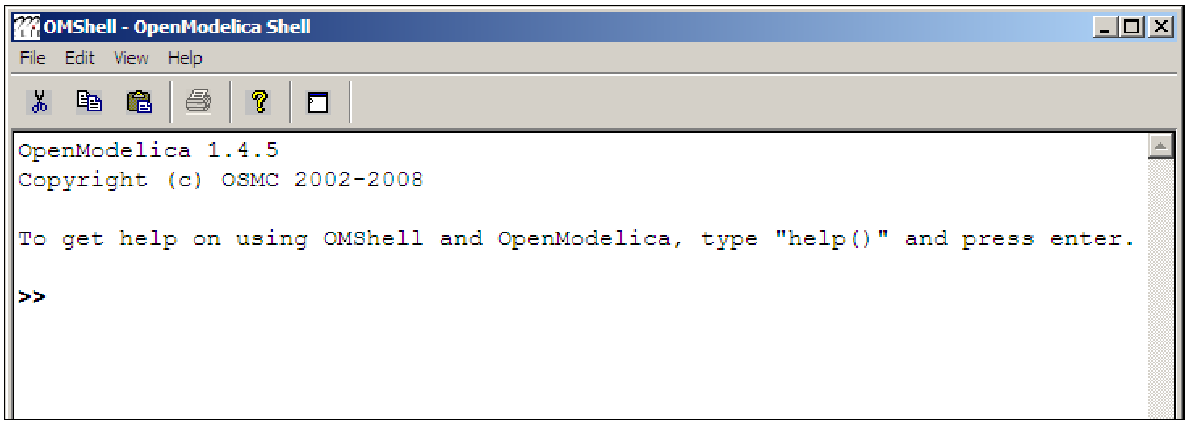
Table 5-3 Available messages from OMI::Transfer to UI

5.1.5 Interactive Simulation general Procedure

Note that OMI is available in an easy-to-use way from OMEdit, see Section **Error! Reference source not found.**, as an alternative to the procedure described below.

5.1.5.1 Initialize an Interactive Simulation Session

Start the OpenModelica Shell or OMNotebook which is available in the start menu as OpenModelica->OpenModelica Shell or OpenModelica->OMNotebook.



```
OpenModelica 1.4.5
Copyright (c) OSMC 2002-2008

To get help on using OMShell and OpenModelica, type "help()" and press enter.

>>
```

1. Load a model or file.
   Optional: You can check if your model or file has been loaded correctly with the operation “list()”
2. Build the model using the operation “buildModel(…)” with the following parameters:
   a) *Model main class name*: Name of the main class of your model.
   b) *numberOfIntervals*: Number of output values in an interval of one second. For Example: “numberOfIntervall=5” means that 5 results will be put out every one second (0s, 0.2s, 0.4s, 0.6s, 0.8s, 1.0s…).
   c) *Note***:** You can use all parameters which are accepted from the operation “buildModel” except the parameters “Start” and “Stop”. These parameters are unnecessary because an interactive simulation always starts at the time “0s” and runs as long as it won’t be stopped or aborted.
3. Execute the created simulation runtime with the parameter “-interactive” and with a port for the control server optionally “-port xxxxx”. After starting the runtime it will wait until a client connects to its control server port. Now you can enter the operations mentioned above.

5.1.6 Interactive Simulation Example

In this chapter we will explain how to simulate a Modelica system interactively. This procedure should be a default step by step procedure for using OMI with a UI.

5.1.6.1 How to get an example Modelica Model

The application sample for Windows is present in C:\OpenModelica1.9.1\share\doc\omc\interactive-simulation. Also read C:\OpenModelica1.9.1\share\doc\omc\interactive-simulation\README.txt.

The source code for the client is in the Subversion repository: trunk/SimulationRuntime/interactive.

An application test is in the Subversion repository here: trunk/testsuite/openmodelica/interactive-simulation.

See here how to get the code: https://www.openmodelica.org/index.php/developer/source-code

5.1.6.2 Create the simulation runtime

We will use an example system based on a demonstration model which is given in the Modelica book by Peter Fritzson [[2], Page 386].

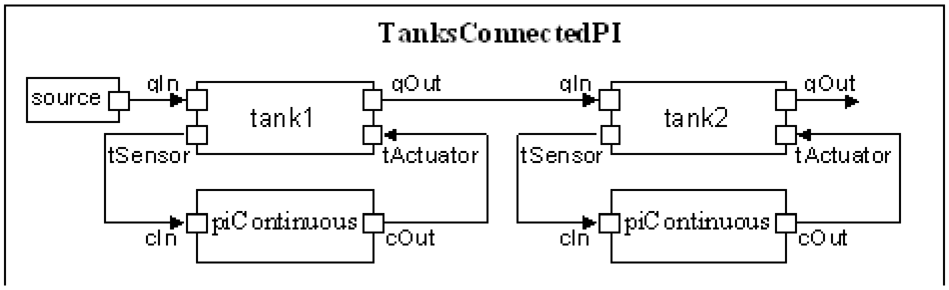
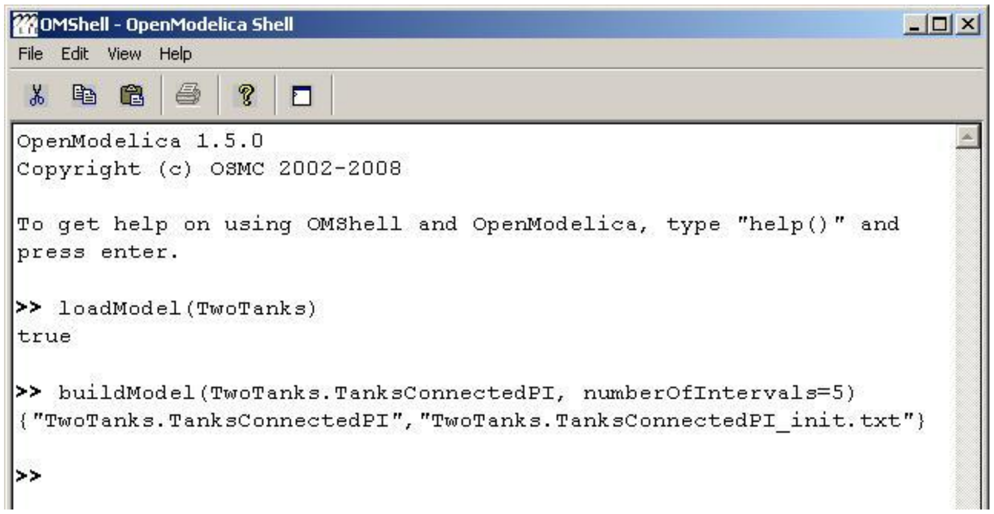


Figure 5-2. TanksConnectedPI structure diagram.

Please follow the steps to create an executable simulation runtime file.

1. Start OMShell “Start->OpenModelica->OpenModelica Shell”
2. Enter the operation **“loadModel(TwoTanks)”**
   NOTE: We assume that the TwoTanks model is in the ModelicaLibary OM installation folder **(…\OpenModelica1.9.1\ModelicaLibrary\TwoTanks)** otherwise please load the file from its location **(…\OpenModelica1.9.1\share\doc\omc\interactive-simulation\*.zip).**
3. Use the “**buildModel**” operation with the following parameters to build the TwoTanks model: **buildModel(TwoTanks.TanksConnectedPI, numberOfIntervals=5)**



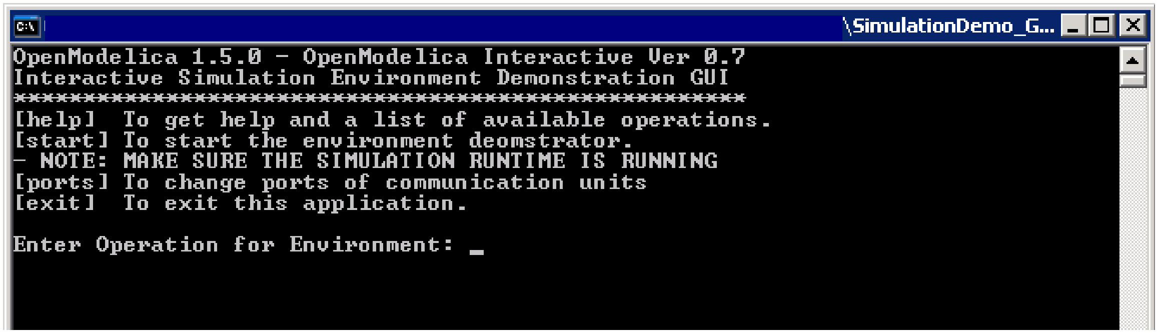
```
OpenModelica 1.5.0
Copyright (c) OSMC 2002-2008

To get help on using OMShell and OpenModelica, type "help()" and
press enter.

>> loadModel(TwoTanks)
true

>> buildModel(TwoTanks.TanksConnectedPI, numberOfIntervals=5)
{"TwoTanks.TanksConnectedPI","TwoTanks.TanksConnectedPI_init.txt"}

>>
```

5.1.6.3 Start an interactive Simulation Session

Start the created simulation runtime it should be located in the "tmp" folder of the OM installation folder (…\OpenModelica1.9.1\tmp\TwoTanks.TanksConnectedPI.exe)

Use the b "`-interactive -port xxxxx`". **NOTE:** If the default port (10501) should be used ignore the parameter "–port". Now the simulation runtime will be waiting until a UI client has been connected on its port.

Start the client: "client.exe".



```
OpenModelica 1.5.0 - OpenModelica Interactive Ver 0.7
Interactive Simulation Environment Demonstration GUI
*******************************************************
[help]  To get help and a list of available operations.
[start] To start the environment deomstrator.
- NOTE: MAKE SURE THE SIMULATION RUNTIME IS RUNNING
[ports] To change ports of communication units
[exit]  To exit this application.

Enter Operation for Environment: _
```

(Deprecated: Now enter "`start`" into the console and wait until the client is successfully connected.)

Enter following operation for the simulation runtime:

```
setcontrolclienturl#1#127.0.0.1#10500#end
settransferclienturl#2#127.0.0.1#10502#end
setfilter#3#tank1.h#source.flowLevel#end
```

Start the simulation with: `start#4#end`

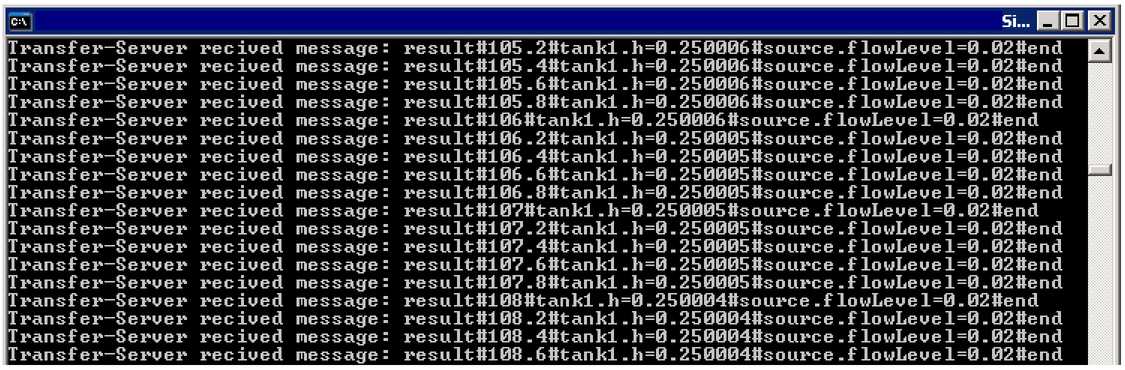
NOTE: After starting the simulation your keyboard entries and the results will be displayed in the same console and you can't see what you are typing. Please pause the simulation first than enter a longer operation string.

Pause the simulation with: `pause#5#end`

Change a Value with: changevalue#6#xx.x#source.flowLevel=0.04#end.

For example if time is higher than 60 and lower than 200 enter →

```
changevalue#6#60.0#source.flowLevel=0.0004#end
```



```
Transfer-Server recived message: result#105.2#tank1.h=0.250006#source.flowLevel=0.02#end
Transfer-Server recived message: result#105.4#tank1.h=0.250006#source.flowLevel=0.02#end
Transfer-Server recived message: result#105.6#tank1.h=0.250006#source.flowLevel=0.02#end
Transfer-Server recived message: result#105.8#tank1.h=0.250006#source.flowLevel=0.02#end
Transfer-Server recived message: result#106#tank1.h=0.250006#source.flowLevel=0.02#end
Transfer-Server recived message: result#106.2#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#106.4#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#106.6#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#106.8#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#107#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#107.2#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#107.4#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#107.6#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#107.8#tank1.h=0.250005#source.flowLevel=0.02#end
Transfer-Server recived message: result#108#tank1.h=0.250004#source.flowLevel=0.02#end
Transfer-Server recived message: result#108.2#tank1.h=0.250004#source.flowLevel=0.02#end
Transfer-Server recived message: result#108.4#tank1.h=0.250004#source.flowLevel=0.02#end
Transfer-Server recived message: result#108.6#tank1.h=0.250004#source.flowLevel=0.02#end
```

Shutdown the simulation runtime and the environment with: `shutdown#7#end`

Chapter 6

Model Import and Export with FMI 1.0

The new standard for model exchange with Functional Mockup Interface (FMI) 1.0 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 model exchange standard is Modelica independent. Import and export works both between different Modelica tools, or between certain non-Modelica tools and Modelica tools. OpenModelica supports FMI 1.0 import as well as FMI 1.0 export.

6.1 FMI Import

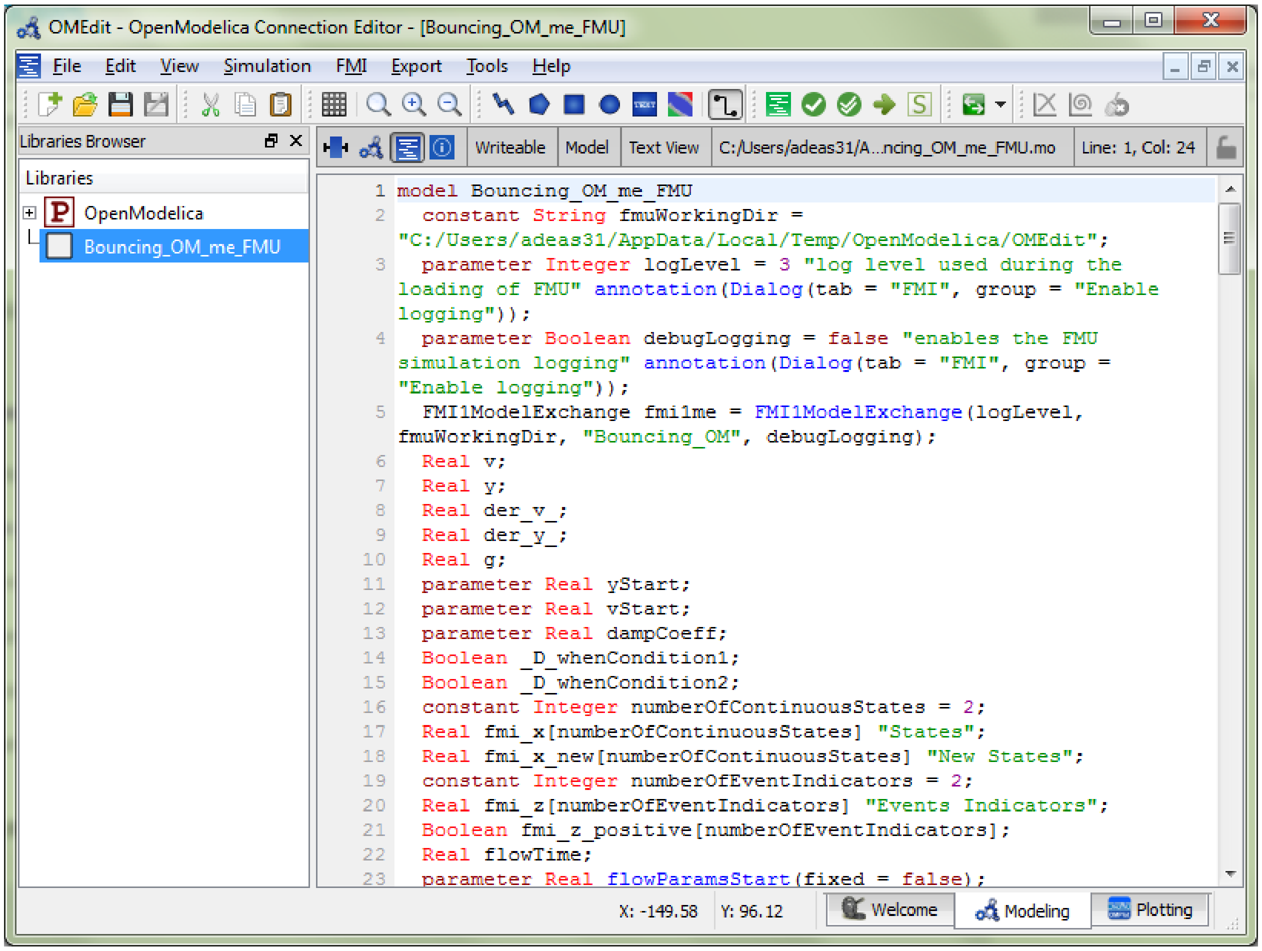
To import the FMU package use the OpenModelica command `importFMU`,

```
function importFMU
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files.
<default> will put the files to current working directory.";
  input Integer loglevel = 3
"loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;loglevel_warning=3;loglevel_info
=4;loglevel_verbose=5;loglevel_debug=6";
  input Boolean fullPath = false "When true the full output path is returned otherwise
only the file name.";
  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 and FMI for Co-Simulation 1.0 stand-alone. The support for FMI Co-Simulation 1.0 tool coupling 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.


OMEdit - OpenModelica Connection Editor - [Bouncing_OM_me_FMU]
File Edit View Simulation FMI Export Tools Help
Libraries Browser
Libraries
OpenModelica
Bouncing_OM_me_FMU
Writeable
Model
Text View
C:/Users/adeas31/A...ncing_OM_me_FMU.mo
Line: 1, Col: 24
X: -149.58
Y: 96.12
Welcome
Modeling
Plotting

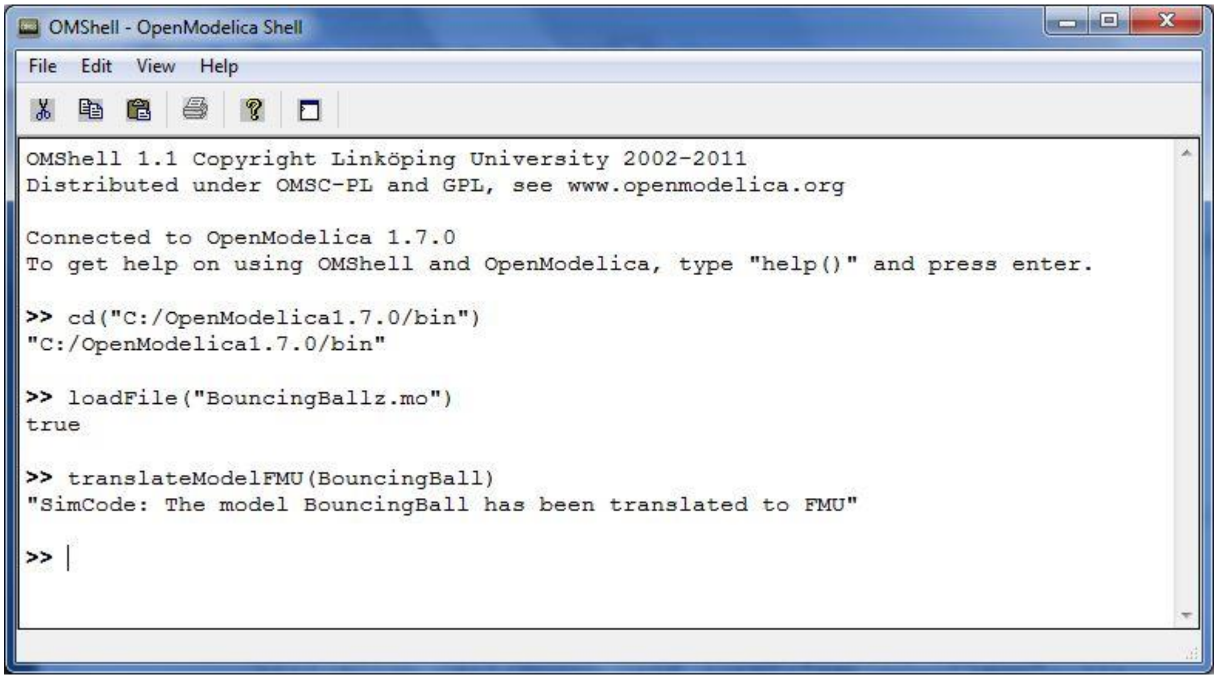
```
model Bouncing_OM_me_FMU
  constant String fmuWorkingDir =
"C:/Users/adeas31/AppData/Local/Temp/OpenModelica/OMEdit";
  parameter Integer logLevel = 3 "log level used during the
loading of FMU" annotation(Dialog(tab = "FMI", group = "Enable
logging"));
  parameter Boolean debugLogging = false "enables the FMU
simulation logging" annotation(Dialog(tab = "FMI", group =
"Enable logging"));
  FMI1ModelExchange fmi1me = FMI1ModelExchange(logLevel,
fmuWorkingDir, "Bouncing_OM", debugLogging);
  Real v;
  Real y;
  Real der_v_;
  Real der_y_;
  Real g;
  parameter Real yStart;
  parameter Real vStart;
  parameter Real dampCoeff;
  Boolean _D_whenCondition1;
  Boolean _D_whenCondition2;
  constant Integer numberOfContinuousStates = 2;
  Real fmi_x[numberOfContinuousStates] "States";
  Real fmi_x_new[numberOfContinuousStates] "New States";
  constant Integer numberOfEventIndicators = 2;
  Real fmi_z[numberOfEventIndicators] "Events Indicators";
  Boolean fmi_z_positive[numberOfEventIndicators];
  Real flowTime;
  parameter Real flowParamsStart(fixed = false);
```

Figure 6-1: Example of FMU Import in OpenModelica where a bouncing ball model is imported.

6.2 FMI Export

To export the FMU use the OpenModelica command `translateModelFMU(ModelName)` from command line interface, OMShell, 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.

After the command execution is complete you will see that a file `ModelName.fmu` has been created. As depicted in Figure 6-2, we first changed the current directory to `C:/OpenModelica1.7.0/bin` , then we loaded a Modelica file with `BouncingBall` example model and finally created an FMU for it using the `translateModelFMU` call.



```
OMShell 1.1 Copyright Linköping University 2002-2011
Distributed under OMSC-PL and GPL, see www.openmodelica.org

Connected to OpenModelica 1.7.0
To get help on using OMShell and OpenModelica, type "help()" and press enter.

>> cd("C:/OpenModelica1.7.0/bin")
"C:/OpenModelica1.7.0/bin"

>> loadFile("BouncingBallz.mo")
true

>> translateModelFMU(BouncingBall)
"SimCode: The model BouncingBall has been translated to FMU"

>> 
```

Figure 6-2: OMShell screenshot for creating an FMU

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.

Chapter 7

Optimization with OpenModelica

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

- Builtin dynamic optimization with OpenModelica and IpOpt using dynamic optimization, Section 7.1 This is the recommended way of performing dynamic optimization with OpenModelica.
- Dynamic optimization with OpenModelica by automatic export of the problem to CasADi, Section 7.2. Use this if you want to employ the CasADi tool for dynamic optimization.
- Classical parameter sweep based design optimization, Section 7.3. Use this if you have a static optimization problem.

7.1 Builtin Dynamic Optimization with OpenModelica and IpOpt

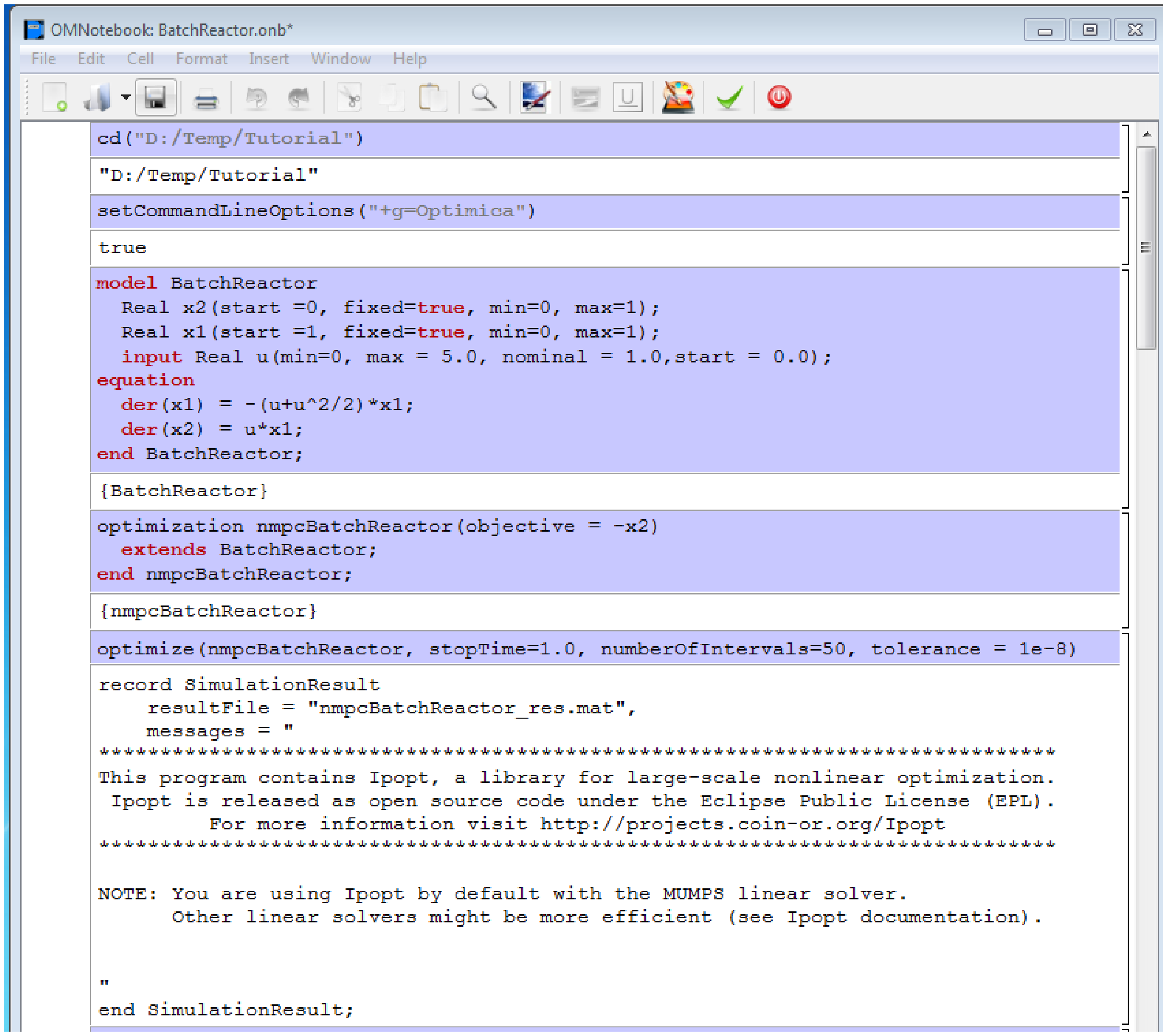
Note: this is a very short preliminary decription which soon will be considerably improved.

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 7.3.

For more detailed information regarding background and methods, see the papers:

- Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, Martin Sivertsson. Parallel Multiple-Shooting and Collocation Optimization with OpenModelica. In Proceedings of the 9th International Modelica Conference (Modelica'2012), Munich, Germany, Sept.3-5, 2012.
- Vitalij Ruge, Willi Braun, Bernhard Bachmann, Andrea Walther and Kshitij Kulshreshtha. Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL–C. In Proceedings of the 10th International Modelica Conference (Modelica'2014), Munich, Germany, March.10-12, 2014.


OMNotebook: BatchReactor.onb*
File Edit Cell Format Insert Window Help

```
cd("D:/Temp/Tutorial")
```

```
"D:/Temp/Tutorial"
```

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

```
true
```

```
model BatchReactor
  Real x2(start =0, fixed=true, min=0, max=1);
  Real x1(start =1, fixed=true, min=0, max=1);
  input Real u(min=0, max = 5.0, nominal = 1.0,start = 0.0);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;
```

```
{BatchReactor}
```

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

```
{nmpcBatchReactor}
```

```
optimize(nmpcBatchReactor, stopTime=1.0, numberOfIntervals=50, tolerance = 1e-8)
```

```
record SimulationResult
    resultFile = "nmpcBatchReactor_res.mat",
    messages = "
******************************************************************************
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
******************************************************************************

NOTE: You are using Ipopt by default with the MUMPS linear solver.
      Other linear solvers might be more efficient (see Ipopt documentation).


"
end SimulationResult;
```

Figure 7-1: OMNotebook screenshot for dynamic optimization.

7.1.1 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.

```
// name: BatchReactor.mos

setCommandLineOptions("+g=Optimica");
getErrorString();

loadFile("BatchReactor.mo");
getErrorString();

optimize(nmpcBatchReactor, numberOfIntervals=16, stopTime=1, tolerance=1e-8);
getErrorString();
```

7.1.2 An Example

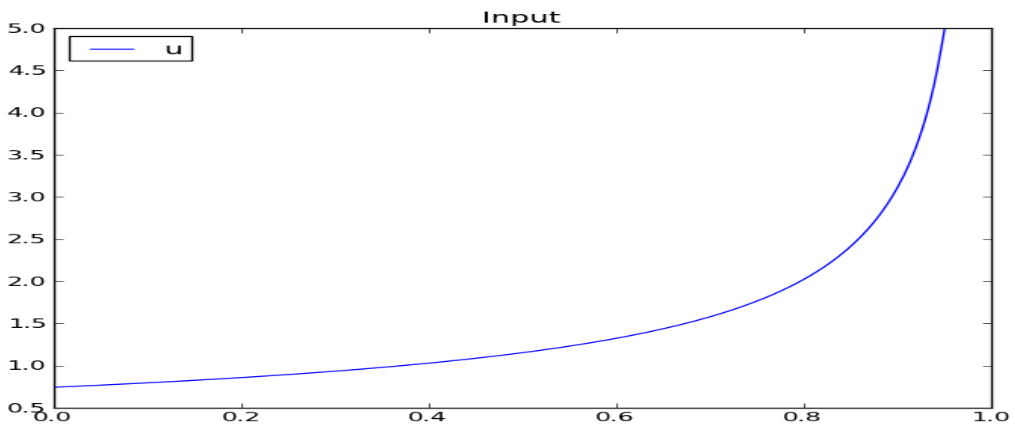
In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by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

```
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;
```

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

Once we have formulated the undelying optimal control problems, we can run the optimization by using OMShell, OMNotebook , MDT, OMEdit or command line terminals similar as described in Figure 7-1.

The control and state trajectories of the optimization results are shown in Figure 7-2.



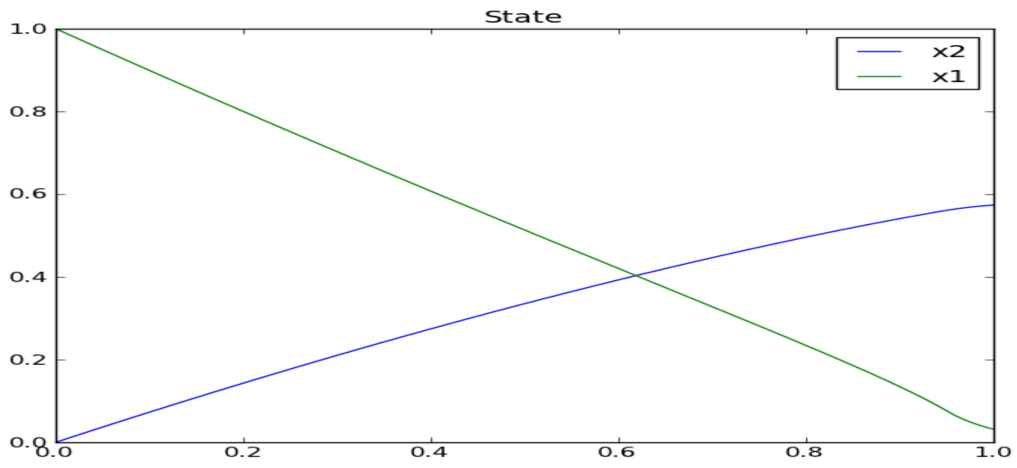


Figure 7-2: Optimization results for Batch Reactor model – state and control variables.

7.1.3 Different Options for the Optimizer IPOPT

Compiler options

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

Run/debug options

`-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

Figure 7-3: Compiler options for IpOpt.

7.2 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.

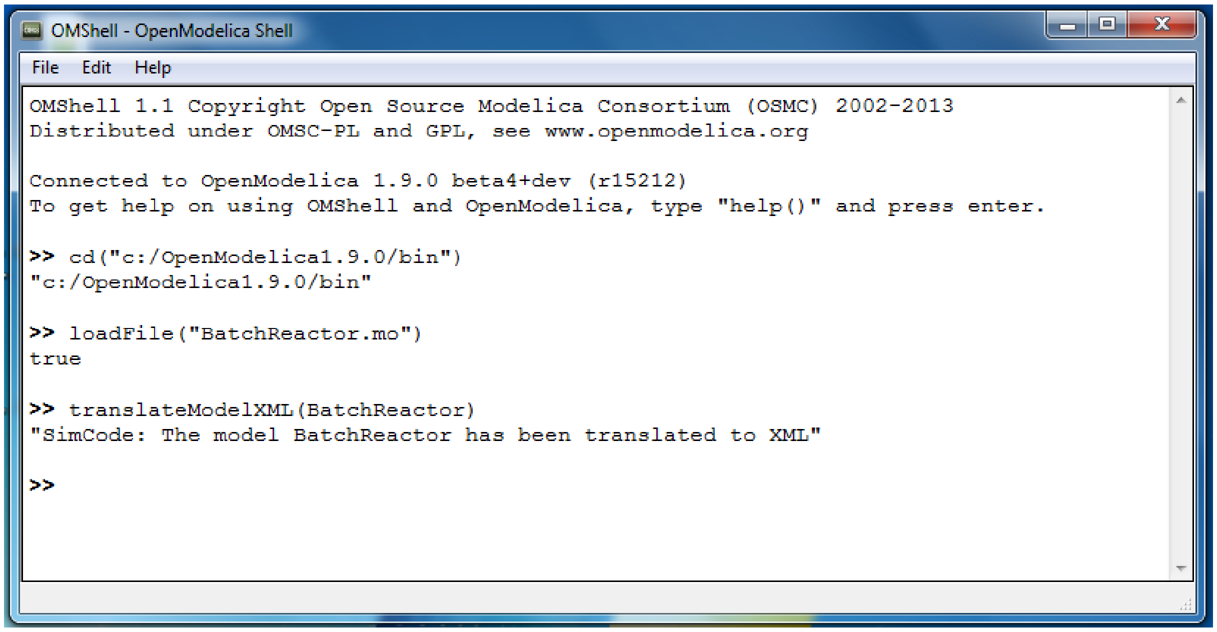
7.2.1 Compiling the Modelica code

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. As depicted in Figure 7-4, we first changed the current directory to `C:/OpenModelica1.9.1/bin`, and then we loaded a Modelica file with BatchReactor example model and finally exported an XML for it using the `translateModelXML` call.

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 `(C:/<%path to modelName .mo file%>)`.
- Go to omc path `(<%path to omc%>/omc)` and write the flag `+s +g=Optimica +simCodeTarget=XML <%your.mo file name%>.mo>`



```
OMShell 1.1 Copyright Open Source Modelica Consortium (OSMC) 2002-2013
Distributed under OMSC-PL and GPL, see www.openmodelica.org

Connected to OpenModelica 1.9.0 beta4+dev (r15212)
To get help on using OMShell and OpenModelica, type "help()" and press enter.

>> cd("c:/OpenModelica1.9.0/bin")
"c:/OpenModelica1.9.0/bin"

>> loadFile("BatchReactor.mo")
true

>> translateModelXML(BatchReactor)
"SimCode: The model BatchReactor has been translated to XML"

>>
```

Figure 7-4: OMShell screenshot for exporting an XML.

7.2.2 An example

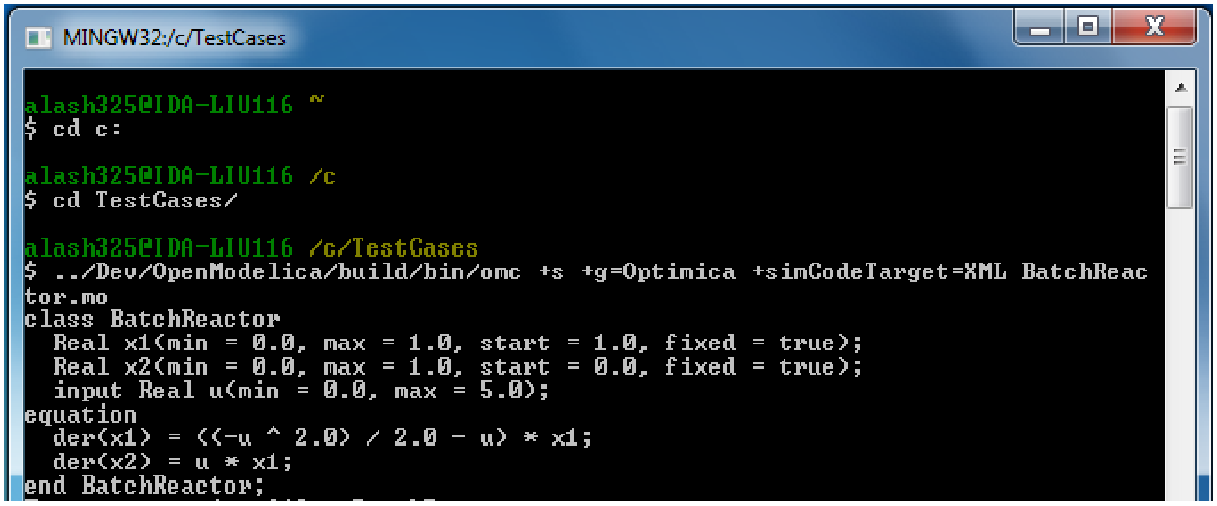
In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by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

```
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;
```

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

One we have formulated the undelying optimal control problems, we can export the XML by using OMShell, OMNotebook , MDT, OMEdit or command line terminals which are described in Section 7.2.4 .

To export XML using terminals as depicted in Figure 7-5, we first changed the current directory to `C:/TestCases`, and run command `../Dev/OpenModleica/build/bin omc +s +g=Optimica +simCodeTarget=XML BatchReactor.mo`. This will generate an XML file under `C:/TestCases` directory named `BatchReactor.xml` shown in Section 7.2.3 that contains a symbolic representation of the optimal control problem and can be inspected in a standard XML editor.


MINGW32:/c/TestCases

```
alash325@IDA-LIU116 ~
$ cd c:

alash325@IDA-LIU116 /c
$ cd TestCases/

alash325@IDA-LIU116 /c/TestCases
$ ../Dev/OpenModelica/build/bin/omc +s +g=Optimica +simCodeTarget=XML BatchReac
tor.mo
class BatchReactor
  Real x1(min = 0.0, max = 1.0, start = 1.0, fixed = true);
  Real x2(min = 0.0, max = 1.0, start = 0.0, fixed = true);
  input Real u(min = 0.0, max = 5.0);
equation
  der(x1) = ((-u ^ 2.0) / 2.0 - u) * x1;
  der(x2) = u * x1;
end BatchReactor;
```

Figure 7-5: Terminal screenshot for exporting an XML.

7.2.3 Generated XML for Example

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenModelicaModelDescription
   xmlns:exp="https://svn.jmodelica.org/trunk/XML/daeExpressions.xsd"
   xmlns:equ="https://svn.jmodelica.org/trunk/XML/daeEquations.xsd"
   xmlns:fun="https://svn.jmodelica.org/trunk/XML/daeFunctions.xsd"
   xmlns:opt="https://svn.jmodelica.org/trunk/XML/daeOptimization.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   fmiVersion="1.0"
   modelName="BatchReactor"
   modelIdentifier="BatchReactor"
   guid="{d06ca497-3a14-4c61-ab0a-ee9f3edfca73}"
   generationDateAndTime="2012-05-18T17:47:35"
   variableNamingConvention="Structured"
   numberOfContinuousStates="2"
   numberOfEventIndicators="0">
   <VendorAnnotations>
       <Tool name="OpenModelica Compiler 1.8.1+ (r11925)">
       </Tool>
   </VendorAnnotations>
   <ModelVariables>
       <ScalarVariable name="finalTime" valueReference="0"
      variability="parameter" causality="internal" alias="noAlias">
           <Real relativeQuantity="false" start="1.0" free="false"
                initialGuess="0.0" />
           <QualifiedName>
       <exp:QualifiedNamePart name="finalTime"/>
           </QualifiedName>
```

```
            <isLinear>true</isLinear>
            <isLinearTimedVariables>
                <TimePoint index="0" isLinear="true"/>
            </isLinearTimedVariables>
            <VariableCategory>independentParameter</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="startTime" valueReference="1"
               variability="parameter" causality="internal" alias="noAlias">
            <Real relativeQuantity="false" start="0.0" free="false"
                  initialGuess="0.0" />
            <QualifiedName>
                 <exp:QualifiedNamePart name="startTime"/>
            </QualifiedName>
            <isLinear>true</isLinear>
            <isLinearTimedVariables>
                <TimePoint index="0" isLinear="true"/>
            </isLinearTimedVariables>
            <VariableCategory>independentParameter</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="x1" valueReference="2" variability="continuous"
             causality="internal" alias="noAlias">
             <Real relativeQuantity="false" min="0.0" max="1.0" start="1.0"
                  fixed="true"  />
             <QualifiedName>
         <exp:QualifiedNamePart name="x1"/>
             </QualifiedName>
             <VariableCategory>state</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="x2" valueReference="3"
              variability="continuous" causality="internal" alias="noAlias">
              <Real relativeQuantity="false" min="0.0" max="1.0" start="0.0"
                    fixed="true"  />
              <QualifiedName>
          <exp:QualifiedNamePart name="x2"/>
              </QualifiedName>
              <VariableCategory>state</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="der(x1)" valueReference="4"
                variability="continuous" causality="internal" alias="noAlias">
     <Real relativeQuantity="false" />
     <QualifiedName>
          <exp:QualifiedNamePart name="x1"/>
     </QualifiedName>
     <VariableCategory>derivative</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="der(x2)"  valueReference="5"
        variability="continuous" causality="internal" alias="noAlias">
     <Real relativeQuantity="false"  />
     <QualifiedName>
              <exp:QualifiedNamePart name="x2"/>
     </QualifiedName>
     <VariableCategory>derivative</VariableCategory>
        </ScalarVariable>
        <ScalarVariable name="u"  valueReference="6"
         variability="continuous" causality="input" alias="noAlias">
     <Real relativeQuantity="false" min="0.0" max="5.0"/>
     <QualifiedName>
```

```
             <exp:QualifiedNamePart name="u"/>
      </QualifiedName>
      <VariableCategory>algebraic</VariableCategory>
         </ScalarVariable>
  </ModelVariables>
  <equ:BindingEquations>
        <equ:BindingEquation>
              <equ:Parameter>
           <exp:QualifiedNamePart name="startTime"/>
      </equ:Parameter>
      <equ:BindingExp>
           <exp:IntegerLiteral>0</exp:IntegerLiteral>
      </equ:BindingExp>
        </equ:BindingEquation>
        <equ:BindingEquation>
      <equ:Parameter>
           <exp:QualifiedNamePart name="finalTime"/>
      </equ:Parameter>
      <equ:BindingExp>
            <exp:IntegerLiteral>1</exp:IntegerLiteral>
      </equ:BindingExp>
        </equ:BindingEquation>
  </equ:BindingEquations>
  <equ:DynamicEquations>
       <equ:Equation>
            <exp:Sub>
        <exp:Der>
             <exp:Identifier>
            <exp:QualifiedNamePart name="x2"/>
             </exp:Identifier>
        </exp:Der>
        <exp:Mul>
             <exp:Identifier>
            <exp:QualifiedNamePart name="u"/>
             </exp:Identifier>
             <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
             </exp:Identifier>
         </exp:Mul>
           </exp:Sub>
       </equ:Equation>
       <equ:Equation>
            <exp:Sub>
        <exp:Der>
             <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
             </exp:Identifier>
        </exp:Der>
        <exp:Mul>
                    <exp:Sub>
            <exp:Div>
                 <exp:Neg>
                           <exp:Pow>
                     <exp:Identifier>
                               <exp:QualifiedNamePart name="u"/>
                     </exp:Identifier>
                           <exp:RealLiteral>2.0</exp:RealLiteral>
```

```
                      </exp:Pow>
                      </exp:Neg>
                      <exp:RealLiteral>2.0</exp:RealLiteral>
            </exp:Div>
                 <exp:Identifier>
                      <exp:QualifiedNamePart name="u"/>
                 </exp:Identifier>
                    </exp:Sub>
                    <exp:Identifier>
                         <exp:QualifiedNamePart name="x1"/>
            </exp:Identifier>
       </exp:Mul>
          </exp:Sub>
     </equ:Equation>
</equ:DynamicEquations>
<equ:InitialEquations>
     <equ:Equation>
          <exp:Sub>
               <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
       </exp:Identifier>
       <exp:RealLiteral>1.0</exp:RealLiteral>
          </exp:Sub>
     </equ:Equation>
     <equ:Equation>
          <exp:Sub>
               <exp:Identifier>
            <exp:QualifiedNamePart name="x2"/>
               </exp:Identifier>
               <exp:RealLiteral>0.0</exp:RealLiteral>
          </exp:Sub>
     </equ:Equation>
</equ:InitialEquations>
<opt:Optimization>
      <opt:ObjectiveFunction>
           <exp:Neg>
                <exp:TimedVariable timePointIndex = "0" >
                     <exp:Identifier>
                          <exp:QualifiedNamePart name="x2"/>
             </exp:Identifier>
                </exp:TimedVariable>
           </exp:Neg>
      </opt:ObjectiveFunction>
      <opt:IntervalStartTime>
    <opt:Value>0.0</opt:Value>
    <opt:Free>false</opt:Free>
    <opt:InitialGuess>0.0</opt:InitialGuess>
      </opt:IntervalStartTime>
      <opt:IntervalFinalTime>
    <opt:Value>1.0</opt:Value>
    <opt:Free>false</opt:Free>
    <opt:InitialGuess>1.0</opt:InitialGuess>
      </opt:IntervalFinalTime>
      <opt:TimePoints>
    <opt:TimePoint index = "0" value = "1.0">
          <opt:QualifiedName>
               <exp:QualifiedNamePart name="x2"/>
          </opt:QualifiedName>
    </opt:TimePoint>
       </opt:TimePoints>
       <opt:Constraints>
       </opt:Constraints>
   </opt:Optimization>
   <fun:FunctionsList>
   </fun:FunctionsList>
</OpenModelicaModelDescription>
```

7.2.4 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 you working directory.

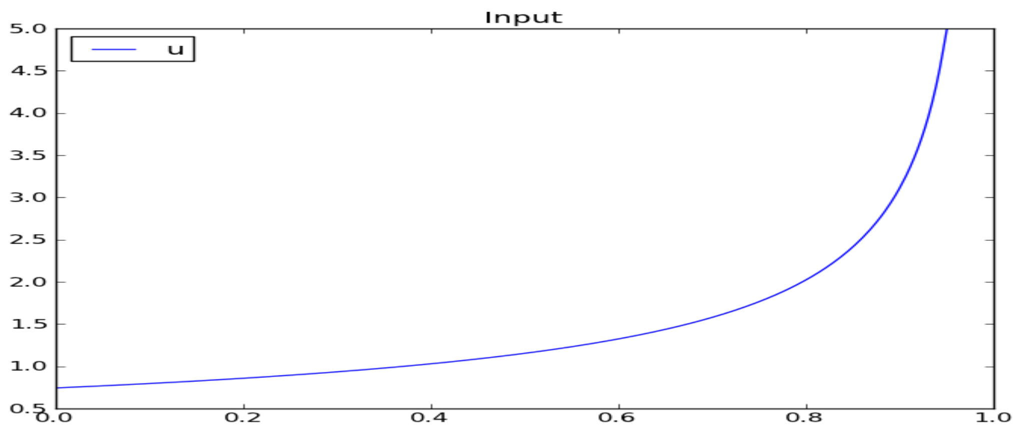
   E.g. `C:\OpenModelica1.9.1\share\casadi\testmodel`
2. Perform compilation and generate the XML file
   a. Go to your working directory

      E.g. `cd C:\OpenModelica1.9.1\share\casadi\testmodel`
   b. 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 in Figure 7-6.



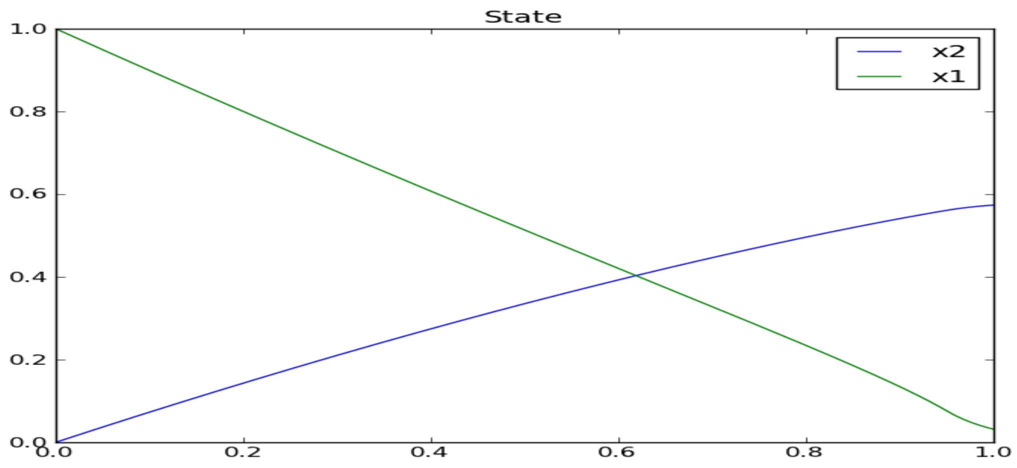


Figure 7-6: Optimization results for Batch Reactor model – state and control variables.

7.3 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 swep, 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.

7.3.1 Preparing the Model

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

7.3.1.1 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 <u>*constant*</u> 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.

7.3.1.2 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.

7.3.1.3 Objectives

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

7.3.2 Set problem in OMOptim

7.3.2.1 Launch OMOptim

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

7.3.2.2 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.

7.3.2.3 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.

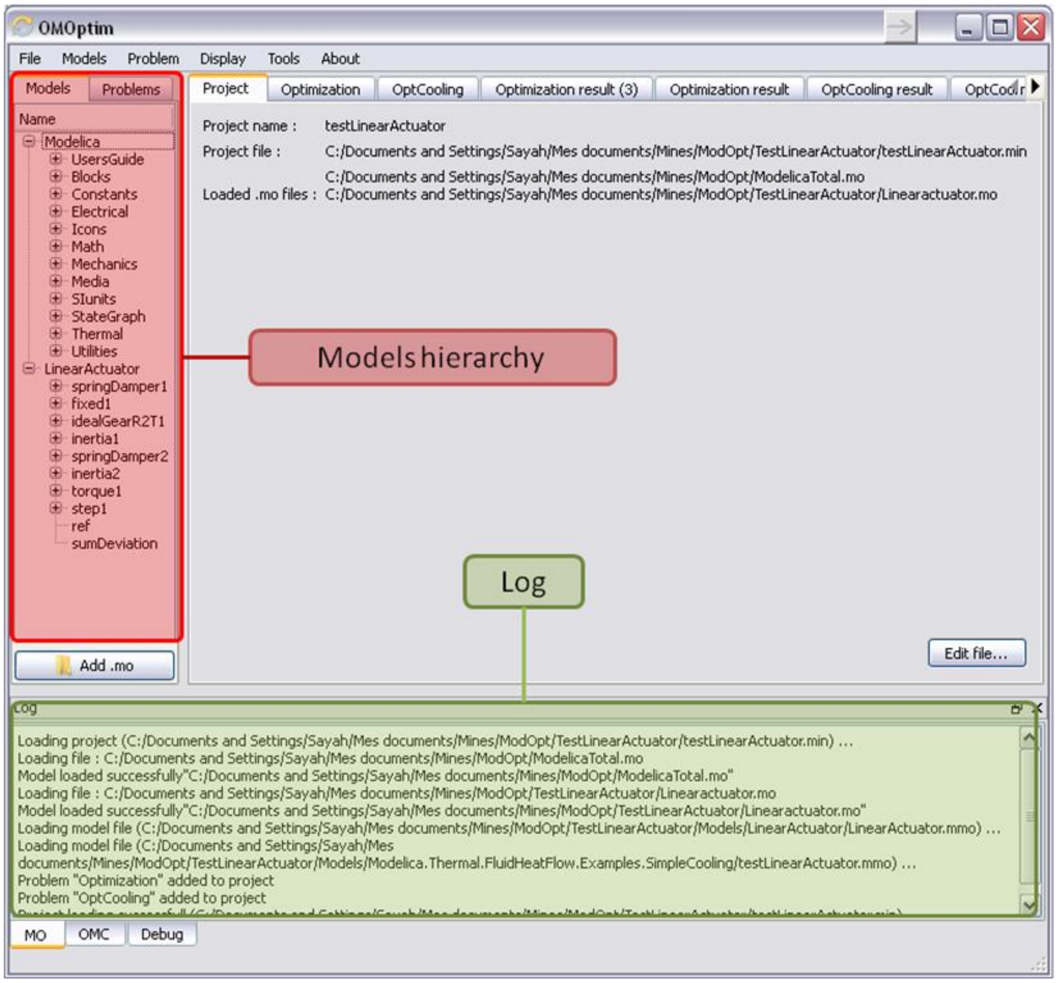
If an error occurs (indicated in log window), this might be because:

- 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, it 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 7-7.



Project name : testLinearActuator
Project file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/testLinearActuator.min
Loaded .mo files : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/ModelicaTotal.mo
C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Linearactuator.mo

```
Loading project (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/testLinearActuator.min) ...
Loading file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/ModelicaTotal.mo
Model loaded successfully"C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/ModelicaTotal.mo"
Loading file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Linearactuator.mo
Model loaded successfully"C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Linearactuator.mo"
Loading model file (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Models/LinearActuator/LinearActuator.mmo) ...
Loading model file (C:/Documents and Settings/Sayah/Mes
documents/Mines/ModOpt/TestLinearActuator/Models/Modelica.Thermal.FluidHeatFlow.Examples.SimpleCooling/testLinearActuator.mmo) ...
Problem "Optimization" added to project
Problem "OptCooling" added to project
```

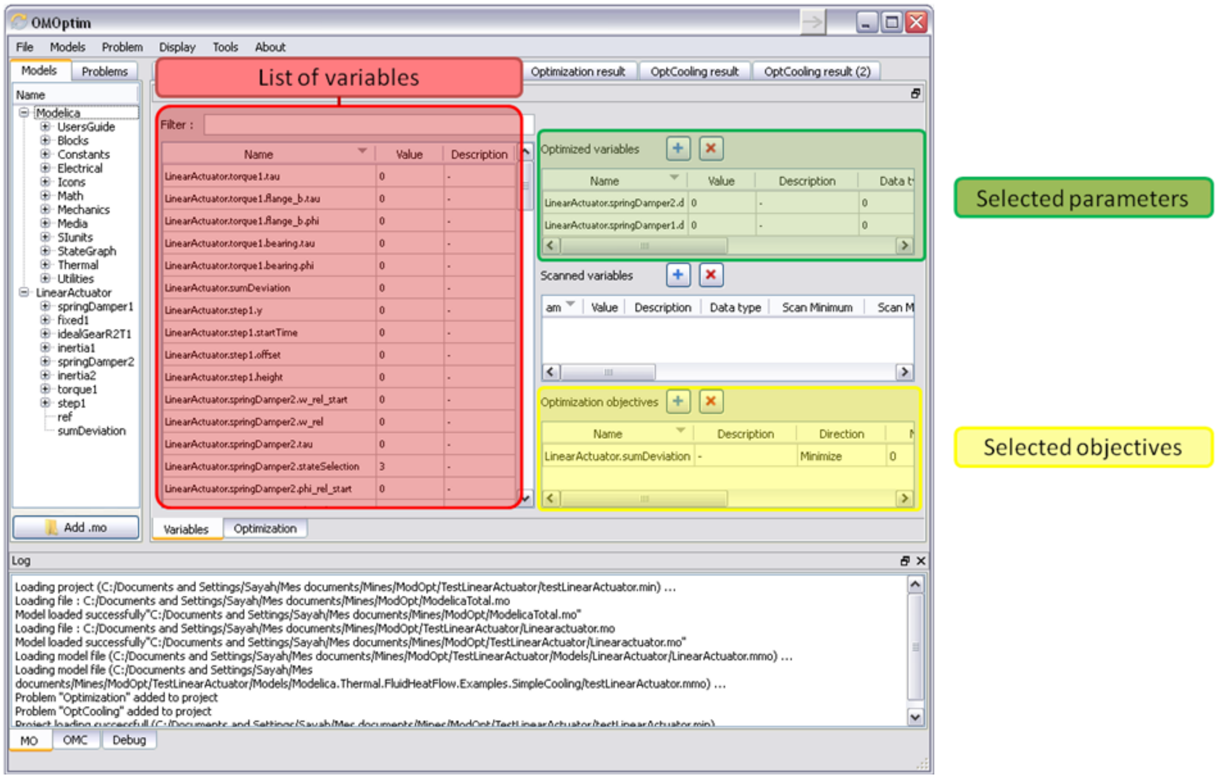
Figure 7-7. OMOptim window after having loaded model.

7.3.2.4 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`.



Name	Value	Description
LinearActuator.torque1.tau	0	-
LinearActuator.torque1.flange_b.tau	0	-
LinearActuator.torque1.flange_b.phi	0	-
LinearActuator.torque1.bearing.tau	0	-
LinearActuator.torque1.bearing.phi	0	-
LinearActuator.sumDeviation	0	-
LinearActuator.step1.y	0	-
LinearActuator.step1.startTime	0	-
LinearActuator.step1.offset	0	-
LinearActuator.step1.height	0	-
LinearActuator.springDamper2.w_rel_start	0	-
LinearActuator.springDamper2.w_rel	0	-
LinearActuator.springDamper2.tau	0	-
LinearActuator.springDamper2.stateSelection	3	-
LinearActuator.springDamper2.phi_rel_start	0	-

Optimized variables

Name	Value	Description	Data t
LinearActuator.springDamper2.d	0	-	0
LinearActuator.springDamper1.d	0	-	0

Scanned variables

am	Value	Description	Data type	Scan Minimum	Scan M

Optimization objectives

Name	Description	Direction	N
LinearActuator.sumDeviation	-	Minimize	0

```
Loading project (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/testLinearActuator.min) ...
Loading file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/ModelicaTotal.mo
Model loaded successfully"C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/ModelicaTotal.mo"
Loading file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Linearactuator.mo
Model loaded successfully"C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Linearactuator.mo"
Loading model file (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/Models/LinearActuator/LinearActuator.mmo) ...
Loading model file (C:/Documents and Settings/Sayah/Mes
documents/Mines/ModOpt/TestLinearActuator/Models/Modelica.Thermal.FluidHeatFlow.Examples.SimpleCooling/testLinearActuator.mmo) ...
Problem "Optimization" added to project
Problem "OptCooling" added to project
Project loading successfull (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestLinearActuator/testLinearActuator.min)
```

Figure 7-8. Forms for defining a new optimization problem.

List of Variables is Empty

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

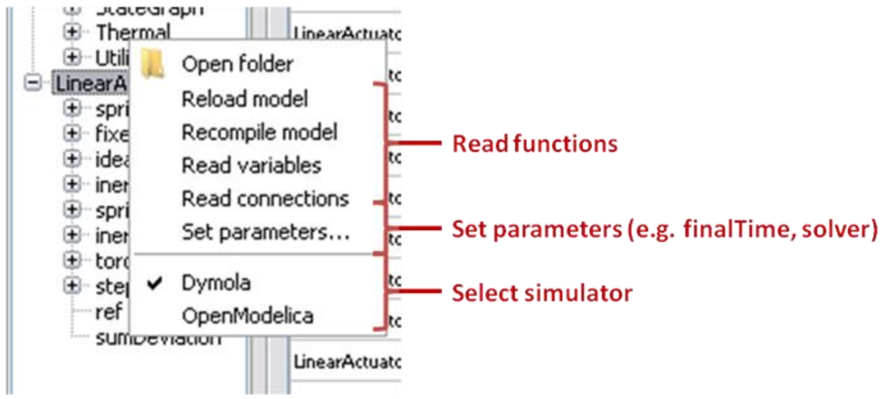


Figure 7-9. Selecting read variables, set parameters, and selecting simulator.

7.3.2.5 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.

7.3.2.6 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.

7.3.2.7 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:

- <u>Population size</u>: 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.
- <u>Off spring rate</u>: 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).
- <u>Max generations</u>: 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.
- <u>Save frequency</u>: 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 iteraion1.sav, iteration2.sav and are store in *Temp* directory, and moved to *SolvedProblems* directory when optimization is finished.
- <u>ReinitStdDev</u>: 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.

7.3.2.8 Launch

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

7.3.2.9 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 loosing 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.

7.3.3 Results

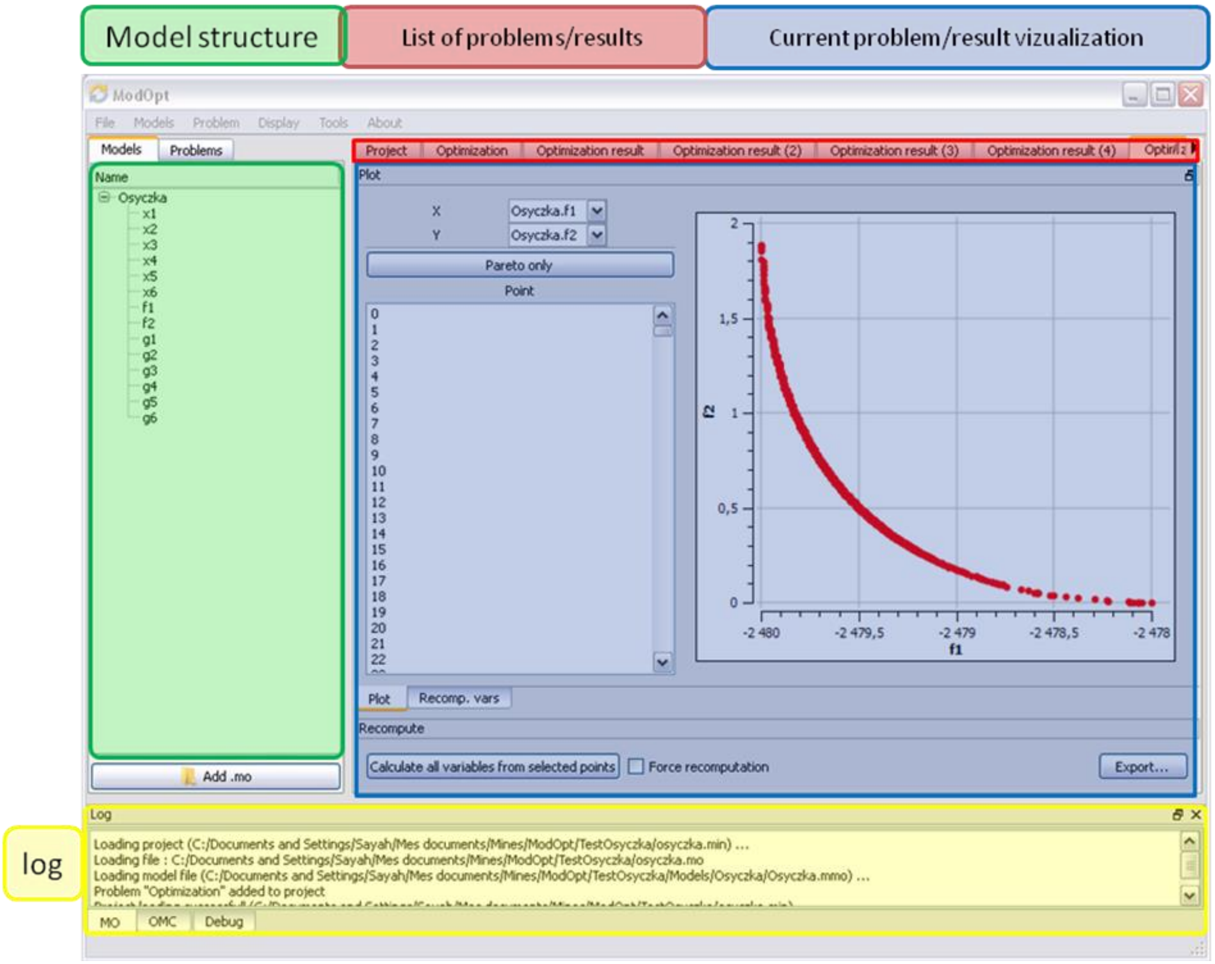
The result tab appear when the optimization is finished. It consists of two parts: a table where variables are displayed and a plot region.

7.3.3.1 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).

7.3.4 Window Regions in OMOptim GUI



```
Loading project (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestOsyczka/osyczka.min) ...
Loading file : C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestOsyczka/osyczka.mo
Loading model file (C:/Documents and Settings/Sayah/Mes documents/Mines/ModOpt/TestOsyczka/Models/Osyczka/Osyczka.mmo) ...
Problem "Optimization" added to project
```

Figure 7-10. Window regions in OMOptim GUI.

Chapter 8

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).

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 Open Modelica 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 to 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 5-8-1

Figure 5-8-1. Eclipse Setup – Switching Workspace.

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`.

Figure 5-8-2. Eclipse Setup – creating a Modelica project in the workspace.

You need to disable automatic build for the project(s) (Figure 5-8-3).

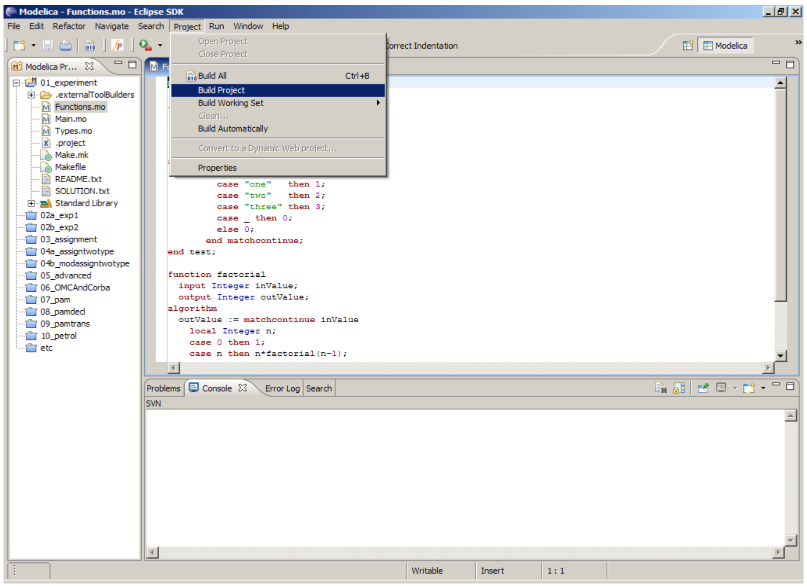
Figure 5-8-3. Eclipse Setup – disable automatic build for the projects.

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_assigntwotype, 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).



```
          case "one"   then 1;
          case "two"   then 2;
          case "three" then 3;
          case _ then 0;
          else 0;
        end matchcontinue;
end test;

function factorial
  input Integer inValue;
  output Integer outValue;
algorithm
  outValue := matchcontinue inValue
    local Integer n;
    case 0 then 1;
    case n then n*factorial(n-1);
```

Figure 8-4. Eclipse MDT – Building a project.

There are several options: building, building from scratch (clean), running, see Figure 8-5.

??missing figure

Figure 8-5. Eclipse – building and running a project.

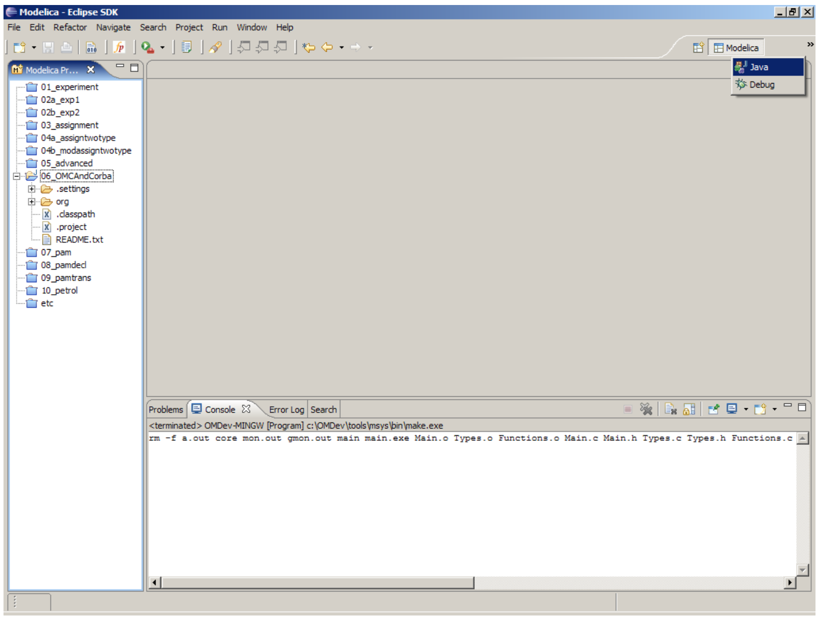
You may also open additional views, e.g as in Figure 8-6.

??missing figure

Figure 8-6. Eclipse – Opening views.

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.

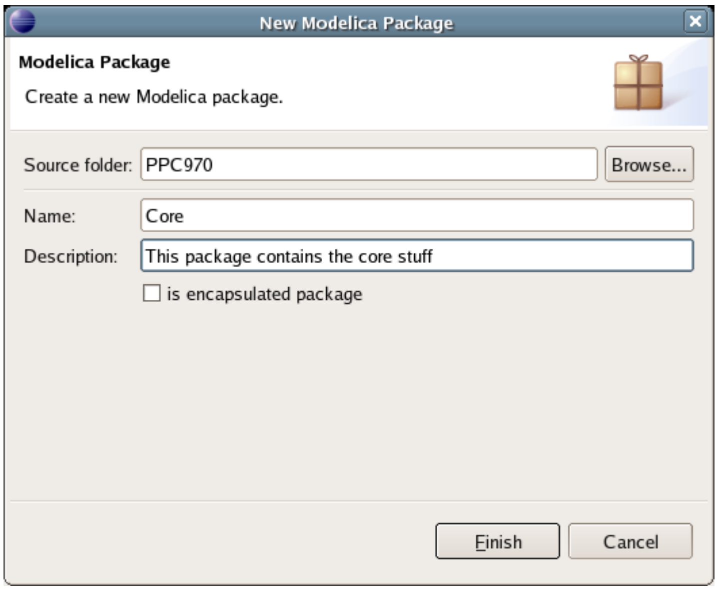


```
<terminated> OMDev-MINGW [Program] c:\OMDev\tools\msys\bin\make.exe
rm -f a.out core mon.out gmon.out main main.exe Main.o Types.o Functions.o Main.c Main.h Types.c Types.h Functions.c
```

Figure 8-7. Eclipse – Switching to another perspective – e.g. the Java Perspective.

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.


New Modelica Package

Modelica Package

Create a new Modelica package.

Source folder: PPC970

Name: Core

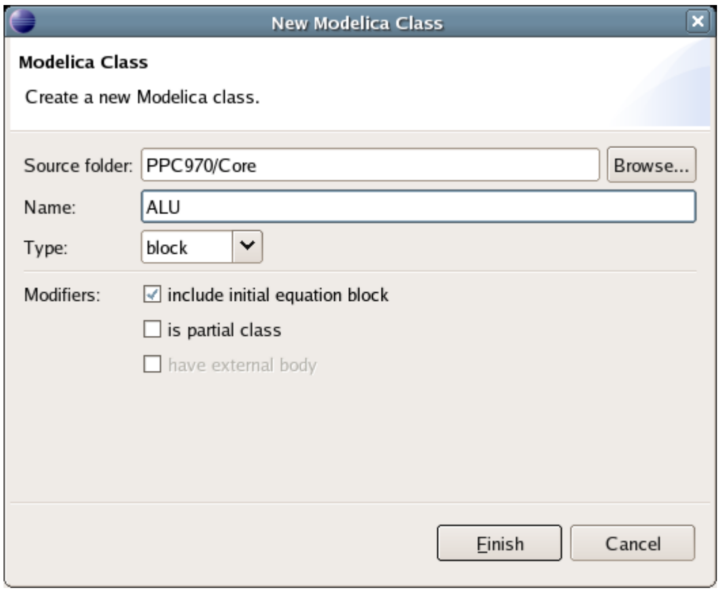
Description: This package contains the core stuff

☐ is encapsulated package

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.



Modelica Class

Create a new Modelica class.

Source folder: PPC970/Core

Name: ALU

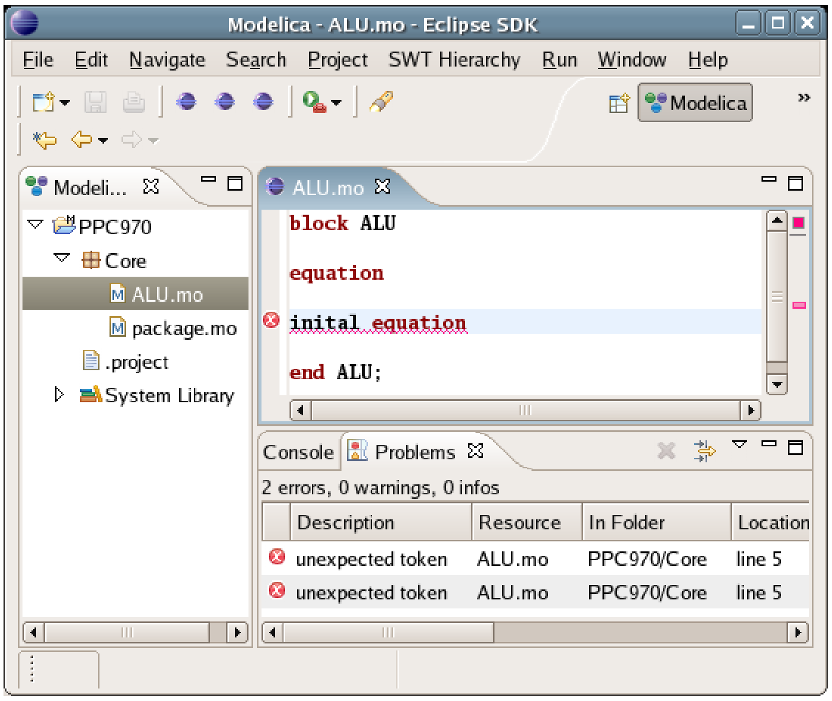
Type: block

Modifiers: [✓ include initial equation block]
is partial class
have external body

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.


Modelica - ALU.mo - Eclipse SDK
File Edit Navigate Search Project SWT Hierarchy Run Window Help
Modelica
Modeli...
PPC970
Core
ALU.mo
package.mo
.project
System Library
ALU.mo
Console
Problems

```
block ALU

equation

inital equation

end ALU;
```

2 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location
unexpected token	ALU.mo	PPC970/Core	line 5
unexpected token	ALU.mo	PPC970/Core	line 5

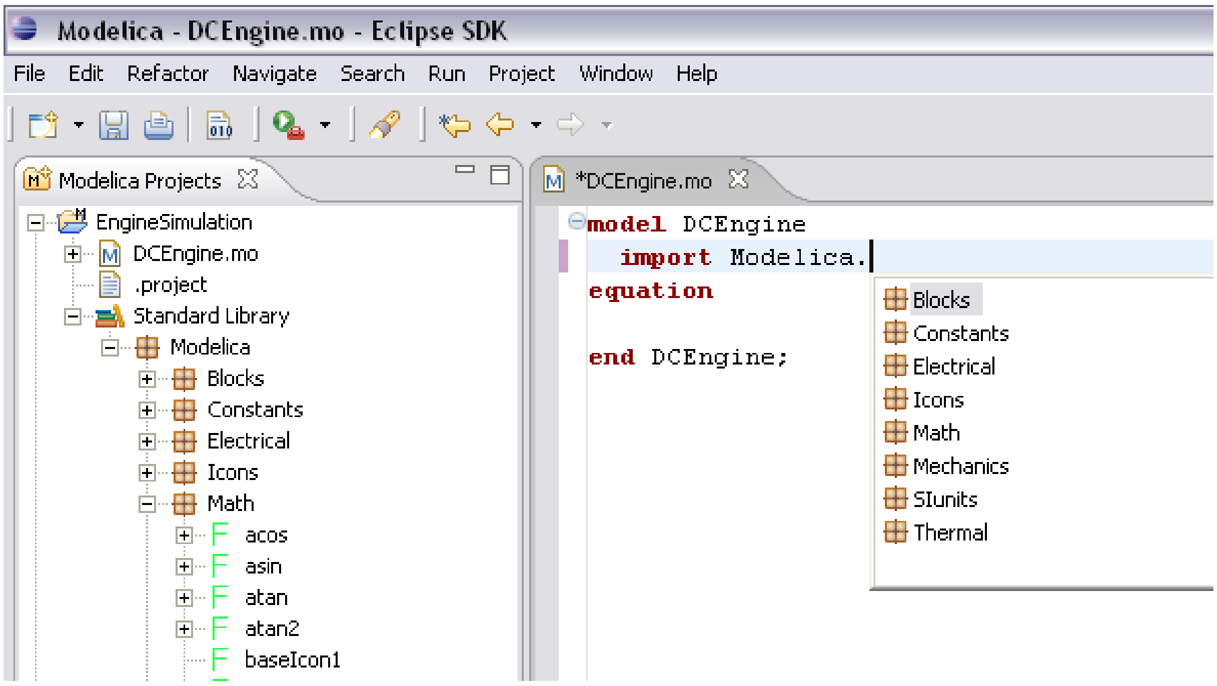
Figure 8-10. Syntax checking.

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.

8.3.11 Code Completion

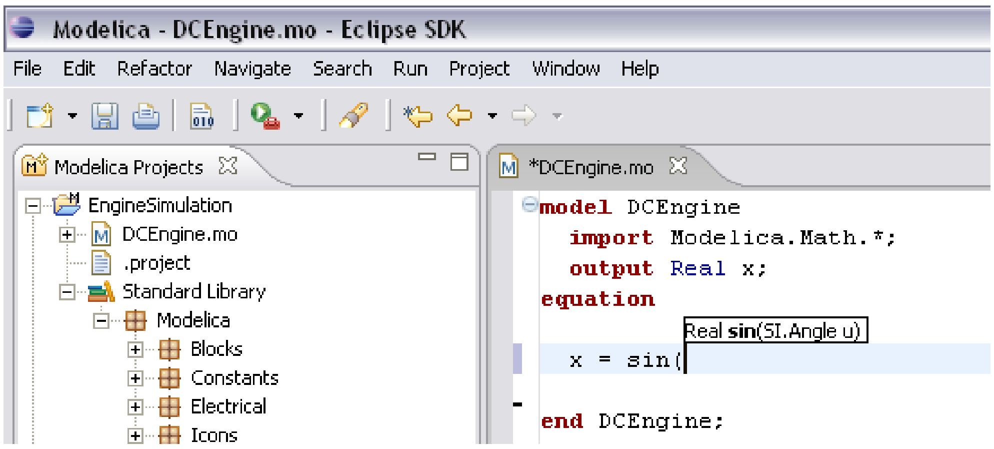
MDT supports Code Completion in two variants. The first variant, code completion when typing a dot after a class (package) name, shows alternatives in a menu. Besides the alternatives, Modelica documentation from comments is shown if is available. This makes the selection easyer.



```
model DCEngine
  import Modelica.
equation

end DCEngine;
```

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:

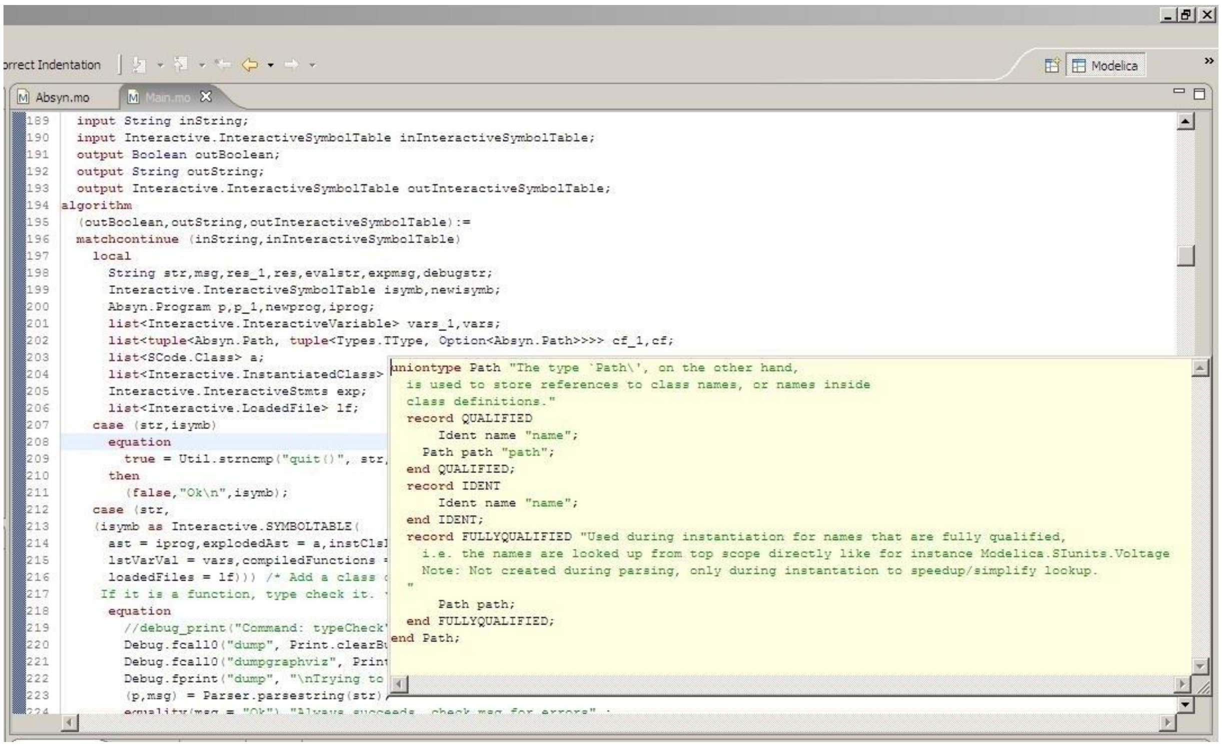


```
model DCEngine
  import Modelica.Math.*;
  output Real x;
equation

  x = sin(

end DCEngine;
```

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.



```
  input String inString;
  input Interactive.InteractiveSymbolTable inInteractiveSymbolTable;
  output Boolean outBoolean;
  output String outString;
  output Interactive.InteractiveSymbolTable outInteractiveSymbolTable;
algorithm
  (outBoolean,outString,outInteractiveSymbolTable):=
  matchcontinue (inString,inInteractiveSymbolTable)
    local
      String str,msg,res_1,res,evalstr,expmsg,debugstr;
      Interactive.InteractiveSymbolTable isymb,newisymb;
      Absyn.Program p,p_1,newprog,iprog;
      list<Interactive.InteractiveVariable> vars_1,vars;
      list<tuple<Absyn.Path, tuple<Types.TType, Option<Absyn.Path>>>> cf_1,cf;
      list<SCode.Class> a;
      list<Interactive.InstantiatedClass>
      Interactive.InteractiveStmts exp;
      list<Interactive.LoadedFile> lf;
    case (str,isymb)
      equation
        true = Util.strncmp("quit()", str,
      then
        (false,"Ok\n",isymb);
    case (str,
    (isymb as Interactive.SYMBOLTABLE(
      ast = iprog,explodedAst = a,instClsl
      lstVarVal = vars,compiledFunctions =
      loadedFiles = lf))) /* Add a class c
     If it is a function, type check it.
      equation
        //debug_print("Command: typeCheck"
        Debug.fcall0("dump", Print.clearBu
        Debug.fcall0("dumpgraphviz", Print
        Debug.fprint("dump", "\nTrying to
        (p,msg) = Parser.parsestring(str)
```

```
uniontype Path "The type `Path\', on the other hand,
  is used to store references to class names, or names inside
  class definitions."
  record QUALIFIED
      Ident name "name";
    Path path "path";
  end QUALIFIED;
  record IDENT
      Ident name "name";
  end IDENT;
  record FULLYQUALIFIED "Used during instantiation for names that are fully qualified,
    i.e. the names are looked up from top scope directly like for instance Modelica.SIunits.Voltage
    Note: Not created during parsing, only during instantation to speedup/simplify lookup.
  "
      Path path;
  end FULLYQUALIFIED;
end Path;
```

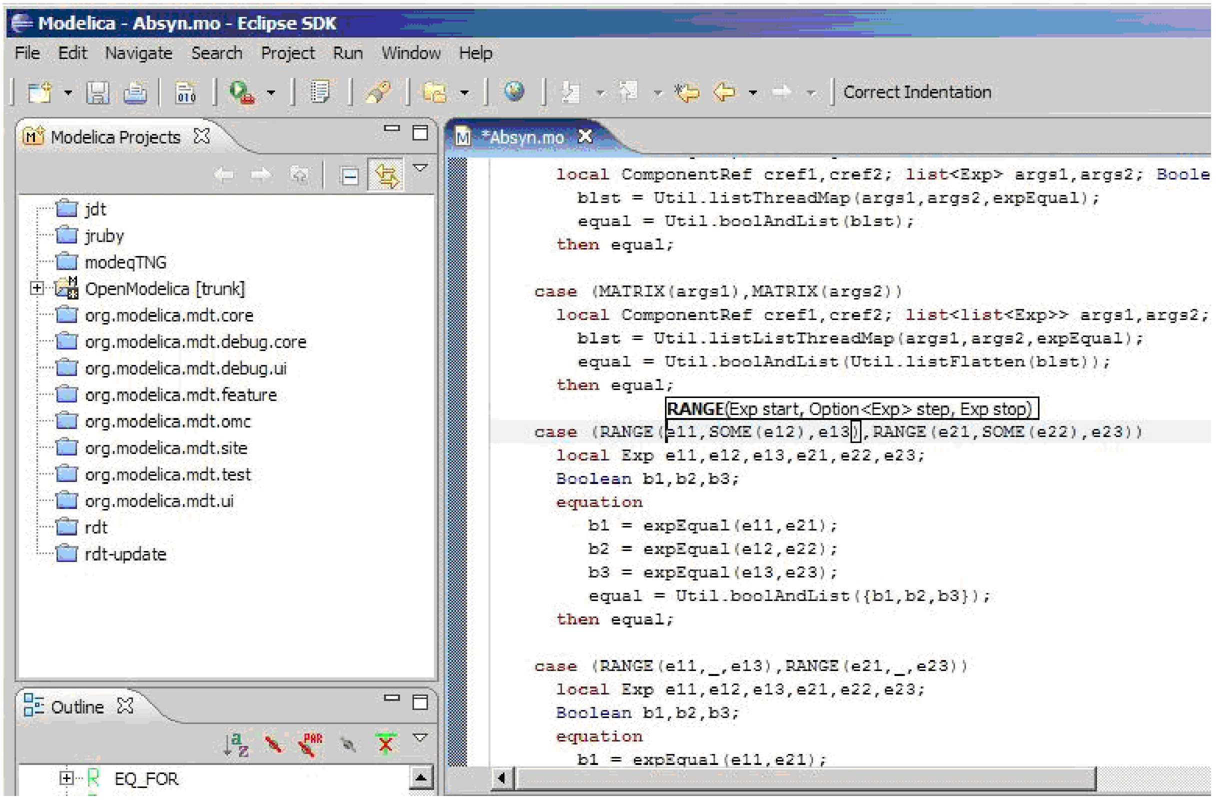
Figure 8-13. Displaying information for identifiers on hovering

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

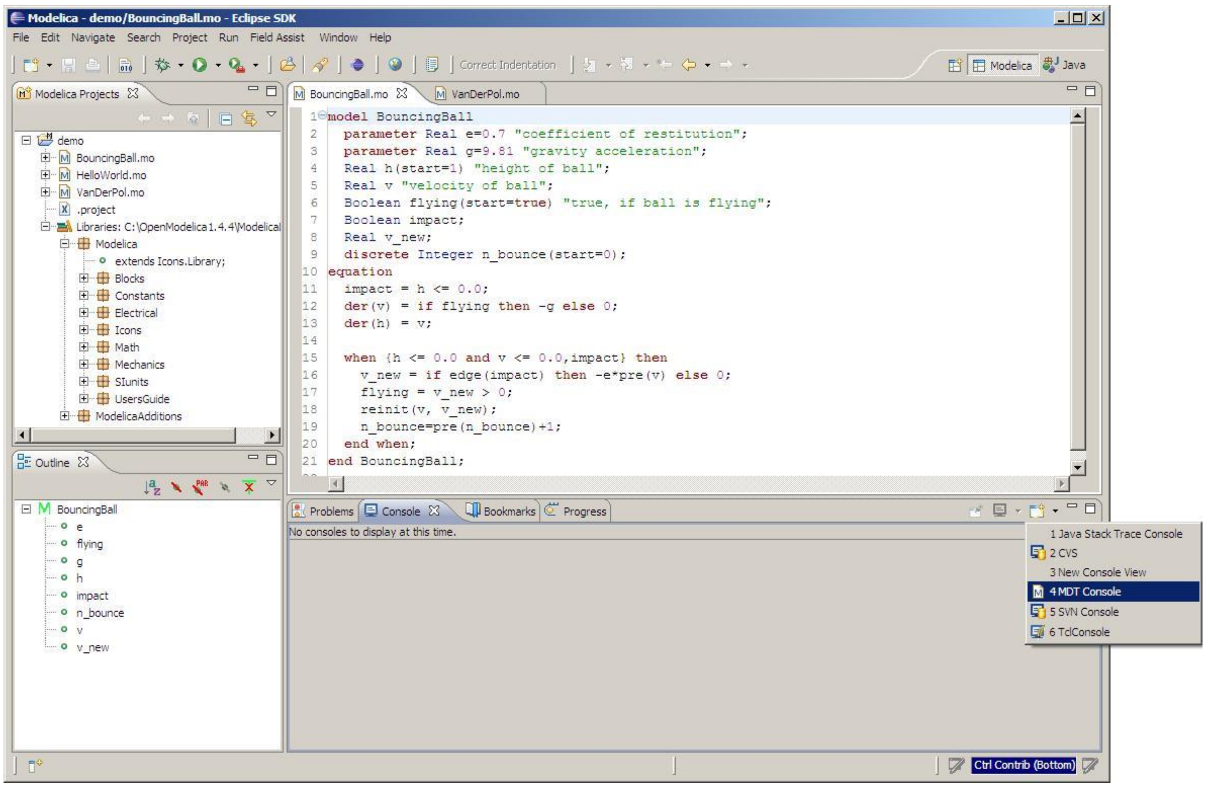
When writing records, the same functionality as for function calls is used. This is useful especially in MetaModelica when writing cases in match constructs.



```
      local ComponentRef cref1,cref2; list<Exp> args1,args2; Boole
        blst = Util.listThreadMap(args1,args2,expEqual);
        equal = Util.boolAndList(blst);
      then equal;

    case (MATRIX(args1),MATRIX(args2))
      local ComponentRef cref1,cref2; list<list<Exp>> args1,args2;
        blst = Util.listListThreadMap(args1,args2,expEqual);
        equal = Util.boolAndList(Util.listFlatten(blst));
      then equal;

    case (RANGE(e11,SOME(e12),e13),RANGE(e21,SOME(e22),e23))
      local Exp e11,e12,e13,e21,e22,e23;
      Boolean b1,b2,b3;
      equation
         b1 = expEqual(e11,e21);
         b2 = expEqual(e12,e22);
         b3 = expEqual(e13,e23);
         equal = Util.boolAndList({b1,b2,b3});
      then equal;

    case (RANGE(e11,_,e13),RANGE(e21,_,e23))
      local Exp e11,e12,e13,e21,e22,e23;
      Boolean b1,b2,b3;
      equation
        b1 = expEqual(e11,e21);
```

Figure 8-14. Code assistance when writing cases with records in MetaModelica.

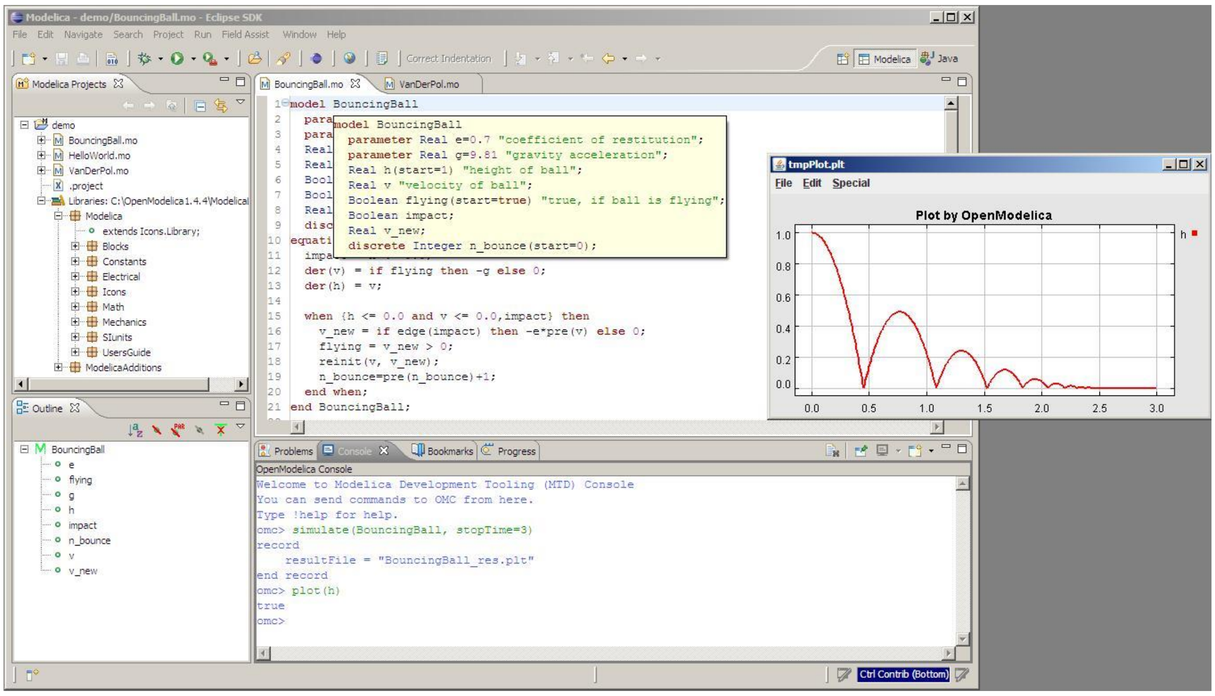
8.3.15 Using the MDT Console for Plotting



```
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;
  discrete Integer n_bounce(start=0);
equation
  impact = h <= 0.0;
  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);
    n_bounce=pre(n_bounce)+1;
  end when;
end BouncingBall;
```

No consoles to display at this time.

Figure 8-15. Activate the MDT Console



```
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;
  discrete Integer n_bounce(start=0);
```

```
equati
  impa
  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);
    n_bounce=pre(n_bounce)+1;
  end when;
end BouncingBall;
```

```
Welcome to Modelica Development Tooling (MTD) Console
You can send commands to OMC from here.
Type !help for help.
omc> simulate(BouncingBall, stopTime=3)
record
    resultFile = "BouncingBall_res.plt"
end record
omc> plot(h)
true
omc>
```

Figure 8-16. Simulation from MDT Console

Chapter 9

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 A
   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 A;
```

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

```
simulate(A, measureTime = true)
```

```
   // // record SimulationResult
   //  resultFile = "A_res.mat",
   //  messages = "Time measurements are stored in A_prof.html (human-readable)
and A_prof.xml (for XSL transforms or more details)"
   // end SimulationResult;
```

9.1 Example Report Generated for the A Model

9.1.1 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 priviliges (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 the generated file A_prof.xml, shown in Section 9.1.7 below.

9.1.2 Settings

The settings for the simulation are summarized in the table below:

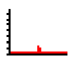
Name	**Value**
Integration method	euler
Output format	mat
Output name	A_res.mat
Output size	24.0 kB
Profiling data	A_prof.data
Profiling size	27.3 kB

9.1.3 Summary

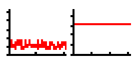
Execution times for different activities:

Task	**Time**	**Fraction**
Pre-Initialization	0.000401	19.17%
Initialization	0.000046	2.20%
Event-handling	0.000036	1.72%
Creating output file	0.000264	12.62%
Linearization	0.000000	0.00%
Time steps	0.001067	51.00%
Overhead	0.000273	13.05%
Unknown	0.000406	0.24%
Total simulation time	0.002092	100.00%

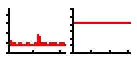
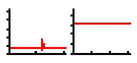
9.1.4 Global Steps

	Steps	Total Time	Fraction	Average Time	Max Time	Deviation
	499	0.001067	51.00%	2.13827655310621e-06	0.000006611	2.09x

9.1.5 Measured Function Calls

	Name	Calls	Time	Fraction	Max Time	Deviation
	A.f	1506	0.000092990	4.45%	0.000000448	6.26x

9.1.6 Measured Blocks

	Name	Calls	Time	Fraction	Max Time	Deviation
	residualFunc3	2018	0.000521137	24.91%	0.000035456	136.30x
	residualFunc1	1506	0.000393709	18.82%	0.000002735	9.46x

9.1.6.1 Equations

Name	Variables
SES_ALGORITHM 0	
SES_SIMPLE_ASSIGN 1	der(y)
residualFunc3	z2, z1
residualFunc1	x

9.1.6.2 Variables

Name	Comment
y	
der(y)	
x	This is x
z1	
z2	

S	

9.1.7 Genenerated XML for the Example

```
<!DOCTYPE doc (View Source for full doctype...)>
- <simulation>
- <modelinfo>
  <name>A</name>
  <prefix>A</prefix>
  <date>2011-03-07 12:55:53</date>
  <method>euler</method>
  <outputFormat>mat</outputFormat>
  <outputFilename>A_res.mat</outputFilename>
  <outputFilesize>24617</outputFilesize>
  <overheadTime>0.000273</overheadTime>
  <preinitTime>0.000401</preinitTime>
  <initTime>0.000046</initTime>
  <eventTime>0.000036</eventTime>
  <outputTime>0.000264</outputTime>
  <linearizeTime>0.000000</linearizeTime>
  <totalTime>0.002092</totalTime>
  <totalStepsTime>0.001067</totalStepsTime>
  <numStep>499</numStep>
  <maxTime>0.000006611</maxTime>
  </modelinfo>
- <profilingdataheader>
  <filename>A_prof.data</filename>
  <filesize>28000</filesize>
- <format>
  <uint32>step</uint32>
  <double>time</double>
  <double>cpu time</double>
  <uint32>A.f (calls)</uint32>
  <uint32>residualFunc3 (calls)</uint32>
  <uint32>residualFunc1 (calls)</uint32>
  <double>A.f (cpu time)</double>
  <double>residualFunc3 (cpu time)</double>
  <double>residualFunc1 (cpu time)</double>
  </format>
  </profilingdataheader>
- <variables>
- <variable id="1000" name="y" comment="">
  <info filename="a.mo" startline="8" startcol="3" endline="8" endcol="18"
readonly="writable" />
  </variable>
- <variable id="1001" name="der(y)" comment="">
  <info filename="a.mo" startline="8" startcol="3" endline="8" endcol="18"
readonly="writable" />
  </variable>
- <variable id="1002" name="x" comment="This is x">
  <info filename="a.mo" startline="7" startcol="3" endline="7" endcol="28"
readonly="writable" />
  </variable>
- <variable id="1003" name="z1" comment="">
  <info filename="a.mo" startline="9" startcol="3" endline="9" endcol="20"
readonly="writable" />
  </variable>
- <variable id="1004" name="z2" comment="">
  <info filename="a.mo" startline="10" startcol="3" endline="10" endcol="20"
readonly="writable" />
  </variable>
```

```
- <variable id="1005" name="s" comment="">
  <info filename="a.mo" startline="6" startcol="3" endline="6" endcol="19"
readonly="writable" />
  </variable>
  </variables>
- <functions>
- <function id="1006">
  <name>A.f</name>
  <ncall>1506</ncall>
  <time>0.000092990</time>
  <maxTime>0.000000448</maxTime>
  <info filename="a.mo" startline="2" startcol="3" endline="5" endcol="8"
readonly="writable" />
  </function>
  </functions>
- <equations>
- <equation id="1007" name="SES_ALGORITHM 0">
  <refs />
  </equation>
- <equation id="1008" name="SES_SIMPLE_ASSIGN 1">
- <refs>
  <ref refid="1001" />
  </refs>
  </equation>
- <equation id="1009" name="residualFunc3">
- <refs>
  <ref refid="1004" />
  <ref refid="1003" />
  </refs>
  </equation>
- <equation id="1010" name="residualFunc1">
- <refs>
  <ref refid="1002" />
  </refs>
  </equation>
  </equations>
- <profileblocks>
- <profileblock>
  <ref refid="1009" />
  <ncall>2018</ncall>
  <time>0.000521137</time>
  <maxTime>0.000035456</maxTime>
  </profileblock>
- <profileblock>
  <ref refid="1010" />
  <ncall>1506</ncall>
  <time>0.000393709</time>
  <maxTime>0.000002735</maxTime>
  </profileblock>
  </profileblocks>
  </simulation>
```

Chapter 10

MDT Debugger for Algorithmic Modelica

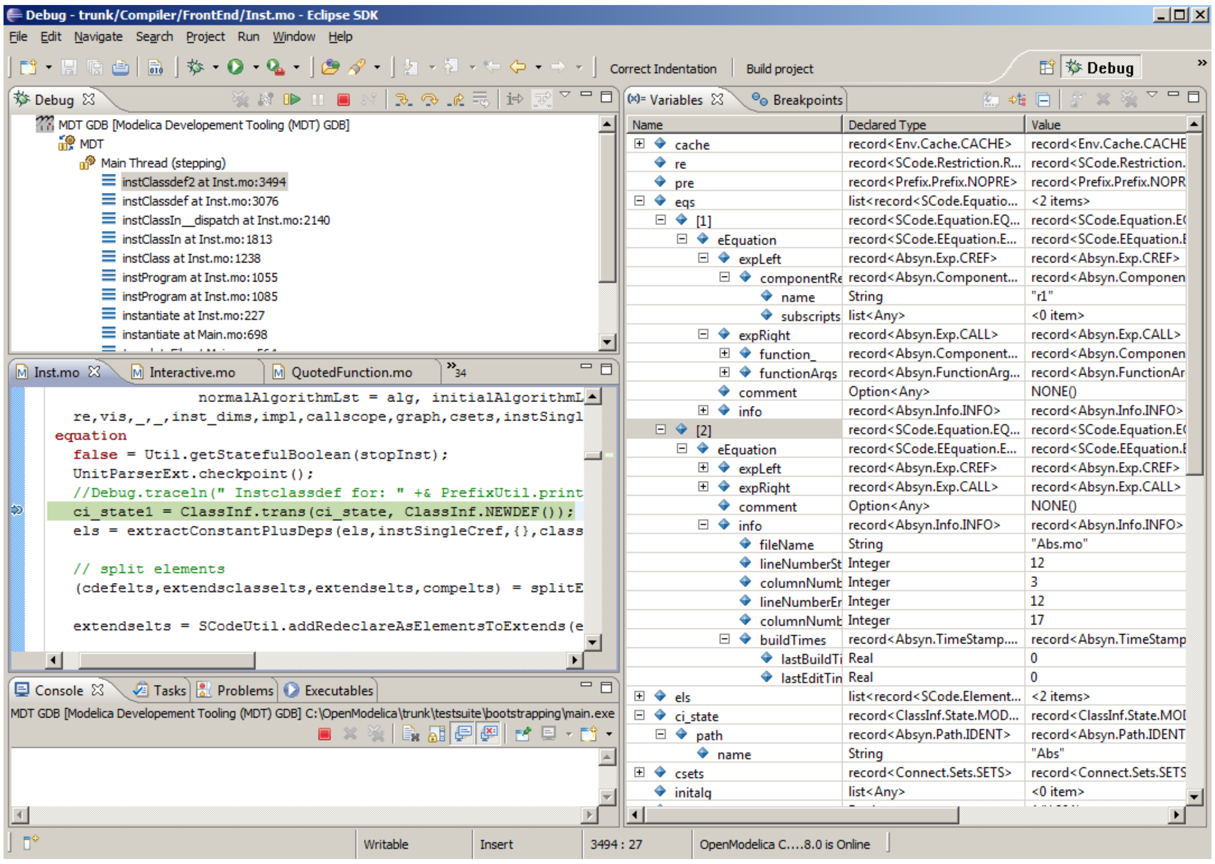
The algorithmic code debugger, used for the algorithmic subset of the Modelica language as well as the MetaModelica language is described in Section 10.1. Using this debugger replaces debugging of algorithmic code by primitive means such as print statements or asserts which is complex, time-consuming 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.

10.1 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.



```
MDT GDB [Modelica Developement Tooling (MDT) GDB]
  MDT
    Main Thread (stepping)
      instClassdef2 at Inst.mo:3494
      instClassdef at Inst.mo:3076
      instClassIn__dispatch at Inst.mo:2140
      instClassIn at Inst.mo:1813
      instClass at Inst.mo:1238
      instProgram at Inst.mo:1055
      instProgram at Inst.mo:1085
      instantiate at Inst.mo:227
      instantiate at Main.mo:698
```

```
                 normalAlgorithmLst = alg, initialAlgorithmL
  re,vis,_,_,inst_dims,impl,callscope,graph,csets,instSingl
equation
  false = Util.getStatefulBoolean(stopInst);
  UnitParserExt.checkpoint();
  //Debug.traceln(" Instclassdef for: " +& PrefixUtil.print
  ci_state1 = ClassInf.trans(ci_state, ClassInf.NEWDEF());
  els = extractConstantPlusDeps(els,instSingleCref,{},class

  // split elements
  (cdefelts,extendsclasselts,extendselts,compelts) = splitE

  extendselts = SCodeUtil.addRedeclareAsElementsToExtends(e
```

```
MDT GDB [Modelica Developement Tooling (MDT) GDB] C:\OpenModelica\trunk\testsuite\bootstrapping\main.exe
```

Name	Declared Type	Value
cache	record<Env.Cache.CACHE>	record<Env.Cache.CACHE
re	record<SCode.Restriction.R...	record<SCode.Restriction.
pre	record<Prefix.Prefix.NOPRE>	record<Prefix.Prefix.NOPR
eqs	list<record<SCode.Equatio...	<2 items>
[1]	record<SCode.Equation.EQ...	record<SCode.Equation.E(
eEquation	record<SCode.EEquation.E...	record<SCode.EEquation.E
expLeft	record<Absyn.Exp.CREF>	record<Absyn.Exp.CREF>
componentRe	record<Absyn.Component...	record<Absyn.Componen
name	String	"r1"
subscripts	list<Any>	<0 item>
expRight	record<Absyn.Exp.CALL>	record<Absyn.Exp.CALL>
function_	record<Absyn.Component...	record<Absyn.Componen
functionArgs	record<Absyn.FunctionArg...	record<Absyn.FunctionAr
comment	Option<Any>	NONE()
info	record<Absyn.Info.INFO>	record<Absyn.Info.INFO>
[2]	record<SCode.Equation.EQ...	record<SCode.Equation.E(
eEquation	record<SCode.EEquation.E...	record<SCode.EEquation.E
expLeft	record<Absyn.Exp.CREF>	record<Absyn.Exp.CREF>
expRight	record<Absyn.Exp.CALL>	record<Absyn.Exp.CALL>
comment	Option<Any>	NONE()
info	record<Absyn.Info.INFO>	record<Absyn.Info.INFO>
fileName	String	"Abs.mo"
lineNumberSt	Integer	12
columnNumb	Integer	3
lineNumberEr	Integer	12
columnNumb	Integer	17
buildTimes	record<Absyn.TimeStamp....	record<Absyn.TimeStamp
lastBuildTi	Real	0
lastEditTin	Real	0
els	list<record<SCode.Element...	<2 items>
ci_state	record<ClassInf.State.MOD...	record<ClassInf.State.MOI
path	record<Absyn.Path.IDENT>	record<Absyn.Path.IDENT
name	String	"Abs"
csets	record<Connect.Sets.SETS>	record<Connect.Sets.SETS
initalg	list<Any>	<0 item>

Figure 10-1. Debugging functionality.

10.1.1 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.

10.1.1.1 Create mos file

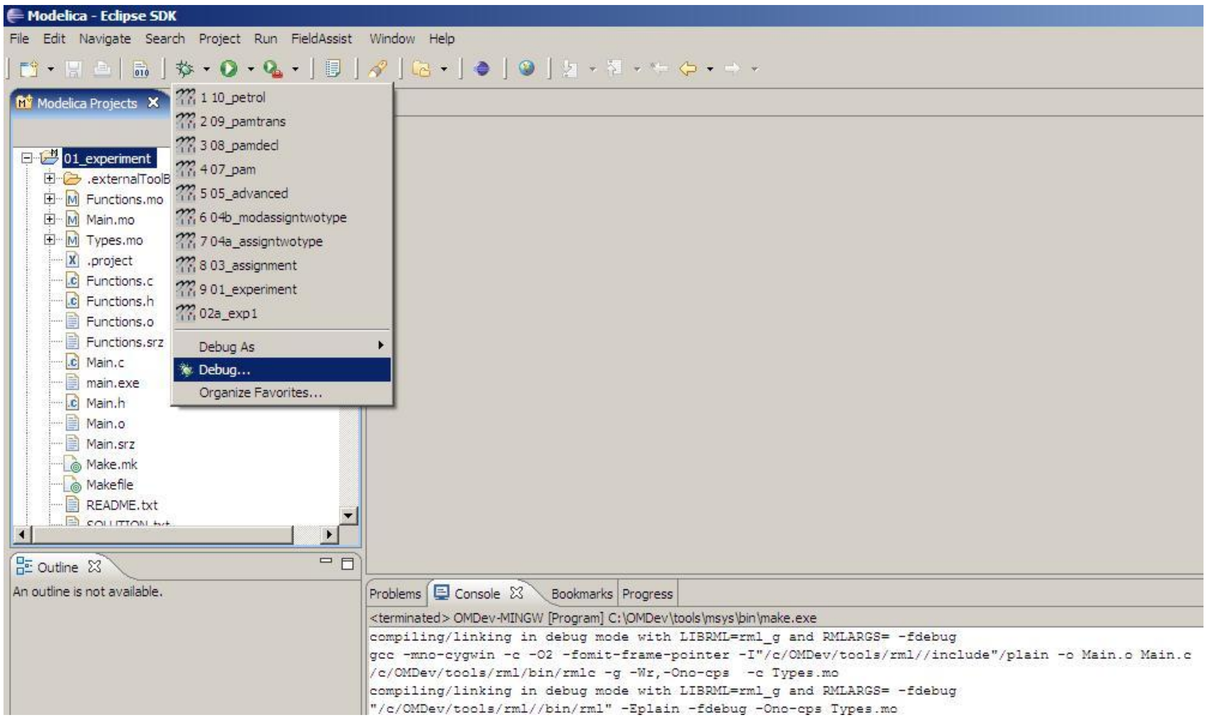
In order to debug Modelica code we need to load the Modelica files into the OpenModelica Compiler. For this we can write a small script file like this,

```
setCommandLineOptions({"+d=rml,noevalfunc","+g=MetaModelica"});
setEnvironmentVar("MODELICAUSERCFLAGS","-g");
loadFile("HelloWorld.mo");
getErrorString();
HelloWorld(120.0);
getErrorString();
```

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);`

10.1.1.2 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.

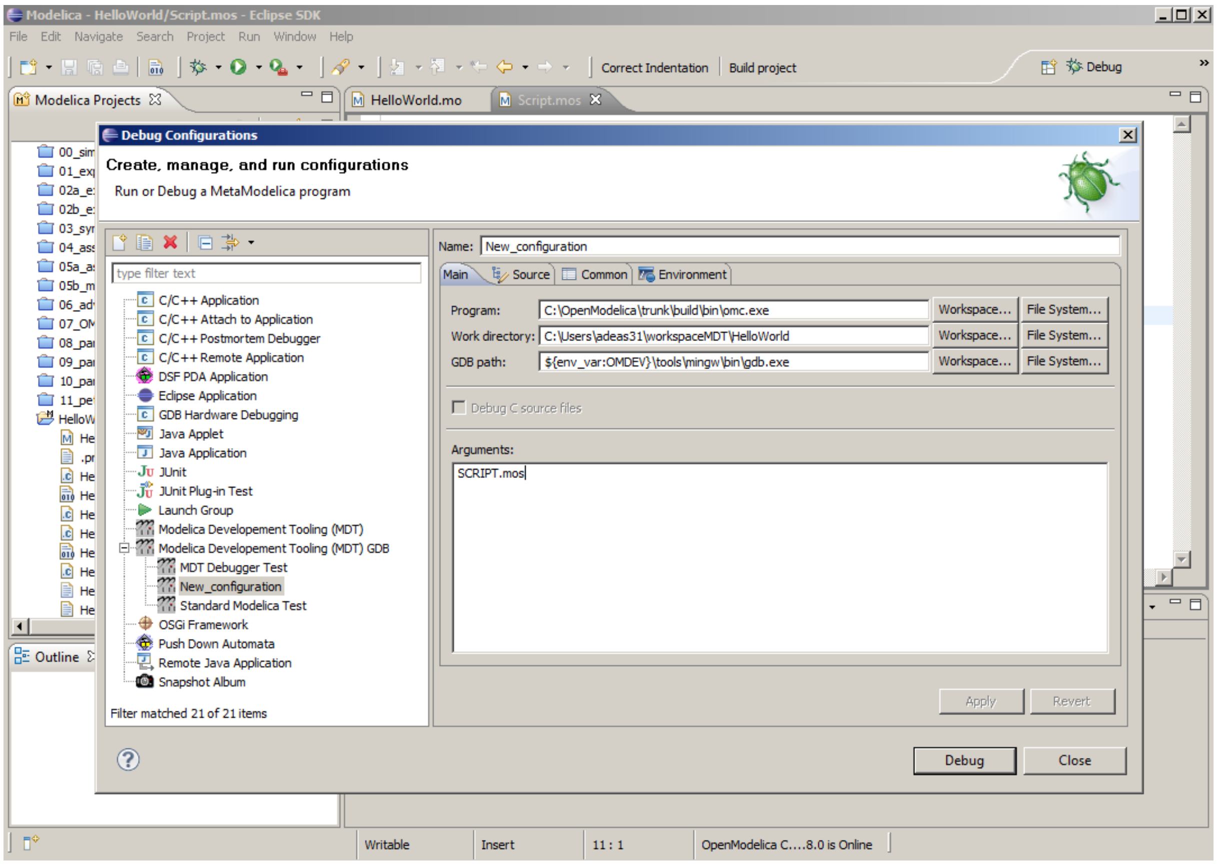


```
<terminated> OMDev-MINGW [Program] C:\OMDev\tools\msys\bin\make.exe
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Main.o Main.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps  -c Types.mo
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
"/c/OMDev/tools/rml//bin/rml" -Eplain -fdebug -Ono-cps Types.mo
```

Figure 10-2. Accessing the debug configuration dialog.

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 must pass our script file as an argument to `OMC`.


Modelica - HelloWorld/Script.mos - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Correct Indentation
Build project
Debug
Modelica Projects
HelloWorld.mo
Script.mos
Debug Configurations
Outline
Writable
Insert
11 : 1
OpenModelica C....8.0 is Online

Create, manage, and run configurations

Run or Debug a MetaModelica program

type filter text

C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
DSF PDA Application
Eclipse Application
GDB Hardware Debugging
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Launch Group
Modelica Developement Tooling (MDT)
Modelica Developement Tooling (MDT) GDB
MDT Debugger Test
New_configuration
Standard Modelica Test
OSGi Framework
Push Down Automata
Remote Java Application
Snapshot Album

Filter matched 21 of 21 items

Name: New_configuration

Main | Source | Common | Environment

Program: C:\OpenModelica\trunk\build\bin\omc.exe

Work directory: C:\Users\adeas31\workspaceMDT\HelloWorld

GDB path: ${env_var:OMDEV}\tools\mingw\bin\gdb.exe

Debug C source files

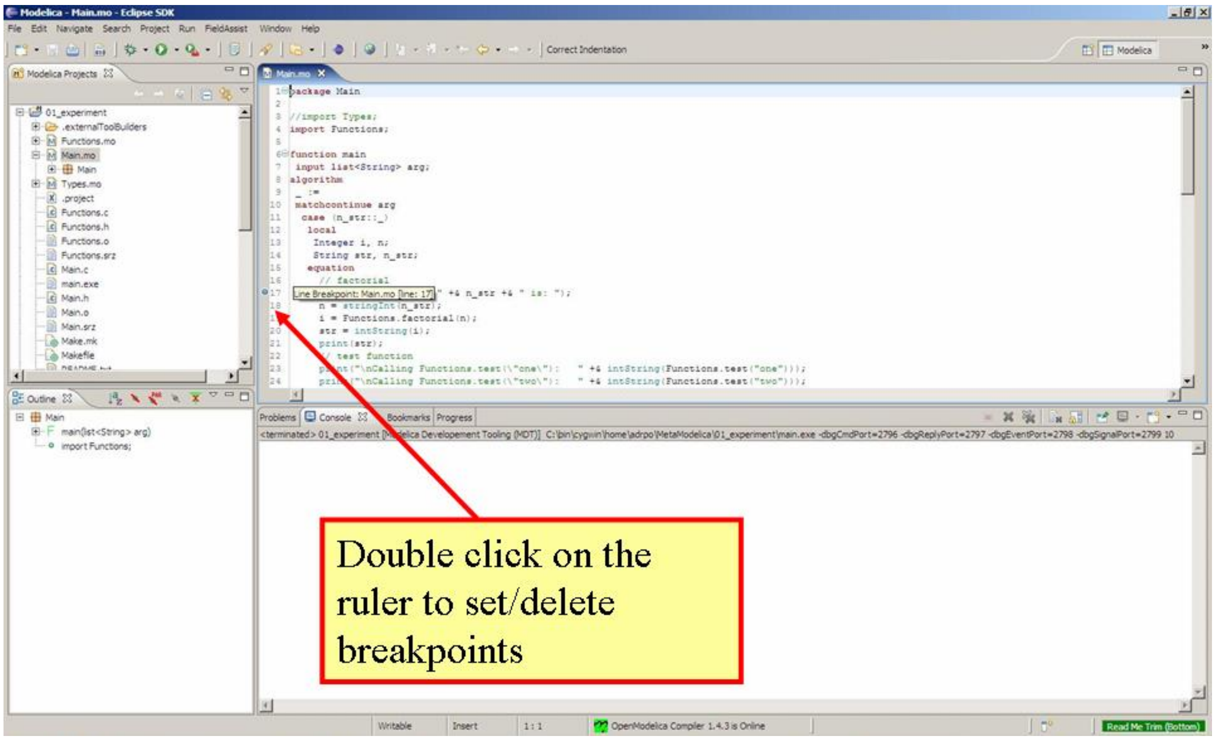
Arguments:

SCRIPT.mos

Figure 10-3. Creating the Debug Configuration.

10.1.1.3 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.



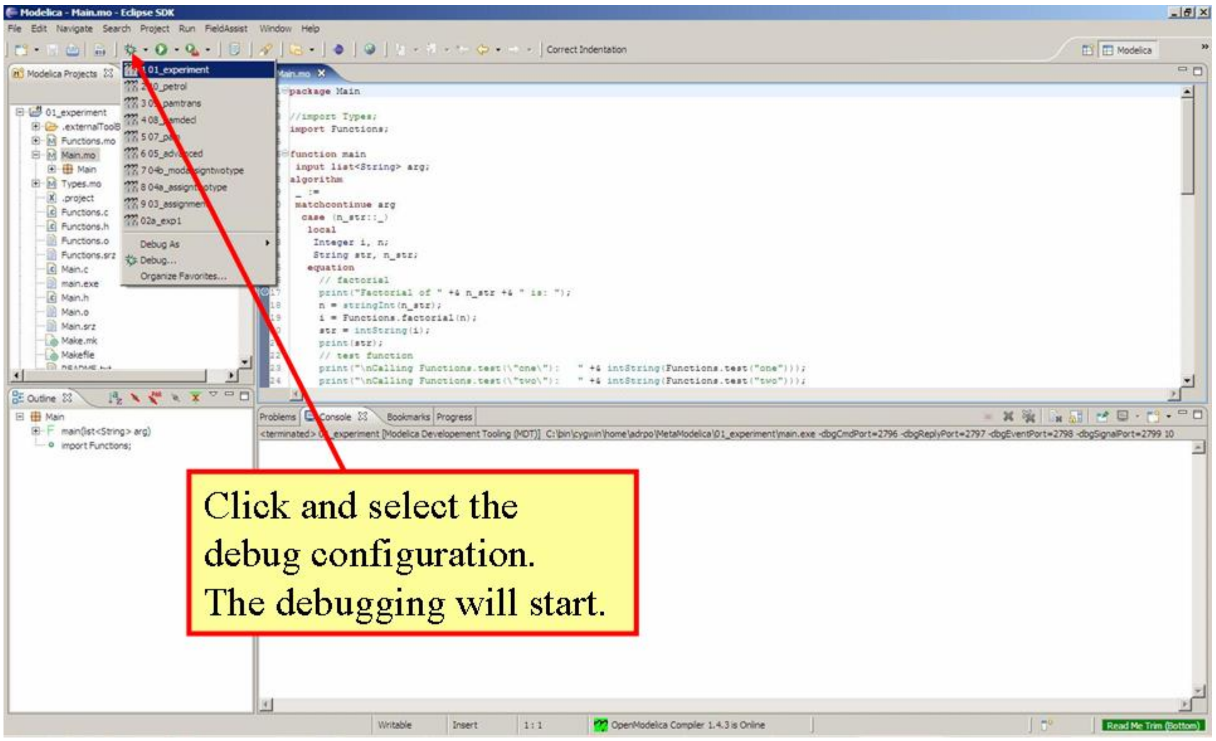
```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
Line Breakpoint: Main.mo [line: 17]" +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
     print(str);
     // test function
     print("\nCalling Functions.test(\"one\"):   " +& intString(Functions.test("one")));
     print("\nCalling Functions.test(\"two\"):   " +& intString(Functions.test("two")));
```

```
<terminated> 01_experiment [Modelica Developement Tooling (MDT)]  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=2796 -dbgReplyPort=2797 -dbgEventPort=2798 -dbgSignalPort=2799 10
```

Double click on the ruler to set/delete breakpoints

Figure 10-4. Setting/deleting breakpoints.

10.1.1.4 Starting the debugging session and enabling the debug perspective

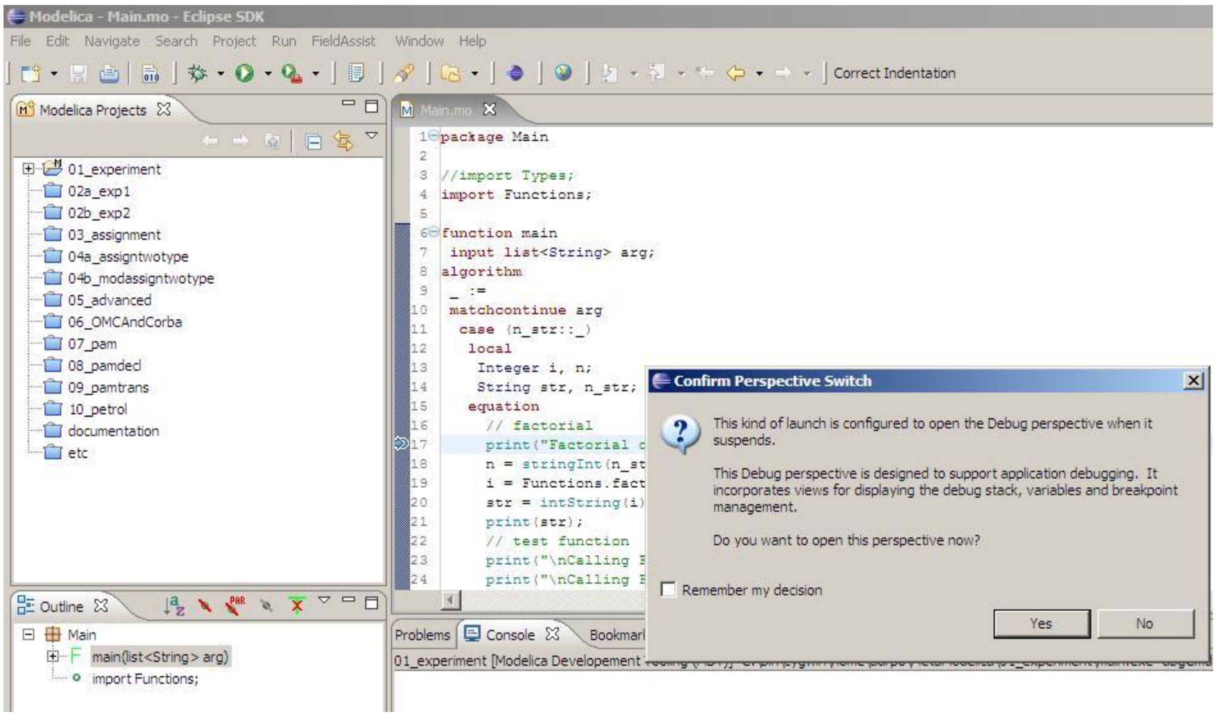


```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial of " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
     print(str);
     // test function
     print("\nCalling Functions.test(\"one\"):   " +& intString(Functions.test("one")));
     print("\nCalling Functions.test(\"two\"):   " +& intString(Functions.test("two")));
```

```
<terminated> 01_experiment [Modelica Developement Tooling (MDT)]  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=2796 -dbgReplyPort=2797 -dbgEventPort=2798 -dbgSignalPort=2799 10
```

Click and select the debug configuration. The debugging will start.

Figure 10-5. Starting the debugging session.



```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial c
     n = stringInt(n_st
     i = Functions.fact
     str = intString(i)
     print(str);
     // test function
     print("\nCalling F
     print("\nCalling F
```

01_experiment [Modelica Developement

Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

Yes No

Figure 10-6. Eclipse will ask if the user wants to switch to the debugging perspective.

10.1.2 Debugging OpenModelica

- Compile and create OpenModelica executable as usual.
- Go to `/trunk/Compiler/boot`
- Now run `make -f Makefile.omdev.mingw` (choose the right make file depending on your platform), this will create a bootstrapped `omc` and will replace the `omc` executable in `/trunk/build/bin`.
- Now in the debug configuration as explained in Section 10.1.1.2 choose `omc` executable as the program.

10.1.3 The Debugging Perspective

The debug view primarily consists of two main views:

- Stack Frames View
- Variables View

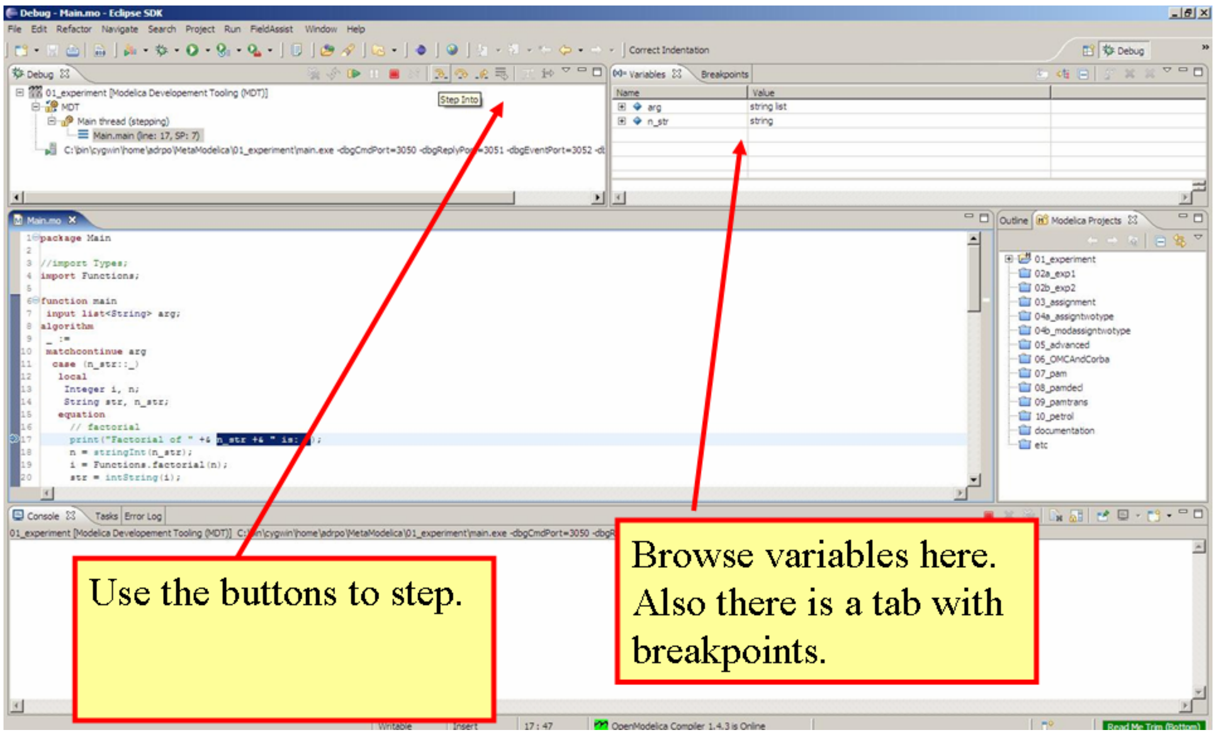
The stack frame view, shown in the figure below, shows a list of frames that indicates how the flow had moved from one function to another or from one file to another. This allows backtracing of the code. It is very much possible to select the previous frame in the stack and inspect the values of the 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 colums:

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

- Actual Type – the mapped C type.

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).


Debug - Main.mo - Eclipse SDK
File Edit Refactor Navigate Search Project Run FieldAssist Window Help
Debug
Step Into
Variables
Breakpoints
Main.mo
Outline
Modelica Projects
Console
Tasks
Error Log
Use the buttons to step.
Browse variables here. Also there is a tab with breakpoints.

```
01_experiment [Modelica Developement Tooling (MDT)]
  MDT
    Main thread (stepping)
      Main.main (line: 17, SP: 7)
  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgReplyPort=3051 -dbgEventPort=3052 -d
```

Name	Value
arg	string list
n_str	string

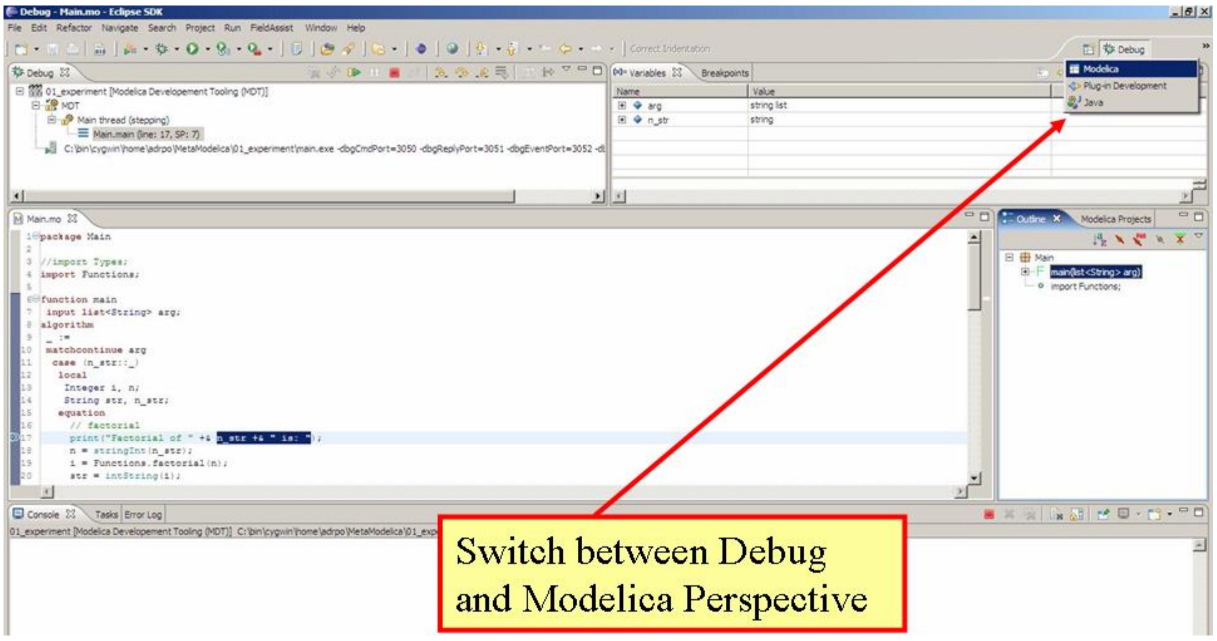
```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial of " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
```

```
01_experiment
02a_exp1
02b_exp2
03_assignment
04a_assigntwotype
04b_modassigntwotype
05_advanced
06_OMCAndCorba
07_pam
08_pamdecl
09_pamtrans
10_petrol
documentation
etc
```

```
01_experiment [Modelica Developement Tooling (MDT)] C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgR
```

Writable   Insert   17 : 47   OpenModelica Compiler 1.4.3 is Online

Figure 10-7. The debugging perspective.


Debug - Main.mo - Eclipse SDK
File Edit Refactor Navigate Search Project Run FieldAssist Window Help
Debug
Modelica
Plug-in Development
Java
Variables
Breakpoints
Main.mo
Outline
Modelica Projects
Console
Tasks
Error Log
Switch between Debug and Modelica Perspective

```
01_experiment [Modelica Developement Tooling (MDT)]
  MDT
    Main thread (stepping)
      Main.main (line: 17, SP: 7)
  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgReplyPort=3051 -dbgEventPort=3052 -d
```

Name	Value
arg	string list
n_str	string

```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial of " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
```

```
Main
  main(list<String> arg)
  import Functions;
```

```
01_experiment [Modelica Developement Tooling (MDT)] C:\bin\cygwin\home\adrpo\MetaModelica\01_exp
```

Figure 10-8. Switching between perspectives.

Chapter 11

Modelica3D

Modelica3D is a lightweight, platform independent 3D-visualisation library for Modelica. Read more about the Modelica3D library here https://mlcontrol.uebb.tu-berlin.de/redmine/projects/modelica3d-public.

11.1 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. Recommmended is pygtk-all-in-one-2.24.2.win32-py2.7.msi.

Run the Modelica3D server by executing the dbus-server.py script located at OPENMODELICAHOME/lib/omlibrary-modelica3d/osg-gtk.

```
python dbus-server.py
```

This 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 simulates the `DoublePendulum` example,

```
loadModelica3D();getErrorString();
loadString("model DoublePendulum
  extends Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum;
  inner ModelicaServices.Modelica3D.Controller m3d_control;
end DoublePendulum;");getErrorString();
simulate(DoublePendulum);
```

If everything goes fine a visualization window will pop-up. To visualize any models from the MultiBody library you can use this script and change the extends to point to the model you want.

```
loadModel(Modelica, {"3.1"}); getErrorString();
loadModel(ModelicaServices, {"1.0 modelica3d"}); getErrorString();
// get the path to modelica3d patches
m3d_path := getInstallationDirectoryPath() + "/lib/omlibrary-modelica3d/";
// load the changed Modelica.Utilities.Internal
loadFile(m3d_path + "Internal.mo");
loadString("
model Visualize_DoublePendulum
  inner ModelicaServices.Modelica3D.Controller m3d_control;
  extends Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum;
end Visualize_DoublePendulum;
");
```

```
simulate(Visualize_DoublePendulum, stopTime=30); getErrorString();
```

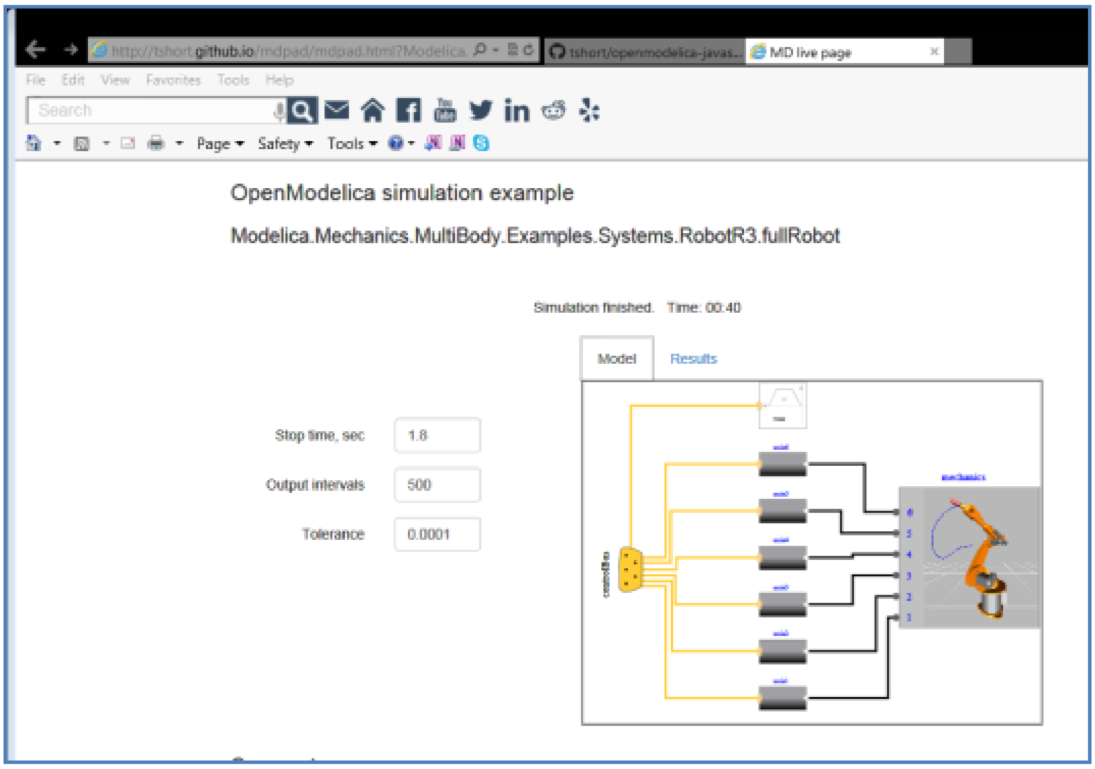
Chapter 12

Simulation in Web Browser

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

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

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

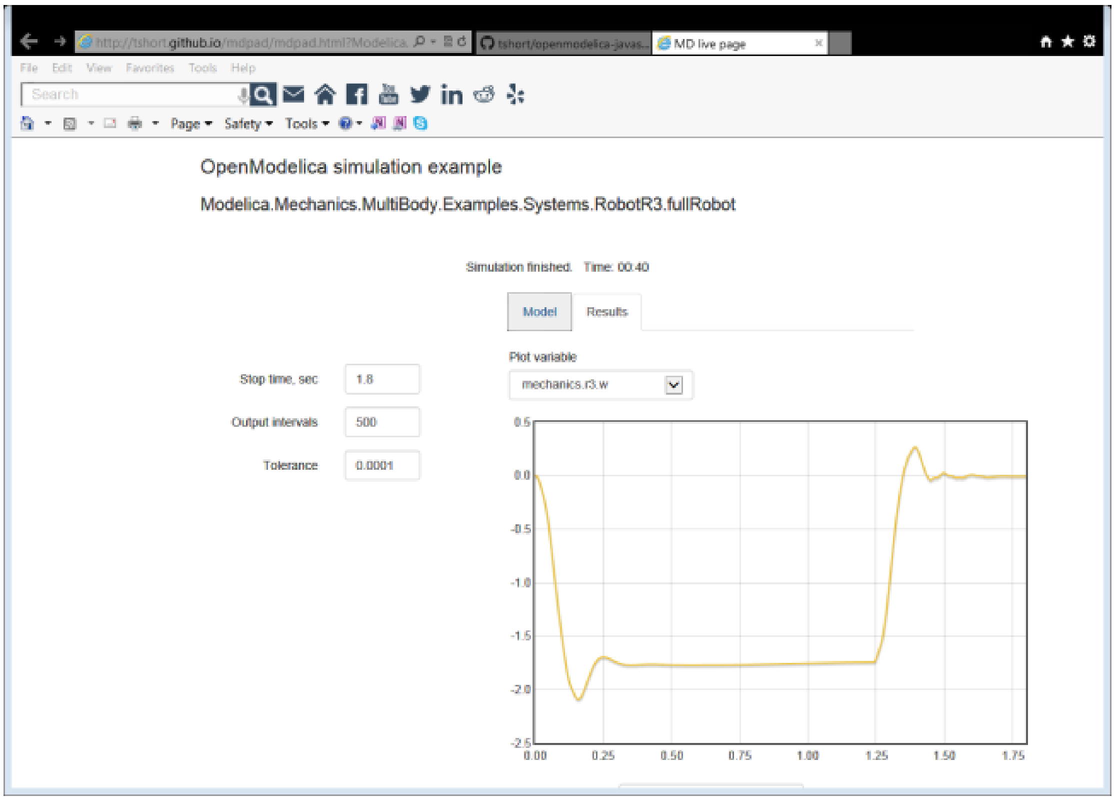

http://tshort.github.io/mdpad/mdpad.html?Modelica
tshort/openmodelica-javas...
MD live page
File Edit View Favorites Tools Help
Search
Page Safety Tools
Model
Results
mechanics

OpenModelica simulation example

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot

Simulation finished. Time: 00:40

Stop time, sec	1.8
Output intervals	500
Tolerance	0.0001


http://tshort.github.io/mdpad/mdpad.html?Modelica
tshort/openmodelica-javas...
MD live page
File Edit View Favorites Tools Help
Search
Page Safety Tools
Model
Results
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

OpenModelica simulation example

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot

Simulation finished. Time: 00:40

Stop time, sec: 1.8

Output intervals: 500

Tolerance: 0.0001

Plot variable: mechanics.r3.w

Chapter 13

Interoperability – C, Java, and Python

Below is information and examples about the OpenModelica external C and Java 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
  Real x(start=1.0),y(start=2.0);
equation
  der(x)=-ExternalFunc1(x);
  der(y)=-ExternalFunc2(y);
end ExternalLibraries;

function ExternalFunc1
  input Real x;
  output Real y;
external
  y=ExternalFunc1_ext(x) annotation(Library="libExternalFunc1_ext.o",
                                    Include="#include \"ExternalFunc1_ext.h\"");
end ExternalFunc1;

function ExternalFunc2
  input Real x;
  output Real y;
  external "C" annotation(Library="libExternalFunc2.a",
                          Include="#include  \"ExternalFunc2.h\"");
end ExternalFunc2;
```

These C (.c) files and header files (.h) are needed:

```
/* file: ExternalFunc1.c */
double ExternalFunc1_ext(double x)
{
  double res;
  res = x+2.0*x*x;
  return res;
}

/* Header file ExternalFunc1_ext.h for ExternalFunc1 function */
double ExternalFunc1_ext(double);

/* file: ExternalFunc2.c */
double ExternalFunc2(double x)
{
  double res;
  res = (x-1.0)*(x+2.0);
  return res;
}

/* Header file ExternalFunc2.h for ExternalFunc2 */
double ExternalFunc2(double);
```

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

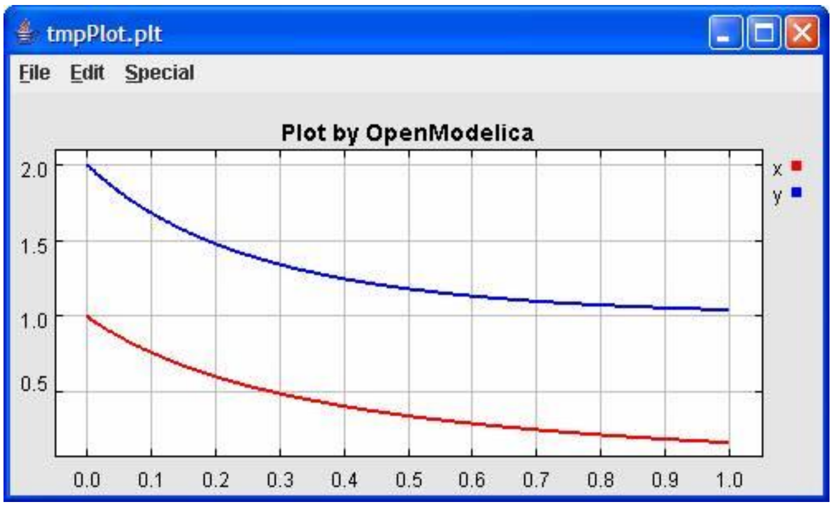
```
loadFile("ExternalLibraries.mo");
system("gcc -c -o libExternalFunc1_ext.o ExternalFunc1.c");
system("gcc -c -o libExternalFunc2.a ExternalFunc2.c");
simulate(ExternalLibraries);
```

We run the script:

```
>> runScript("ExternalLibraries.mos");
```

and plot the results:

```
>> plot({x,y});
```



13.2 Calling External Java Functions

There exists a bidirectional OpenModelica-Java CORBA interface, which is capable of passing both standard Modelica data types, as well as abstract syntax trees and list structures to and from Java and process them in either Java or the OpenModelica Compiler.

The following is a small example (`ExternalJavaLib.mo`) to show the use of external Java function calls in Modelica, i.e., only the case calling Java from Modelica:

```
model ExternalJavaLib
  Real x(start=1.0);
equation
  der(x)=- ExternalJavaLog(x);
end ExternalJavaLib;
```

```
function ExternalJavaLog
  input Real x;
  output Real y;
external "Java" y='java.lang.Math.log'(x) annotation(JavaMapping = "simple");
end ExternalJavaLog;
```

The datatypes are mapped according to the tables below. There is one mapping for interacting with existing Java code (simple), and a default mapping that handles all OpenModelica datatypes. The definitions of the default datatypes exist in the Java package org.openmodelica (see $OPENMODELICA-HOME/share/java/modelica_java.jar).

For more complete examples on how to use the Java interface, download the OpenModelica source code and view the examples in testsuite/openmodelica/java.

Modelica	**Default Mapping**	**JavaMapping = "simple"**
Real	ModelicaReal	double
Integer	ModelicaInteger	int
Boolean	ModelicaBoolean	bool
String	ModelicaString	String
Record	ModelicaRecord	
T[:]	ModelicaArray<T>	

MetaModelica	**Default Mapping**
list<T>	ModelicaArray<T>
tuple<T1, ... , Tn>	ModelicaTuple
Option<T>	ModelicaOption<T>
Uniontype	IModelicaRecord

13.3 Calling Python Code

This section describes a simple-minded approach to calling Python code from OpenModelica. For a description of Python scripting with OpenModelica, see Chapter 14.

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

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

```
# file: PythonCaller.py
#!/usr/bin/python
import sys,os
global newb = 0.5
os.chdir(r'C:\Users\Documents\python')
execfile('CreateMosFile.py')
os.popen(r"C:\OpenModelica1.4.5\bin\omc.exe CalledbyPython.mos").read()
execfile('RetrResult.py')

# file: 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()

# file: 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()
```

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

Chapter 14

OpenModelica Python Interface and PySimulator

This chapter describes the OpenModelica Python integration facilities.

- OMPython – the OpenModelica Python scripting interface, see Section 14.1.
- PySimulator – a Python package that provides simulation and post processing/analysis tools integrated with OpenModelica, see Section 14.2.

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 architectured 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 v1.0 is not a standalone package, it depends upon the OpenModelica installation.

OMPython v1.0 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 the latest OpenModelica (version 1.9.1) installation.

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 Using OMPython

The third party library of OMPython can be created by changing directory to,

```
OpenModelicaInstallationDirectory/share/omc/scripts/PythonInterface/
```

And running the command,

```
python setup.py install
```

This will install the OMPython library into your Python's third party libraries directory. Now OMPython can be imported and used within Python.

14.1.2.1 Test Commands

To test the command outputs, simply import the OMPython library from Python and execute the run() method of OMPython. The module allows you to interactively send commands to the OMC server and display their output.

For example:

```
C:\>python
>>> import OMPython
>>> OMPython.run()
```

Full example:

```
C:\>python
>>> import OMPython
>>> OMPython.run()
>>loadModel(Modelica)
True
```

14.1.2.2 Import As Library

To use the module from within another python program, simply import OMPython 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
import OMPython
cmds =
    ["loadModel(Modelica)",
    "model test end test;",
    "loadFile(\"C:/OpenModelica1.9.1/testmodels/BouncingBall.mo\")",
    "getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)",
    "getElementsInfo(Modelica.Electrical.Analog.Basic.Resistor)",
    "simulate(BouncingBall)",
    "plot(h)"]
for cmd in cmds:
    answer = OMPython.execute(cmd)
    print "\nResult:\n%s"%answer
```

14.1.2.3 Retrieve results from nested dictionaries

Once the result is available from the `OMPython.execute()`, the `OMPython.get()` method can be used to retrieve and use specific values inside the dictionaries by simply querying the result dictionary with a string of nested dictionary names (keys).

The query should define the complete nested structure of the dictionary starting from its root.

Syntax:

```
OMPython.get(dict, "dotted.dict.structure")
```

For example:

```
OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
```

Full example:

```
#test.py
import OMPython
OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
print "result of get is \n%s" %inner
```

14.1.2.4 Set values to nested dictionaries

New dictionaries can be added to the existing nested dictionary by using the `OMPyhton.set()` method.

Syntax:

```
OMPython.set(dict,"new.dotted.dict.structure", new_value)
```

Note: `new_value` can be any of the Python supported datatypes.

For example:

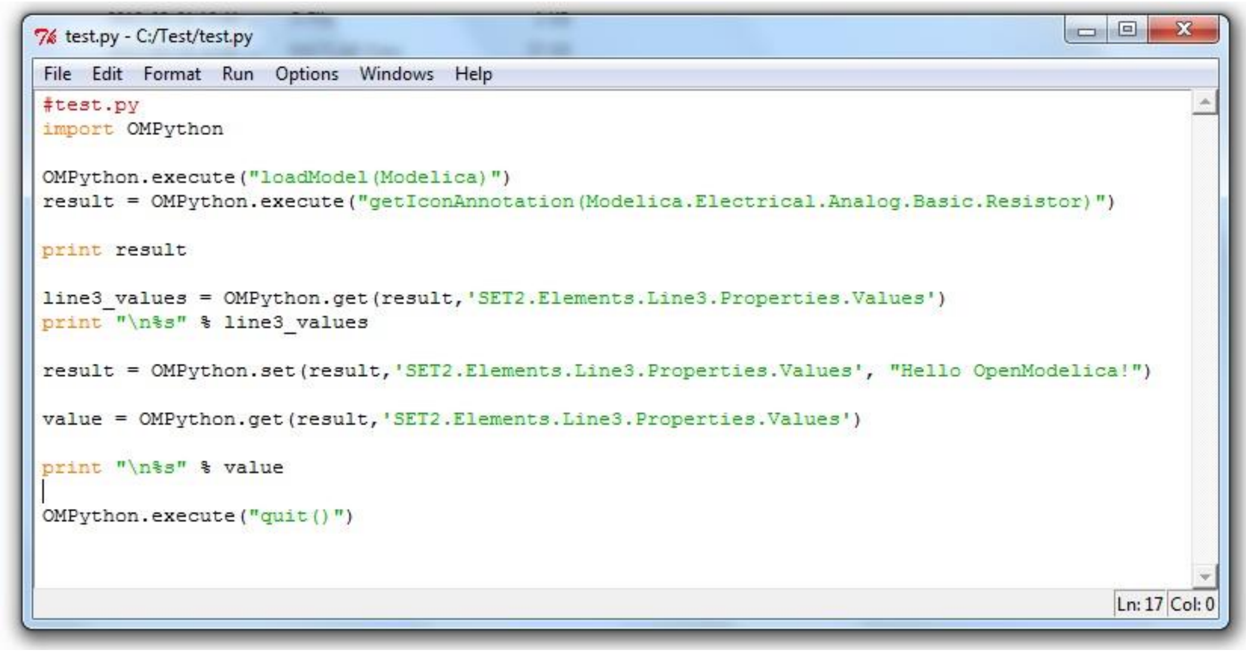
```
OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")
value = OMPython.set(result,"SET2.Elements.Line1.Properties.NEW", 1e-05)
```

The `OMPython.set()` method does not append dictionaries to the existing nest but creates new ones inside the existing. Design your query such that you don't overwrite the dictionaries if you don’t intend to.

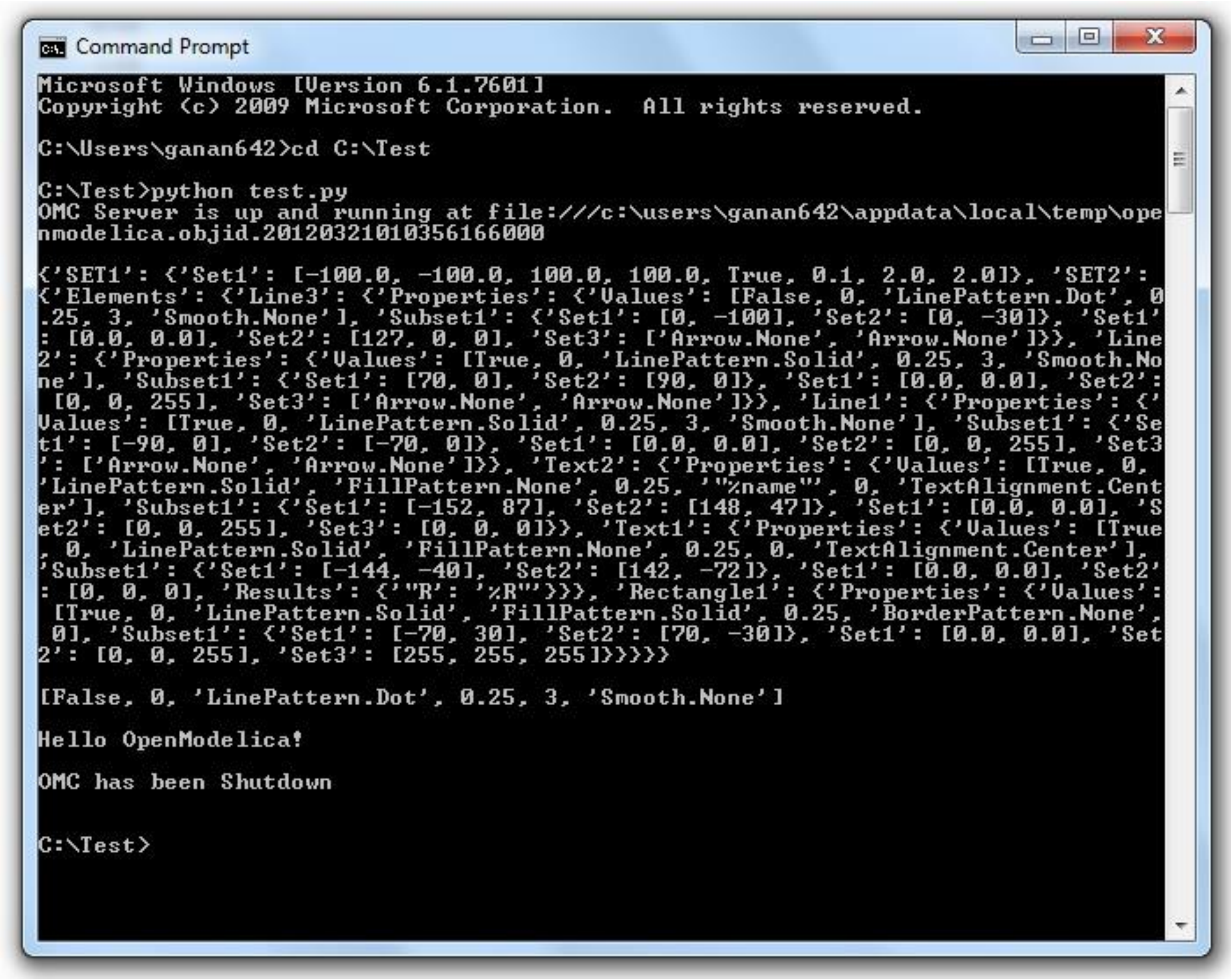
Full example:

```
#test.py
import OMPython
OMPython.execute("loadModel(Modelica)")
result =
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
print "result of get is \n%s" %inner
value = OMPython.set(result,"SET2.Elements.Line1.Properties.NEW", [1,2,3])
print "Result:: \n%s" % OMPython.get(value,'SET2.Elements.Line1.Properties.NEW')
```

14.1.2.5 Example


test.py - C:/Test/test.py
File Edit Format Run Options Windows Help
Ln: 17 Col: 0

```
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
result = OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")

print result

line3_values = OMPython.get(result,'SET2.Elements.Line3.Properties.Values')
print "\n%s" % line3_values

result = OMPython.set(result,'SET2.Elements.Line3.Properties.Values', "Hello OpenModelica!")

value = OMPython.get(result,'SET2.Elements.Line3.Properties.Values')

print "\n%s" % value

OMPython.execute("quit()")
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321010356166000

{'SET1': {'Set1': [-100.0, -100.0, 100.0, 100.0, True, 0.1, 2.0, 2.0]}, 'SET2':
{'Elements': {'Line3': {'Properties': {'Values': [False, 0, 'LinePattern.Dot', 0
.25, 3, 'Smooth.None'], 'Subset1': {'Set1': [0, -100], 'Set2': [0, -30]}, 'Set1'
: [0.0, 0.0], 'Set2': [127, 0, 0], 'Set3': ['Arrow.None', 'Arrow.None']}}, 'Line
2': {'Properties': {'Values': [True, 0, 'LinePattern.Solid', 0.25, 3, 'Smooth.No
ne'], 'Subset1': {'Set1': [70, 0], 'Set2': [90, 0]}, 'Set1': [0.0, 0.0], 'Set2':
 [0, 0, 255], 'Set3': ['Arrow.None', 'Arrow.None']}}, 'Line1': {'Properties': {'
Values': [True, 0, 'LinePattern.Solid', 0.25, 3, 'Smooth.None'], 'Subset1': {'Se
t1': [-90, 0], 'Set2': [-70, 0]}, 'Set1': [0.0, 0.0], 'Set2': [0, 0, 255], 'Set3
': ['Arrow.None', 'Arrow.None']}}, 'Text2': {'Properties': {'Values': [True, 0,
'LinePattern.Solid', 'FillPattern.None', 0.25, '"%name"', 0, 'TextAlignment.Cent
er'], 'Subset1': {'Set1': [-152, 87], 'Set2': [148, 47]}, 'Set1': [0.0, 0.0], 'S
et2': [0, 0, 255], 'Set3': [0, 0, 0]}}, 'Text1': {'Properties': {'Values': [True
, 0, 'LinePattern.Solid', 'FillPattern.None', 0.25, 0, 'TextAlignment.Center'],
'Subset1': {'Set1': [-144, -40], 'Set2': [142, -72]}, 'Set1': [0.0, 0.0], 'Set2'
: [0, 0, 0], 'Results': {'"R': '%R"'}}}, 'Rectangle1': {'Properties': {'Values':
 [True, 0, 'LinePattern.Solid', 'FillPattern.Solid', 0.25, 'BorderPattern.None',
 0], 'Subset1': {'Set1': [-70, 30], 'Set2': [70, -30]}, 'Set1': [0.0, 0.0], 'Set
2': [0, 0, 255], 'Set3': [255, 255, 255]}}}}}

[False, 0, 'LinePattern.Dot', 0.25, 3, 'Smooth.None']

Hello OpenModelica!

OMC has been Shutdown


C:\Test>
```

14.1.3 Implementation

14.1.3.1 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.

14.1.3.2 Parser Implementation

Since the results of OMC are retrieved in a String format over CORBA, some housekeeping has to be done to make sure the results are usable in Python easily.

The Parser is designed to do the following,

- Analyze the result string for categorical data.
- Group each category under a category name.
- Type cast the data within these categories.
- Build a suitable data structure to hold these data so that the results can be easily accessed and used.

Understanding the Parsed output

Each command in OpenModelica produces a result that can be categorized according to the statistics of the pattern of data presented in the text. Grammar based parsers were found to be tedious to use because of the complexity of the patterns of data.

The parser just type casts the data without "curly brackets" to the appropriate data type and displays it as the result.

For example:

```
>> getVectorizationLimit()
<< 20
>> getNthInheritedClass(Modelica.Electrical.Analog.Basic.Resistor,1)
<< Modelica.Electrical.Analog.Interfaces.OnePort
```

However, multiple data types packed within a pair of quotations is always treated as a full string.

For example:

```
>> getModelicaPath()
<< "C:/OpenModelica1.8.0/lib/omlibrary"
```

The Dictionary data type in Python

Dictionaries are found to be a useful datatype to pack data with different datatypes. Dictionaries in Python are indexed by Keys unlike the sequences, which are indexed by a range of numbers.

It is best to think of dictionaries as an unordered set of key:value pairs, with the requirement that the keys are always unique. The common operation on dictionaries is to store a value associated with a key and retrieve the value using the key. This provides us the flexibility of creating keys at runtime and accessing these values using their keys later. All data within the dictionary are stored inside a named dictionary. An empty dictionary is represented by a pair of braces {}.

From the reply of the OMC, the complicated result strings are usually the ones found within the curly braces, in order to make a meaningful categorization of the data within these brackets and to avoid the potential complexities due to creating Dynamic variables, we introduce the following notations that can be used within dictionaries,

- SET
- Set
- Subset
- Element
- Results
- Values

SET

A SET (note the capital letters) is used to group data that belong to the first set of balanced curly brackets. According to the needed semantics of the results, a SET can contain Sets, Subsets, Elements, Values and Results. A SET can also be empty, denoted by {}. The SETs are named with an increasing index starting from 1(one). This feature was planned to eliminate the need for dynamic variable creation and having duplicate Keys. The SET belongs within the dictionary, result.

For example:

```
>> strtok("abcbdef","b")
<< {'SET1': {'Values': ['"a","c","def"']}}
```

The command strtok tokenizes the string "abcbdef" at every occurrence of b and produces a SET with values "a", "c", "def".

Set

A set is used to group all data within the a SET that is enclosed within a pair of balanced {}s. A Set can contain only Values and Elements. A set can also be empty, it can be depicted as {{}}, the outer brackets compose a SET, the inner brackets are the Set within the SET.

Subset

A Subset is a two-level deep set that is found within a SET. A subset can contain multiple Sets within its enclosure.

For example:

```
{SET1 {Subset1{Set1},{Set2},{Set3}}}
```

Element

Elements are the data which are grouped within a pair of Parenthesis ( ). As observed from the results string, elements have an element name that describes the data within them, so elements can be grouped by their names. Also, many elements have the same names, so they are indexed by increasing numbers starting from 1(one). Elements have the special property of having one or more Sets and Subsets within them. However, they are in turn enclosed within the SET.

For example:

```
>> getClassAttributes(test.mymodel)
<< {'SET1': {'Elements': {'rec1': {'Properties': {'Results': {'comment': None,
'res
triction': 'MODEL', 'startLine': 1, 'partial': False, 'name': '"mymodel"', 'enca
psulated': False, 'startColumn': 14, 'readonly': '"writable"', 'endColumn': 69,
'file': '"<interactive>"', 'endLine': 1, 'final': False}}}}}}
```

The result contains, a SET with a Element named rec1 which has Properties which are Results (see below) of the element.

Results

Data that is related by the assignment operator "=", within the SETs are denoted as Results. These assignments cannot be assigned to their actual values unless they are related by a Name = Value relationship. So, they form the sub-dictionary called Results within the Element (for example). Then these values can be related by the key:value pair relationship.

For example:

```
>> getClassAttributes(test.mymodel)
<< {'SET1': {'Elements': {'rec1': {'Properties': {'Results': {'comment': None,
'res
triction': 'MODEL', 'startLine': 1, 'partial': False, 'name': '"mymodel"', 'enca
psulated': False, 'startColumn': 14, 'readonly': '"writable"', 'endColumn': 69,
'file': '"<interactive>"', 'endLine': 1, 'final': False}}}}}}
```

Values

Data within any or all of SETs, Sets, Elements and Subsets that are not assignments and separated by commas are grouped together into a list called "Values". The Values list may also be empty, due to Python's representation of a null string "" as {}. Although a Null string is still a Null value, sometimes it is possible to observe data grouped into Values to look like Sets within the Values list.

For example:

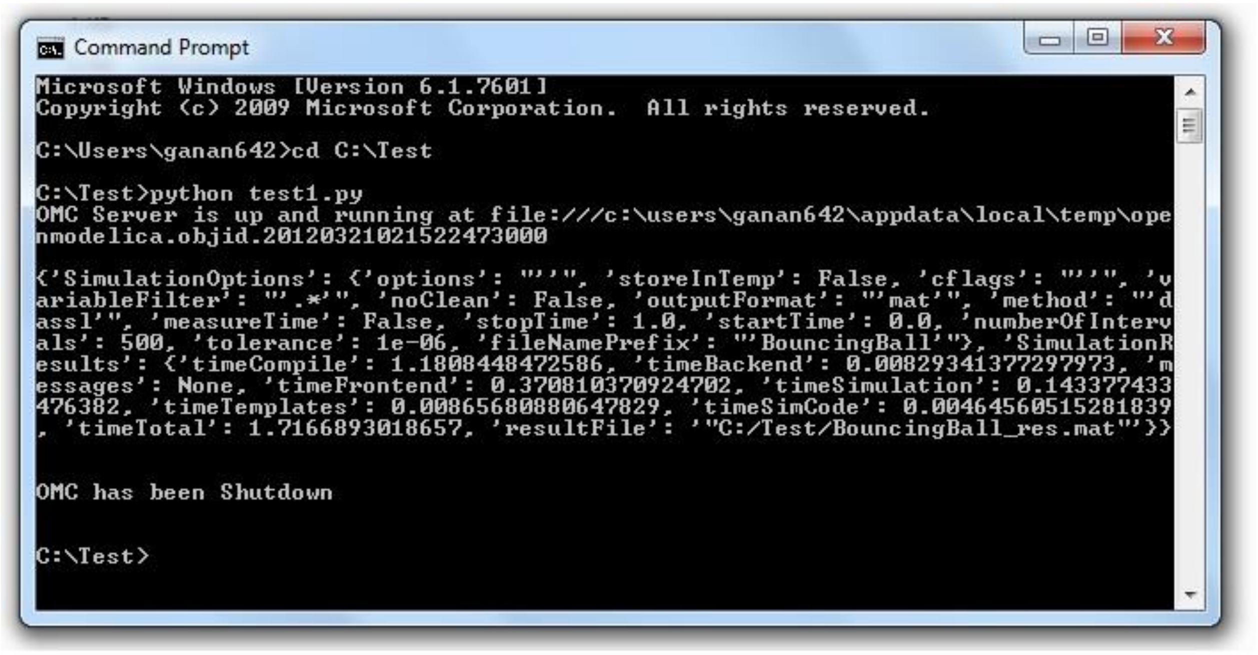
```
>> getNthConnection(Modelica.Electrical.Analog.Examples.ChuaCircuit,2)
<< {'SET1': {'Set1': ['G.n', 'Nr.p', {}]}}
```

14.1.3.3 The Simulation Results

The `simulate()` command produces output that has no SET or Set data in it. Instead, for simplicity, it has two dictionaries namely, SimulationResults and SimulationOptions within the result dictionary.

For example:

```
OMPython.execute("simulate(BouncingBall)")
```


Command Prompt

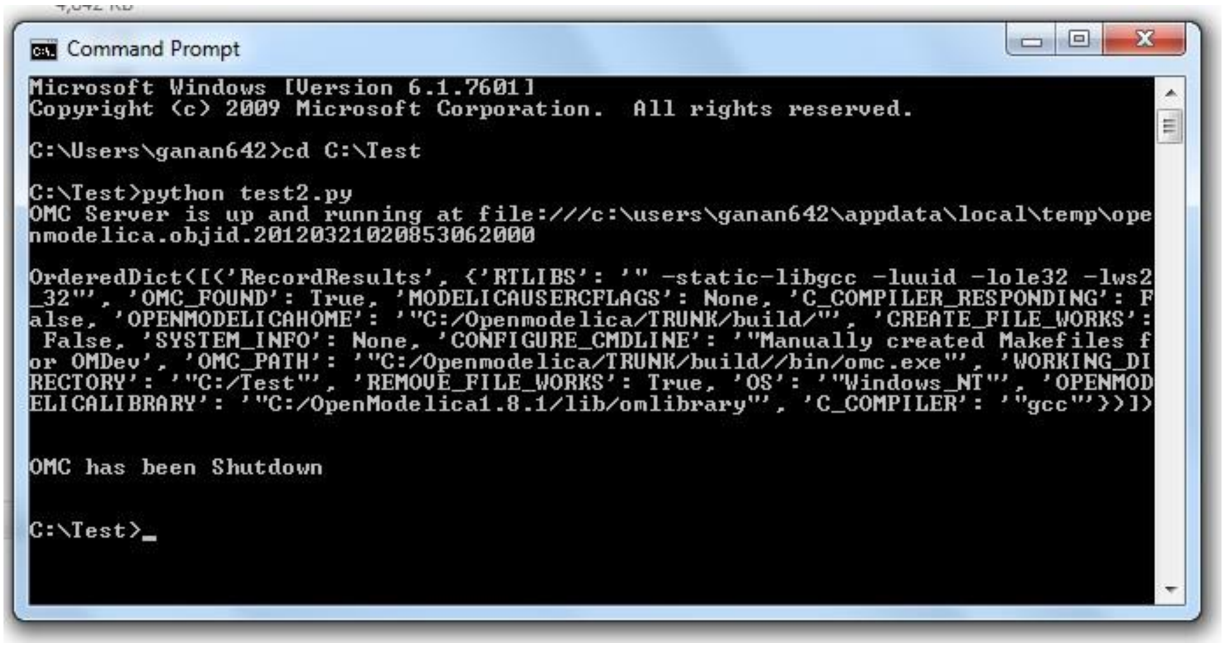
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test1.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321021522473000

{'SimulationOptions': {'options': "''", 'storeInTemp': False, 'cflags': "''", 'v
ariableFilter': "'.*'", 'noClean': False, 'outputFormat': "'mat'", 'method': "'d
assl'", 'measureTime': False, 'stopTime': 1.0, 'startTime': 0.0, 'numberOfInterv
als': 500, 'tolerance': 1e-06, 'fileNamePrefix': "'BouncingBall'"}, 'SimulationR
esults': {'timeCompile': 1.1808448472586, 'timeBackend': 0.00829341377297973, 'm
essages': None, 'timeFrontend': 0.370810370924702, 'timeSimulation': 0.143377433
476382, 'timeTemplates': 0.00865680880647829, 'timeSimCode': 0.00464560515281839
, 'timeTotal': 1.7166893018657, 'resultFile': '"C:/Test/BouncingBall_res.mat"'}}


OMC has been Shutdown


C:\Test>
```

14.1.3.4 Record Construction

The OpenModelica commands that produce output with Record constructs also do not have SET or Set data within them. The results of the output are packed within the RecordResults dictionary.

For example:

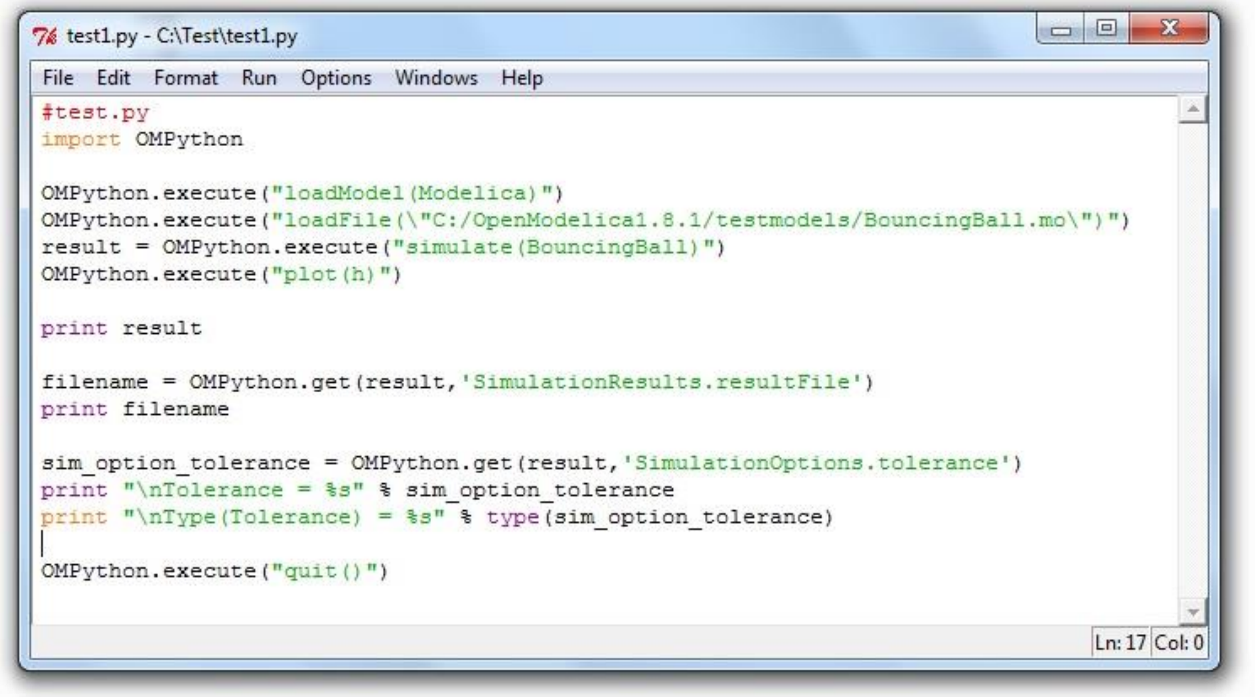
```
OMPython.execute("checkSettings()")
```



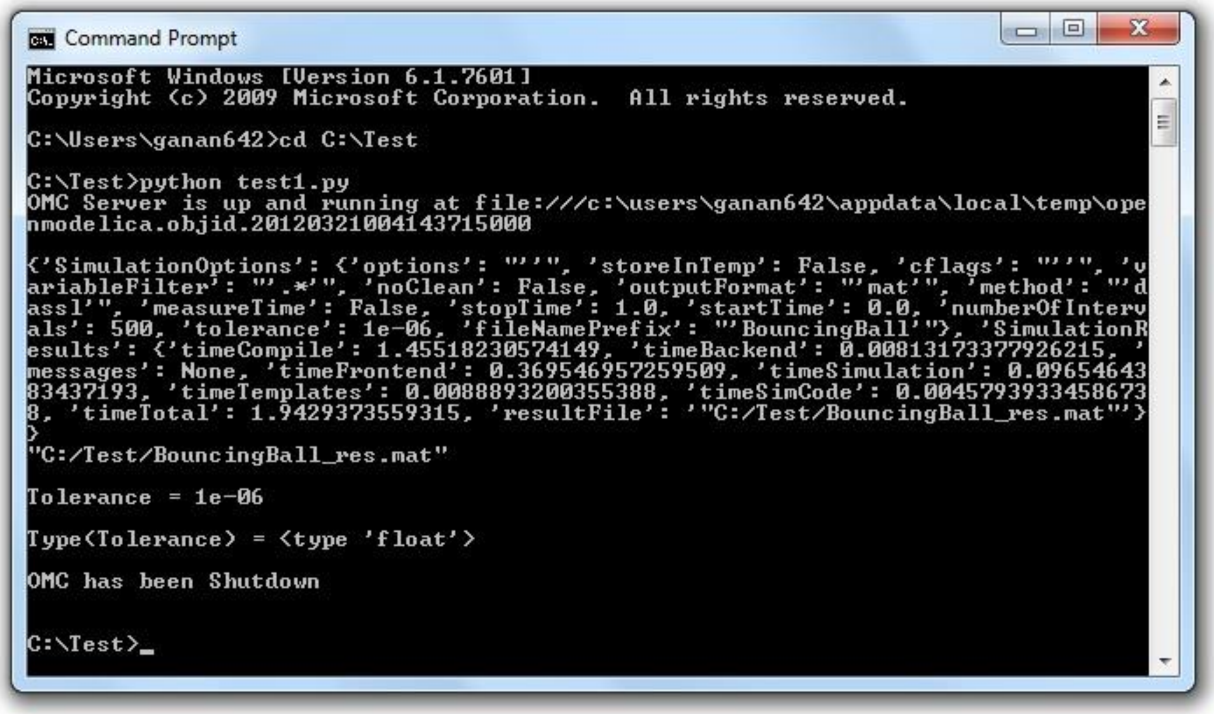
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test2.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321020853062000

OrderedDict([('RecordResults', {'RTLIBS': '" -static-libgcc -luuid -lole32 -lws2
_32"', 'OMC_FOUND': True, 'MODELICAUSERCFLAGS': None, 'C_COMPILER_RESPONDING': F
alse, 'OPENMODELICAHOME': '"C:/Openmodelica/TRUNK/build/"', 'CREATE_FILE_WORKS':
 False, 'SYSTEM_INFO': None, 'CONFIGURE_CMDLINE': '"Manually created Makefiles f
or OMDev', 'OMC_PATH': '"C:/Openmodelica/TRUNK/build//bin/omc.exe"', 'WORKING_DI
RECTORY': '"C:/Test"', 'REMOVE_FILE_WORKS': True, 'OS': '"Windows_NT"', 'OPENMOD
ELICALIBRARY': '"C:/OpenModelica1.8.1/lib/omlibrary"', 'C_COMPILER': '"gcc"'})])


OMC has been Shutdown


C:\Test>_
```

14.1.4 Examples

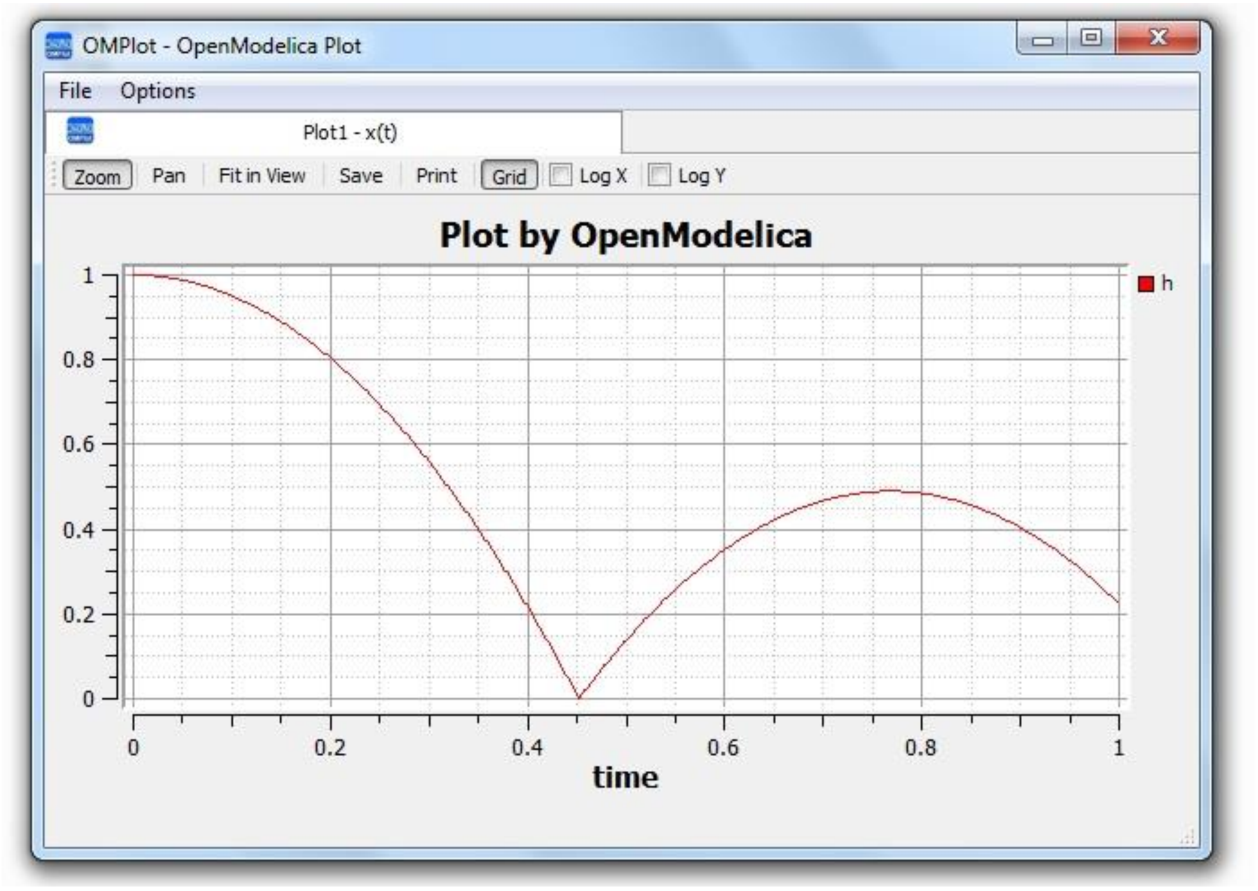
14.1.4.1 Import As Library


test1.py - C:\Test\test1.py
File Edit Format Run Options Windows Help
Ln: 17 Col: 0

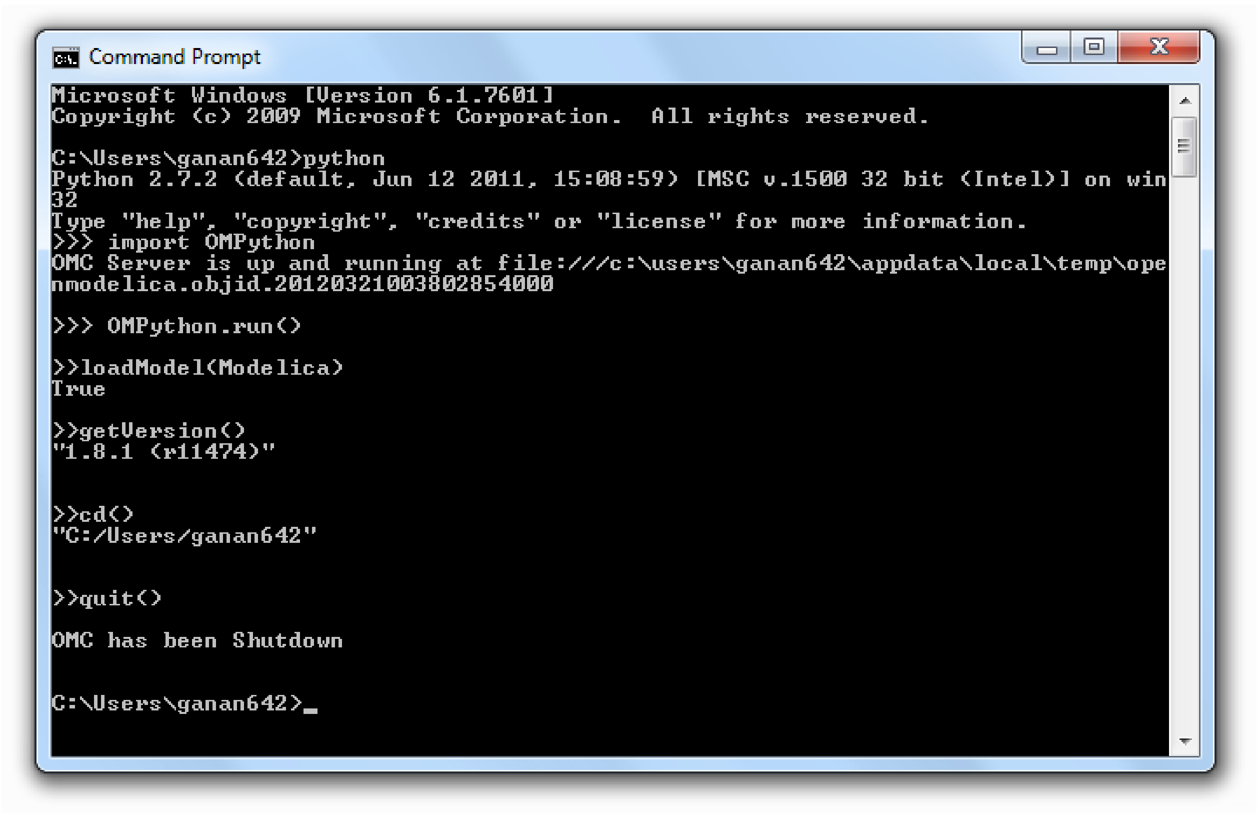
```
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
OMPython.execute("loadFile(\"C:/OpenModelica1.8.1/testmodels/BouncingBall.mo\")")
result = OMPython.execute("simulate(BouncingBall)")
OMPython.execute("plot(h)")

print result

filename = OMPython.get(result,'SimulationResults.resultFile')
print filename

sim_option_tolerance = OMPython.get(result,'SimulationOptions.tolerance')
print "\nTolerance = %s" % sim_option_tolerance
print "\nType(Tolerance) = %s" % type(sim_option_tolerance)

OMPython.execute("quit()")
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test1.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321004143715000

{'SimulationOptions': {'options': "''", 'storeInTemp': False, 'cflags': "''", 'v
ariableFilter': "'.*'", 'noClean': False, 'outputFormat': "'mat'", 'method': "'d
assl'", 'measureTime': False, 'stopTime': 1.0, 'startTime': 0.0, 'numberOfInterv
als': 500, 'tolerance': 1e-06, 'fileNamePrefix': "'BouncingBall'"}, 'SimulationR
esults': {'timeCompile': 1.45518230574149, 'timeBackend': 0.00813173377926215, '
messages': None, 'timeFrontend': 0.369546957259509, 'timeSimulation': 0.09654643
83437193, 'timeTemplates': 0.0088893200355388, 'timeSimCode': 0.00457939334586673
8, 'timeTotal': 1.9429373559315, 'resultFile': '"C:/Test/BouncingBall_res.mat"'}
}
"C:/Test/BouncingBall_res.mat"

Tolerance = 1e-06

Type(Tolerance) = <type 'float'>

OMC has been Shutdown


C:\Test>_
```



14.1.4.2 Test Commands



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>python
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> import OMPython
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321003802854000

>>> OMPython.run()

>>loadModel(Modelica)
True

>>getVersion()
"1.8.1 (r11474)"


>>cd()
"C:/Users/ganan642"


>>quit()

OMC has been Shutdown


C:\Users\ganan642>_
```

14.1.5 API – List of Commands

The following table contains brief descriptions of the API/ commands that are available in the OpenModelica environment, which can be called using Python or Modelica scripting.

Command	Description

simulate	Simulates a model.
	Interface:
	`function simulate`
	`  input TypeName className;`
	`  input Real startTime=0;`
	`  input Real stopTime=1;`
	`  input Integer numberOfIntervals=500;`
	`  input Real tolerance=1e-4;`
	`  input String method=”dassl”; See Appendix C for solver details.`
	`  input String outputFormat=”mat”; {mat, plt, csv, empty}`
	`  input String fileNamePrefix=””;`
	`  input String variableFilter=””;`
	`  input String cflags=””;`
	`  input String simflags=””;`
	`  ouput SimulationResult simRes;`
	`end simulate;`
	SimulationResult:
	`Record SimulationResult`
	`  String resultFile;`
	`  String simulationOptions;`
	`  String messages;`
	`  Real timeFrontend;`
	`  Real timeBackend;`
	`  Real timeSimCode;`
	`  Real timeTemplates;`
	`  Real timeCompile;`
	`  Real timeSimulation;`
	`  Real timeTotal;`
	`End SimulationResult;`

appendEnvironmentVar	Appends a variable to the environment variables list.
	Interface:
	`function appendEnvironmentVar`
	`   input String var;`
	`   input String value;`
	`   output String result "returns \"error\" if the variable could not be appended";`
	`end appendEnvironmentVar;`

basename	Returns the base name (file part) of a file path. Similar to `basename(3)`, but with the safety of Modelica strings.
	Interface:
	`function basename`
	`   input String path;`
	`   output String basename;`
	`end basename;`

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.
	Interface:
	`function cd`
	`   input String newWorkingDirectory = "";`
	`   output String workingDirectory;`
	`end cd;`

checkAllModelsRecursive	Checks all models recursively and returns number of variables and equations.
	Interface:
	`function checkAllModelsRecursive`
	`   input TypeName className;`
	`   input Boolean checkProtected = false "Checks also protected classes if true";`
	`   output String result;`
	`end checkAllModelsRecursive;`

checkModel	Checks a model and returns number of variables and equations.
	Interface:
	`function checkModel`
	`   input TypeName className;`
	`   output String result;`
	`end checkModel;`

checkSettings	Display some diagnostics.
	Interface:
	`function checkSettings`
	`   output CheckSettingsResult result;`
	`end checkSettings;`
	CheckSettingsResult:
	`String OPENMODELICAHOME,OPENMODELICALIBRARY,OMC_PATH;`
	`   Boolean OMC_FOUND;`
	`   String MODELICAUSERCFLAGS,WORKING_DIRECTORY;`
	`   Boolean CREATE_FILE_WORKS,REMOVE_FILE_WORKS;`
	`   String OS, SYSTEM_INFO,SENDDATALIBS,C_COMPILER;`
	`   Boolean C_COMPILER_RESPONDING;`
	`   String CONFIGURE_CMDLINE;`
	`end CheckSettingsResult;`

clear	Clears everything: symboltable and variables.
	Interface:
	`function clear`
	`   output Boolean success;`
	`end clear;`

clearMessages	Clears the error buffer.
	Interface:
	`function clearMessages`
	`   output Boolean success;`
	`end clearMessages;`

clearVariables	Clear all user defined variables.
	Interface:
	`function clearVariables`
	`   output Boolean success;`
	`end clearVariables;`

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.
	Interface:
	`function closeSimulationResultFile`
	`   output Boolean success;`
	`end closeSimulationResultFile;`

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

compareSimulationResults	Compares simulation results.
	Interface:
	`function compareSimulationResults`
	`   input String filename;`
	`   input String reffilename;`
	`   input String logfilename;`
	`   input Real refTol;`
	`   input Real absTol;`
	`   input String[:] vars;`
	`   output String result;`
	`end compareSimulationResults;`

deleteFile	Deletes a file with the given name.
	Interface:
	`function deleteFile`
	`input String fileName;`
	` output Boolean success;`
	`end deleteFile;`

dirname	Returns the directory name of a file path. Similar to dirname(3), but with the safety of Modelica strings.
	Interface:
	`function dirname`
	`   input String path;`
	`   output String dirname;`
	`end dirname;`

dumpXMLDAE	Outputs the DAE system corresponding to a specific model.
	Interface:
	`function dumpXMLDAE`
	`   input TypeName className;`
	`   input String translationLevel = "flat";`
	`   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 Boolean storeInTemp = false;`
	`   output String result[2] "Contents, Message/Filename; why is this an array and not 2 output arguments?";`
	`end dumpXMLDAE;`

echo	echo(false) disables Interactive output, echo(true) enables it again. **Interface:** `function echo`

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

generateCode	The input is a function name for which C-code is generated and compiled into a dll/so.
	Interface:
	`function generateCode`
	`   input TypeName className;`
	`   output Boolean success;`
	`end generateCode;`

generateHeader	**Interface:**
	`function generateHeader`
	`   input String fileName;`
	`   output Boolean success;`
	`end generateHeader;`

generateSeparateCode	**Interface:**
	`function generateSeparateCode`
	`   output Boolean success;`
	`end generateSeparateCode;`

getAlgorithmCount	Counts the number of Algorithm sections in a class.
	Interface:
	`function getAlgorithmCount`
	`   input TypeName class_;`
	`   output Integer count;`
	`end getAlgorithmCount;`

getAlgorithmItemsCount	Counts the number of Algorithm items in a class.
	Interface:
	`function getAlgorithmItemsCount`
	`   input TypeName class_;`
	`   output Integer count;`
	`end getAlgorithmItemsCount;`

getAnnotationCount	Counts the number of Annotation sections in a class.
	Interface:
	`function getAnnotationCount`
	`   input TypeName class_;`
	`   output Integer count;`
	`end getAnnotationCount;`

getAnnotationVersion	Returns the current annotation version.
	Interface:
	`function getAnnotationVersion`
	`   output String annotationVersion;`
	`end getAnnotationVersion;`

getAstAsCorbaString	Print the whole AST on the CORBA format for records, e.g.
	`  record Absyn.PROGRAM`
	`     classes = ...,`
	`     within_ = ...,`
	`     globalBuildTimes = ...`
	`  end Absyn.PROGRAM;`
	Interface:
	`function getAstAsCorbaString`
	`   input String fileName = "<interactive>";`
	`   output String result "returns the string if`

	`fileName is interactive; else it returns ok or error depending on if writing the file succeeded";`
	` end getAstAsCorbaString;`

getClassComment	Returns the class comment.
	Interface:
	` function getClassComment`
	`    input TypeName cl;`
	`    output String comment;`
	` end getClassComment;`

getClassNames	Returns the list of class names defined in the class.
	Interface:
	` 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;`

getClassesInModelicaPath	**Interface:**
	` function getClassesInModelicaPath`
	`    output String classesInModelicaPath;`
	end `getClassesInModelicaPath`;

getCompileCommand	**Interface:**
	` function getCompileCommand`
	`    output String compileCommand;`
	` end getCompileCommand;`

getDocumentationAnnotation	Returns the documentaiton annotation defined in the class.
	Interface:
	` function getDocumentationAnnotation`
	`    input TypeName cl;`
	`    output String out[2] "{info,revision}";`
	` end getDocumentationAnnotation;`

getEnvironmentVar	Returns the value of the environment variable.
	Interface:
	` function getEnvironmentVar`
	`    input String var;`
	`    output String value "returns empty string on failure";`
	` end getEnvironmentVar;`

getEquationCount	Counts the number of Equation sections in a class.
	Interface:
	` function getEquationCount`
	`    input TypeName class_;`
	`    output Integer count;`
	` end getEquationCount;`

getEquationItemsCount	Counts the number of Equation items in a class. **Interface:** ` function getEquationItemsCount`

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

getErrorString	Returns the current error message. `[file.mo:n:n-n:n:b] Error: message`
	Interface:
	```
	function getErrorString
	   output String errorString;
	end getErrorString;
	```

getImportCount	Counts the number of Import sections in a class.
	Interface:
	```
	function getImportCount
	   input TypeName class_;
	   output Integer count;
	end getImportCount;
	```

getInitialAlgorithmCount	Counts the number of Initial Algorithm sections in a class.
	Interface:
	```
	function getInitialAlgorithmCount
	   input TypeName class_;
	   output Integer count;
	end getInitialAlgorithmCount;
	```

getInitialAlgorithmItemsCount	Counts the number of Initial Algorithm items in a class.
	Interface:
	```
	function getInitialAlgorithmItemsCount
	   input TypeName class_;
	   output Integer count;
	end getInitialAlgorithmItemsCount;
	```

getInitialEquationCount	Counts the number of Initial Equation sections in a class.
	Interface:
	```
	function getInitialEquationCount
	   input TypeName class_;
	   output Integer count;
	end getInitialEquationCount;
	```

getInitialEquationItemsCount	Counts the number of Initial Equation items in a class.
	Interface:
	```
	function getInitialEquationItemsCount
	   input TypeName class_;
	   output Integer count;
	end getInitialEquationItemsCount;
	```

getInstallationDirectoryPath	This returns `OPENMODELICAHOME` if it is set; on some platforms the default path is returned if it is not set.
	Interface:
	```
	function getInstallationDirectoryPath
	   output String installationDirectoryPath;
	end getInstallationDirectoryPath;
	```

getLanguageStandard	Returns the current Modelica Language Standard in use.
	Interface:
	```
	function getLanguageStandard
	   output String outVersion;
	end getLanguageStandard;
	```

getMessagesString	see `getErrorString()`

	Interface:
	`function getMessagesString`
	`   output String messagesString;`
	`end getMessagesString;`

getMessagesStringInternal	`{{[file.mo:n:n-n:n:b] Error: message, TRANSLATION, Error, code}}`
	Interface:
	`function getMessagesStringInternal`
	`   output ErrorMessage[:] messagesString;`
	`end getMessagesStringInternal;`

getModelicaPath	Get the Modelica Library Path.
	Interface:
	`function getModelicaPath`
	`   output String modelicaPath;`
	`end getModelicaPath;`

getNoSimplify	Returns true if noSimplify flag is set.
	Interface:
	`function getNoSimplify`
	`   output Boolean noSimplify;`
	`end getNoSimplify;`

getNthAlgorithm	Returns the Nth Algorithm section.
	Interface:
	`function getNthAlgorithm`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthAlgorithm;`

getNthAlgorithmItem	Returns the Nth Algorithm Item.
	Interface:
	`function getNthAlgorithmItem`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthAlgorithmItem;`

getNthAnnotationString	Returns the Nth Annotation section as string.
	Interface:
	`function getNthAnnotationString`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthAnnotationString;`

getNthEquation	Returns the Nth Equation section.
	Interface:
	`function getNthEquation`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthEquation;`

getNthEquationItem	Returns the Nth Equation Item. **Interface:** `function getNthEquationItem`

	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthEquationItem;`

getNthImport	Returns the Nth Import as string.
	Interface:
	`function getNthImport`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String out[3] "{\"Path\",\"Id\",\"Kind\"}";`
	`end getNthImport;`

getNthInitialAlgorithm	Returns the Nth Initial Algorithm section.
	Interface:
	`function getNthInitialAlgorithm`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthInitialAlgorithm;`

getNthInitialAlgorithmItem	Returns the Nth Initial Algorithm Item.
	Interface:
	`function getNthInitialAlgorithmItem`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthInitialAlgorithmItem;`

getNthInitialEquation	Returns the Nth Initial Equation section.
	Interface:
	`function getNthInitialEquation`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthInitialEquation;`

getNthInitialEquationItem	Returns the Nth Initial Equation Item.
	Interface:
	`function getNthInitialEquationItem`
	`   input TypeName class_;`
	`   input Integer index;`
	`   output String result;`
	`end getNthInitialEquationItem;`

getOrderConnections	Returns true if orderConnections flag is set.
	Interface:
	`function getOrderConnections`
	`   output Boolean orderConnections;`
	`end getOrderConnections;`

getPackages	Returns the list of packages defined in the class.
	Interface:
	`function getPackages`
	`   input TypeName class_ = $Code(AllLoadedClasses);`
	`   output TypeName classNames[:];`
	`end getPackages;`

getPlotSilent	Returns true if plotSilent flag is set. **Interface:**

	`function getPlotSilent`
	`   output Boolean plotSilent;`
	`end getPlotSilent;`

getSettings	**Interface:**
	`function getSettings`
	`   output String settings;`
	`end getSettings;`

getShowAnnotations	**Interface:**
	`function getShowAnnotations`
	`   output Boolean show;`
	`end getShowAnnotations;`

getSourceFile	Returns the filename of the class.
	Interface:
	`function getSourceFile`
	`   input TypeName class_;`
	`   output String filename "empty on failure";`
	`end getSourceFile;`

getTempDirectoryPath	Returns the current user temporary directory location.
	Interface:
	`function getTempDirectoryPath`
	`   output String tempDirectoryPath;`
	`end getTempDirectoryPath;`

getVectorizationLimit	**Interface:**
	`function getVectorizationLimit`
	`   output Integer vectorizationLimit;`
	`end getVectorizationLimit;`

getVersion	Returns the version of the Modelica compiler.
	Interface:
	`function getVersion`
	`   input TypeName cl = $Code(OpenModelica);`
	`   output String version;`
	`end getVersion;`

help	Display the OpenModelica help text.
	Interface:
	`function help`
	`   output String helpText;`
	`end help;`

iconv	The `iconv()` function converts one multibyte characters from one character set to another.
	See `man(3) iconv` for more information.
	Interface:
	`function iconv`
	`   input String string;`
	`   input String from;`
	`   input String to = "UTF-8";`
	`   output String result;`
	`end iconv;`

importFMU	Imports the Functional Mockup Unit.
	`Example command:`
	` importFMU("A.fmu");`
	Interface:

	`function importFMU`
	`   input String filename "the fmu file name";`
	`   input String workdir = "./" "The output directory for imported FMU files. <default> will put the files to current working directory.";`
	`   output Boolean success "Returns true on success";`
	`end importFMU;`

instantiateModel	Instantiates the class and returns the flat Modelica code.
	Interface:
	`function instantiateModel`
	`   input TypeName className;`
	`   output String result;`
	`end instantiateModel;`

isModel	Returns true if the given class has restriction model.
	Interface:
	`function isModel`
	`   input TypeName cl;`
	`   output Boolean b;`
	`end isModel;`

isPackage	Returns true if the given class is a package.
	Interface:
	`function isPackage`
	`   input TypeName cl;`
	`   output Boolean b;`
	`end isPackage;`

isPartial	Returns true if the given class is partial.
	Interface:
	`function isPartial`
	`   input TypeName cl;`
	`   output Boolean b;`
	`end isPartial;`

list	Lists the contents of the given class, or all loaded classes.
	Pretty-prints a class definition.
	`Syntax`
	`list(Modelica.Math.sin)`
	`list(Modelica.Math.sin,interfaceOnly=true)`
	Interface:
	`function list`
	`   input TypeName class_ = $Code(AllLoadedClasses);`
	`   input Boolean interfaceOnly = false;`
	`   input Boolean shortOnly = false "only short class definitions";`
	`   output String contents;`
	`end list;`

listVariables	Lists the names of the active variables in the scripting environment.
	Interface:
	`function listVariables`
	`   output TypeName variables[:];`
	`end listVariables;`

loadFile	load file (*.mo) and merge it with the loaded AST. **Interface:** `function loadFile`

	`input String fileName;`
	`output Boolean success;`
	`end loadFile;`

loadFileInteractive	**Interface:**
	`function loadFileInteractive`
	`input String filename;`
	`output TypeName names[:];`
	`end loadFileInteractive;`

loadFileInteractiveQualified	**Interface:**
	`function loadFileInteractiveQualified`
	`input String filename;`
	`output TypeName names[:];`
	`end loadFileInteractiveQualified;`

loadModel	Loads a Modelica library.
	`Syntax`
	`loadModel(Modelica)`
	`loadModel(Modelica,{"3.2"})`
	Interface:
	`function loadModel`
	`input TypeName className;`
	`input String[:] priorityVersion = {"default"};`
	`output Boolean success;`
	`end loadModel;`

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.
	Interface:
	`function loadString`
	`input String data;`
	`input String filename = "<interactive>";`
	`input String encoding = "UTF-8";`
	`output Boolean success;`
	`end loadString;`

parseFile	**Interface:**
	`function parseFile`
	`input String filename;`
	`output TypeName names[:];`
	`end parseFile;`

parseString	**Interface:**
	`function parseString`
	`input String data;`
	`input String filename = "<interactive>";`
	`output TypeName names[:];`
	`end parseString;`

plot	Launches a plot window using OMPlot. Returns true on success. Don't require sendData support.
	Example command sequences:
	`simulate(A);plot({x,y,z});`
	`simulate(A);plot(x, externalWindow=true);`
	`simulate(A,fileNamePrefix="B");`
	`simulate(C);plot(z,"B.mat",legend=false);`
	Interface:

	`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 the last simulation result";`
	`input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`
	`input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`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 diagram. {0,0} will select a suitable range.";`
	`input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`output Boolean success "Returns true on success";`
	`output String[:] result "Returns list i.e {\"_omc_PlotResult\", \"<fileName>\", \"<title>\", \"<legend>\", \"<grid>\", \"<PlotType>\", \"<logX>\", \"<logY>\", \"<xLabel>\", \"<yLabel>\", \"<xRange>\", \"<yRange>\", \"<PlotVariables>\"}";`
	`end plot;`

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);`
	`simulate(A,fileNamePrefix="B");`
	`simulate(C);`
	`plotAll(x,"B.mat");`
	Interface:
	`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 the last simulation result";`
	`input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`

	`    input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`    input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`    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 diagram. {0,0} will select a suitable range.";`
	`    input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`    output Boolean success "Returns true on success";`
	`    output String[:] result "Returns list i.e {\"_omc_PlotResult\", \"<fileName>\", \"<title>\", \"<legend>\", \"<grid>\", \"<PlotType>\", \"<logX>\", \"<logY>\", \"<xLabel>\", \"<yLabel>\", \"<xRange>\", \"<yRange>\", \"<PlotVariables>\"}";`
	` end plotAll;`

plotParametric	Launches a plotParametric window using OMPlot. Returns true on success. Example command sequences:
	`  simulate(A);plotParametric2(x,y);`
	`  simulate(A);plotParametric2(x,y, externalWindow=true);`
	Interface:
	` 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 the last simulation result";`
	`    input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`    input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`    input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`
	`    input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`    input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`    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 diagram. {0,0} will select a suitable range.";`
	`    input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`    output Boolean success "Returns true on success";`
	`    output String[:] result "Returns list i.e {\"_omc_PlotResult\", \"<fileName>\", \"<title>\", \"<legend>\", \"<grid>\", \"<PlotType>\", \"<logX>\", \"<logY>\", \"<xLabel>\", \"<yLabel>\", \"<xRange>\",`

	`\"<yRange>\",  \"<PlotVariables>\"}";`
	` end plotParametric;`

plotParametric2	Plots the y-variables as a function of the x-variable.
	Example command sequences:
	`  simulate(A);`
	`  plotParametric2(x,y);`
	`  simulate(A,fileNamePrefix="B");`
	`  simulate(C);`
	`  plotParametric2(x,{y1,y2,y3},"B.mat");`
	Interface:
	` function plotParametric2`
	`    input VariableName xVariable;`
	`    input VariableNames yVariables;`
	`    input String fileName = "<default>";`
	`    output Boolean success "Returns true on success";`
	` end plotParametric2;`

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.
	Interface:
	` function readFile`
	`    input String fileName;`
	`    output String contents;`
	` end readFile;`

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?
	Interface:
	` function readFileNoNumeric`
	`    input String fileName;`
	`    output String contents;`
	` end readFileNoNumeric;`

readFilePostprocessLineDirective	Searches lines for the #modelicaLine directive. If it is found, all lines up until the next #modelicaLine or #endModelicaLine are put on a single file, following a #line linenumber "filename" line. This causes GCC to output an executable that we can set breakpoints in and debug.
	Note: You could use a stack to keep track of start/end of #modelicaLine and match them up. But this is not really desirable since that will cause extra breakpoints for the same line (you would get breakpoints before and after each case if you break on a match-expression, etc).
	Interface:
	` function readFilePostprocessLineDirective`
	`    input String fileName;`
	`    output String out;`
	` end readFilePostprocessLineDirective;`

readFileShowLineNumbers	Prefixes each line in the file with <n>:, where n is the line number.
	Note: Scales O(n^2)
	Interface:
	` function readFileShowLineNumbers`
	`    input String fileName;`
	`    output String out;`
	` end readFileShowLineNumbers;`

readSimulationResult	Reads a result file, returning a matrix corresponding to the variables and size given.
	Interface:
	`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 function fails";`
	`   output Real result[:,:];`
	`end readSimulationResult;`

readSimulationResultSize	The number of intervals that are present in the output file.
	Interface:
	`function readSimulationResultSize`
	`   input String fileName;`
	`   output Integer sz;`
	`end readSimulationResultSize;`

readSimulationResultVars	Returns the variables in the simulation file; you can use val() and plot() commands using these names.
	Interface:
	`function readSimulationResultVars`
	`   input String fileName;`
	`   output String[:] vars;`
	`end readSimulationResultVars;`

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
	Interface:
	`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;`

regexBool	Returns true if the string matches the regular expression.
	Interface:
	`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;`

regularFileExists	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.
	Interface:
	`function regularFileExists`
	`   input String fileName;`
	`   output Boolean exists;`
	`end regularFileExists;`

reopenStandardStream	Interface:
	`function reopenStandardStream`
	`   input StandardStream _stream;`
	`   input String filename;`
	`   output Boolean success;`
	`end reopenStandardStream;`

runScript	Runs the mos-script specified by the filename.
	Interface:
	`function runScript`
	`   input String fileName "*.mos";`
	`   output String result;`
	`end runScript;`

Save	**Interface:**
	`function save`
	`   input TypeName className;`
	`   output Boolean success;`
	`end save;`

saveAll	Save the entire loaded AST to file.
	Interface:
	`function saveAll`
	`   input String fileName;`
	`   output Boolean success;`
	`end saveAll;`

saveModel	Save class definition in a file.
	Interface:
	`function saveModel`
	`   input String fileName;`
	`   input TypeName className;`
	`   output Boolean success;`
	`end saveModel;`

saveTotalModel	Save total class definition into file of a class.
	`Inputs: String fileName; TypeName className`
	`Outputs: Boolean res;`
	Interface:
	`function saveTotalModel`
	`   input String fileName;`
	`   input TypeName className;`
	`   output Boolean success;`
	`end saveTotalModel;`

saveTotalSCode	**Interface:**
	`function saveTotalSCode`
	`   input String fileName;`

	`   input TypeName className;`
	`   output Boolean success;`
	`end saveTotalSCode;`

setAnnotationVersion	Sets the annotation version.
	Interface:
	`function setAnnotationVersion`
	`   input String annotationVersion;`
	`   output Boolean success;`
	`end setAnnotationVersion;`

setCXXCompiler	**Interface:**
	`function setCXXCompiler`
	`   input String compiler;`
	`   output Boolean success;`
	`end setCXXCompiler;`

setClassComment	Sets the class comment.
	Interface:
	`function setClassComment`
	`   input TypeName class_;`
	`   input String filename;`
	`   output Boolean success;`
	`end setClassComment;`

setCommandLineOptions	The input is a regular command-line flag given to OMC, e.g. +d=failtrace or +g=MetaModelica.
	Interface:
	`function setCommandLineOptions`
	`   input String option;`
	`   output Boolean success;`
	`end setCommandLineOptions;`

setCompileCommand	**Interface:**
	`function setCompileCommand`
	`   input String compileCommand;`
	`   output Boolean success;`
	`end setCompileCommand;`

setCompiler	**Interface:**
	`function setCompiler`
	`   input String compiler;`
	`   output Boolean success;`
	`end setCompiler;`

setCompilerFlags	**Interface:**
	`function setCompilerFlags`
	`   input String compilerFlags;`
	`   output Boolean success;`
	`end setCompilerFlags;`

setCompilerPath	**Interface:**
	`function setCompilerPath`
	`   input String compilerPath;`
	`   output Boolean success;`
	`end setCompilerPath;`

setDebugFlags	example input: failtrace,-noevalfunc **Interface:** `function setDebugFlags`

	`   input String debugFlags;`
	`   output Boolean success;`
	`end setDebugFlags;`

setEnvironmentVar	**Interface:**
	`function setEnvironmentVar`
	`   input String var;`
	`   input String value;`
	`   output Boolean success;`
	`end setEnvironmentVar;`

setIndexReductionMethod	example input: dummyDerivative
	Interface:
	`function setIndexReductionMethod`
	`   input String method;`
	`   output Boolean success;`
	`end setIndexReductionMethod;`

setInstallationDirectoryPath	Sets the `OPENMODELICAHOME` environment variable. Use this method instead of `setEnvironmentVar`.
	Interface:
	`function setInstallationDirectoryPath`
	`   input String installationDirectoryPath;`
	`   output Boolean success;`
	`end setInstallationDirectoryPath;`

setLanguageStandard	Sets the Modelica Language Standard.
	Interface:
	`function setLanguageStandard`
	`   input String inVersion;`
	`   output Boolean success;`
	`end setLanguageStandard;`

setLinker	**Interface:**
	`function setLinker`
	`   input String linker;`
	`   output Boolean success;`
	`end setLinker;`

setLinkerFlags	**Interface:**
	`function setLinkerFlags`
	`   input String linkerFlags;`
	`   output Boolean success;`
	`end setLinkerFlags;`

setModelicaPath	See `loadModel()` for a description of what the `MODELICAPATH` is used for.
	Interface:
	`function setModelicaPath`
	`   input String modelicaPath;`
	`   output Boolean success;`
	`end setModelicaPath;`

setNoSimplify	Sets the noSimplify flag.
	Interface:
	`function setNoSimplify`
	`   input Boolean noSimplify;`
	`   output Boolean success;`
	`end setNoSimplify;`

setOrderConnections	Sets the orderConnection flag.
	Interface:
	`function setOrderConnections`
	`   input Boolean orderConnections;`
	`   output Boolean success;`
	`end setOrderConnections;`

setPastOptModules	example input: lateInline,inlineArrayEqn,removeSimpleEquations
	Interface:
	`function setPastOptModules`
	`   input String modules;`
	`   output Boolean success;`
	`end setPastOptModules;`

setPlotCommand	**Interface:**
	`function setPlotCommand`
	`   input String plotCommand;`
	`   output Boolean success;`
	`end setPlotCommand;`

setPlotSilent	Sets the plotSilent flag.
	Interface:
	`function setPlotSilent`
	`   input Boolean silent;`
	`   output Boolean success;`
	`end setPlotSilent;`

setPreOptModules	example input: removeFinalParameters,removeSimpleEquations,expandDerOperator
	Interface:
	`function setPreOptModules`
	`   input String modules;`
	`   output Boolean success;`
	`end setPreOptModules;`

setShowAnnotations	**Interface:**
	`function setShowAnnotations`
	`   input Boolean show;`
	`   output Boolean success;`
	`end setShowAnnotations;`

setSourceFile	**Interface:**
	`function setSourceFile`
	`   input TypeName class_;`
	`   input String filename;`
	`   output Boolean success;`
	`end setSourceFile;`

setTempDirectoryPath	**Interface:**
	`function setTempDirectoryPath`
	`   input String tempDirectoryPath;`
	`   output Boolean success;`
	`end setTempDirectoryPath;`

setVectorizationLimit	**Interface:**
	`function setVectorizationLimit`
	`   input Integer vectorizationLimit;`
	`   output Boolean success;`
	`end setVectorizationLimit;`

solveLinearSystem	Solve A*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.
	Interface:
	`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;`

strictRMLCheck	Checks if any loaded function.
	Interface:
	`function strictRMLCheck`
	`   output String message "empty if there was no`
	`problem";`
	`end strictRMLCheck;`

stringReplace	**Interface:**
	`function stringReplace`
	`   input String str;`
	`   input String source;`
	`   input String target;`
	`   output String res;`
	`end stringReplace;`

Strtok	Splits the strings at the places given by the token, for example:
	`strtok("abcbdef","b") => {"a","c","def"}`
	Interface:
	`function strtok`
	`   input String string;`
	`   input String token;`
	`   output String[:] strings;`
	`end strtok;`

System	Similar to `system(3)`. Executes the given command in the system shell.
	Interface:
	`function system`
	`   input String callStr "String to call: bash -c`
	`$callStr";`
	`   output Integer retval "Return value of the system`
	`call; usually 0 on success";`
	`end system;`

translateGraphics	**Interface:**
	`function translateGraphics`
	`   input TypeName className;`
	`   output String result;`
	`end translateGraphics;`

typeNameString	**Interface:**
	`function typeNameString`
	`   input TypeName cl;`
	`   output String out;`
	`end typeNameString;`

typeNameStrings	**Interface:**

	```
	function typeNameStrings
	   input TypeName cl;
	   output String out[:];
	end typeNameStrings;
	```

typeOf	**Interface:**
	```
	function typeOf
	   input VariableName variableName;
	   output String result;
	end typeOf;
	```

uriToFilename	Handles modelica:// and file:// URI's. The result is an absolute path on the local system. The result depends on the current MODELICAPATH. Returns the empty string on failure.
	Interface:
	```
	function uriToFilename
	   input String uri;
	   output String filename;
	end uriToFilename;
	```

val	Works on the filename pointed to by the scripting variable currentSimulationResult. The result is the value of the variable at a certain time point. 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.
	Interface:
	```
	function val
	   input VariableName var;
	   input Real time;
	   output Real valAtTime;
	end val;
	```

verifyCompiler	**Interface:**
	```
	function verifyCompiler
	   output Boolean compilerWorks;
	end verifyCompiler;
	```

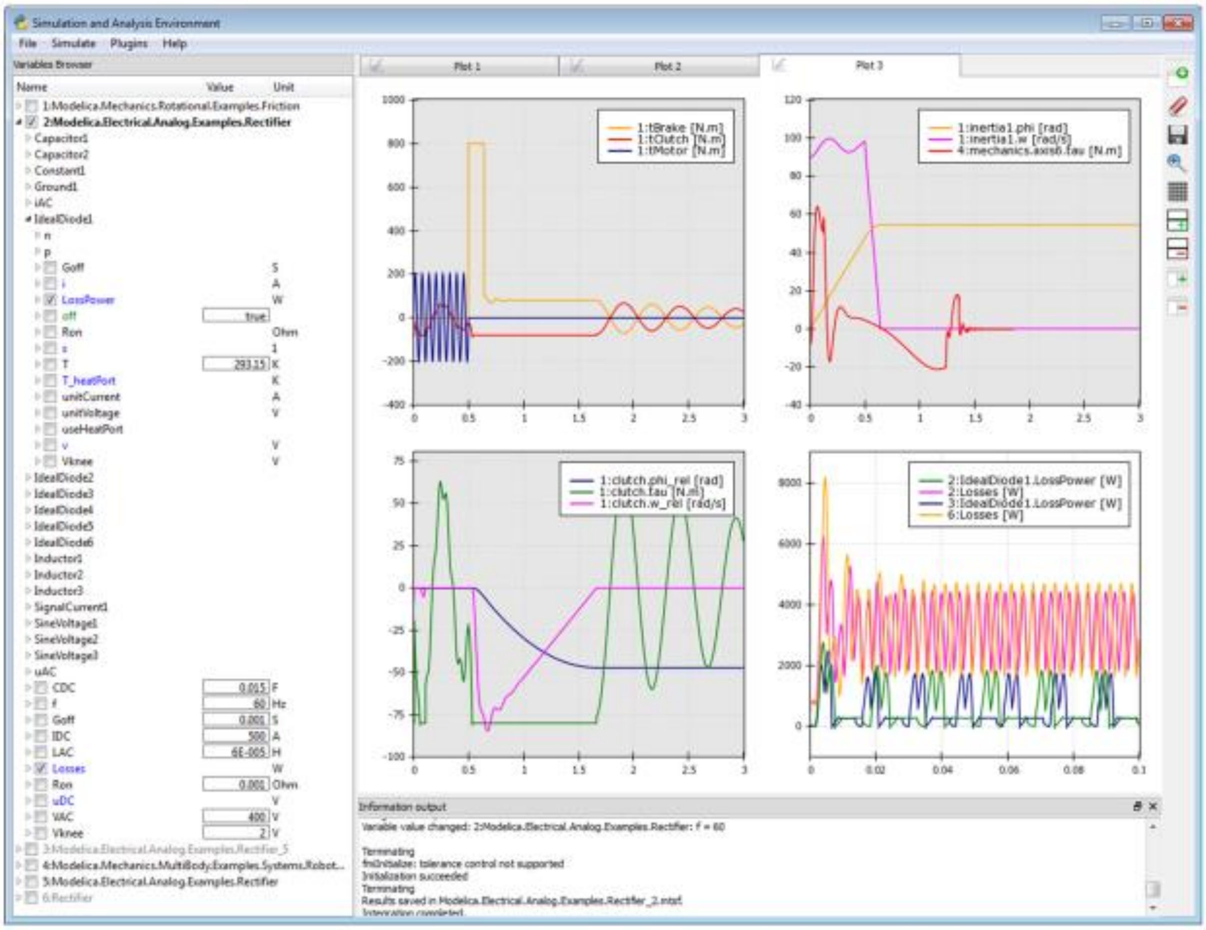
visualize	Uses the 3D visualization package, SimpleVisual.mo, to visualize the model. See chapter 3.4 (3D Animation) of the OpenModelica System Documentation for more details. Writes the visulizations objects into the file "model_name.visualize".
	Example command sequence:
	```
	  simulate(A,outputFormat="mat");
	  visualize(A);
	  visualize(A,"B.mat");
	  visualize(A,"B.mat", true);
	```
	Interface:
	```
	function visualize
	   input TypeName className;
	   input Boolean externalWindow = false "Opens the visualize in a new window";
	   input String fileName = "<default>" "The filename containing the variables. <default> will read the last simulation result";
	   output Boolean success "Returns true on success";
	end visualize;
	```

writeFile	Write the data to file. Returns true on success. **Interface:**

	`function writeFile`
	`   input String fileName;`
	`   input String data;`
	`   input Boolean append = false;`
	`   output Boolean success;`
	`end writeFile;`

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.


Simulation and Analysis Environment
File
Simulate
Plugins
Help
Variables Browser
Name
Value
Unit
1:Modelica.Mechanics.Rotational.Examples.Friction
2:Modelica.Electrical.Analog.Examples.Rectifier
Plot 1
Plot 2
Plot 3
1:tBrake [N.m]
1:tClutch [N.m]
1:tMotor [N.m]
1:inertia1.phi [rad]
1:inertia1.w [rad/s]
4:mechanics.axisB.tau [N.m]
1:clutch.phi_rel [rad]
1:clutch.tau [N.m]
1:clutch.w_rel [rad/s]
2:IdealDiode1.LossPower [W]
2:Losses [W]
3:IdealDiode1.LossPower [W]
6:Losses [W]
3:Modelica.Electrical.Analog.Examples.Rectifier_5
4:Modelica.Mechanics.MultiBody.Examples.Systems.Robot...
5:Modelica.Electrical.Analog.Examples.Rectifier
6:Rectifier
Information output

```
Variable value changed: 2:Modelica.Electrical.Analog.Examples.Rectifier: f = 60

Terminating
fmiInitialize: tolerance control not supported
Initialization succeeded
Terminating
Results saved in Modelica.Electrical.Analog.Examples.Rectifier_2.mtsf
Integration completed
```

Figure 14-1: OMShell screenshot for creating an FMU

Read more about the PySimulator here https://github.com/PySimulator/PySimulator.

Chapter 15

Modelica and Python 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.

15.1.1 OpenModelica Basic Commands

This is a selection of the most common commands related to simulation and plotting:

`checkModel``(className)`	Checks a model and returns number of variables and equations. *Inputs*: `TypeName className;` *Outputs*: `String res;`
`clear``()`	Clears everything in OpenModelica compiler and interpreter workspace: symboltable and variables. *Outputs*: `Boolean res;`
`getMessagesString``()`	Returns the current error message. *Outputs*: `String messagesString`;
`help``()`	Display the OpenModelica help text. *Outputs*: `String helpText`;
`importFMU``(fileName, workDir)`	Imports a Functional Mockup Unit. *Inputs*: `String filename; String workdir "./"` "The output directory for imported FMU files. <default> will put the files to current working directory."; *Outputs*: `Boolean res`;
`instantiateModel``(className)`	Flatten model, resulting in a `.mof` file of flattened Modelica. *Inputs*: `TypeName className;` *Outputs*: `String res;`

`list``(className)`	Print class definition. *Inputs*: `TypeName className;` *Outputs*: `String classDef;`
`listVariables``()`	Print user defined variables. *Outputs*: `VariableName res;`
`loadFile``(fileName)`	Load models from file. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
`loadModel``(className)`	Load the file corresponding to the class, using the Modelica class name to file name mapping to locate the file. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
`plot``(variable, options)`	Plot `variable`, which is a single variable name. *Inputs*: `VariableName variable; String title; Boolean legend; Boolean gridLines; Real xrange[2]` i.e. `{xmin,xmax}; Real yrange[2]` i.e. `{ymin,ymax};` *Outputs*: `Boolean res;`
`plot``(variables, options)`	Plot `variables`, which is a vector of variable names. *Inputs*: `VariableName variables; String title; Boolean legend; Boolean gridLines; Real xrange[2]` i.e. `{xmin,xmax}; Real yrange[2]` i.e. `{ymin,ymax};` *Outputs*: `Boolean res;`
`plotParametric``(variables1, variables2, options)`	Plot each pair of corresponding variables from the vectors of variables `variables1`, `variables2` as a parametric plot. *Inputs*: `VariableName variables1[:]; VariableName variables2[size(variables1,1)]; String title; Boolean legend; Boolean gridLines; Real range[2,2];` *Outputs*: `Boolean res;`
`readFile``(fileName)`	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 outputs or similar. *Inputs*: `String fileName;` *Outputs*: `String content;`
`save``(className)`	Save class definition. *Inputs*: `TypeName className` *Outputs*: `Boolean res;`
`saveAll``(fileName)`	Save the entire loaded AST (Abstract Syntax Tree internal representation of all loaded models) as text to a file. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
`saveModel``(fileName, className)`	Save class definition in a file. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
`saveTotalModel``(fileName, className)`	Save total class definition into file of a class. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
`setDebugFlags``(debugFlags)`	*Inputs*: `String debugFlags;` *Outputs*: `Boolean res;`
`setInitXmlStartValue``(fileName, variableName, startValue, outputFile)`	Sets the parameter start value in the `model_init.xml` file. No need to recompile the model. *Inputs*: `String fileName; String variableName; String startValue; String outputFile;` *Outputs*: `Boolean success;`
`simulate``(className, options)`	Simulate model, optionally setting simulation values. *Inputs*: `TypeName className; Real startTime; Real stopTime; Integer numberOfIntervals;`

	`Real outputInterval; String method;`
	`Real tolerance; Real fixedStepSize;`
	Outputs: `SimulationResult simRes;`

`typeOf``(variableName)`	*Inputs*: `VariableName variableName;` *Outputs*: `String res;`
`val``(var, timepoint)`	Returns the value of the variable at a certain time point. *Inputs*: `VariableName var; Real timepoint;` *Outputs*: `Real valAtTime;`

15.2 OpenModelica System Commands

This is a selection of common OpenModelica commands related to the operating system:

`cd``(dir)`	Change directory. *Inputs*: `String dir;` *Outputs*: `Boolean res;`
`deleteFile``(fileName)`	Deletes a file with the given name. *Inputs*: `String fileName;` *Outputs*: `Boolean res`;
`dirName``(path)`	Returns the directory name of a file path. *Inputs*: `String path;` *Outputs*: `String dirName`;
`runScript``(fileName)`	Executes the script file given as argument. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
`setLanguageStandard``( inVersion)`	Sets the Modelica Language Standard. *Inputs*: `String inVersion;` *Outputs*: `Boolean res;`
`setModelicaPath``( modelicaPath)`	Sets the modelica path. See `loadModel()` for a description of what the `MODELICAPATH` is used for. *Inputs*: `String modelicaPath;` *Outputs*: `Boolean res;`
`system``(fileName)`	Execute system command. *Inputs*: `String fileName;` *Outputs*: `Integer res;`
`writeFile``(fileName, data, optional)`	Write the data to file. Returns `true` on success. *Inputs*: `String fileName; String data; Boolean append;` *Outputs*: `Boolean res;`

15.3 All OpenModelica API Calls

All OpenModelica API commands shown in alphabetic order:

`appendEnvironmentVar``(var, value)`	Appends a variable to the environment variables list. *Inputs:* `String var; String value;` *Outputs:* `String res;`
`baseName``(path)`	Returns the base name (file part) of a file path. *Inputs:* `String path;` *Outputs:* `String basename;`
`cd``(dir)`	Change directory. *Inputs*: `String dir;` *Outputs*: `Boolean res;`

checkAllModelsRecursive(className, protected)	Checks all models recursively and returns number of variables and equations. *Inputs*: `TypeName className; Boolean protected` *Outputs*: `String res;`
checkModel(className)	Checks a model and returns number of variables and equations. *Inputs*: `TypeName className;` *Outputs*: `String res;`
checkSettings()	Display some diagnostics. *Outputs*: `CheckSettingsResult res;`
clear()	Clears everything: symboltable and variables. *Outputs*: `Boolean res;`
clearMessages()	Clears the error buffer. *Outputs*: `Boolean res;`
clearVariables()	Clear all user defined variables. *Outputs*: `Boolean res`;
codeToString(className)	Converts to a string after encoding. *Inputs*: `$Code className;` *Outputs*: `String res`;
compareSimulationResults(fileName, refFileName, logFileName, refTol, absTol,vars)	Compares simulation results. *Inputs*: `String fileName; String refFileName; String logFileName; Real refTol; Real absTol; String[:] vars;` *Outputs*: `String res`;
deleteFile(fileName)	Deletes a file with the given name. *Inputs*: `String fileName;` *Outputs*: `Boolean res`;
dirName(path)	Returns the directory name of a file path. *Inputs*: `String path;` *Outputs*: `String dirName`;
dumpXMLDAE(className)	Outputs the DAE system corresponding to a specific model. *Inputs*: `TypeName className;` *Outputs*: `String res`;
echo(setEcho)	echo (false) disables Interactive output, echo(true) enables it again. *Inputs*: `Boolean setEcho;` *Outputs*: `Boolean newEcho`;
generateCode(className)	Generate C-code and compiled into a dll. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
generateHeader(fileName)	Generates a C header file containing the external C interface of the MetaModelica uniontypes in the loaded files. This C interface is called by the parser to build nodes for the abstract syntax tree. *Inputs*: `String fileName;` *Outputs*: `Boolean res`;
getAlgorithmCount(className)	Counts the number of Algorithm sections in a class. *Inputs*: `TypeName className;` *Outputs*: `Integer count`;
getAlgorithmItemsCount(className)	Counts the number of Algorithm items in a class. *Inputs*: `TypeName className;` *Outputs*: `Integer count`;
getAnnotationCount(className)	Counts the number of Annotation sections in a class. *Inputs*: `TypeName className;` *Outputs*: `Integer count`;
getAnnotationVersion()	Returns the current annotation version. *Outputs*: `String annotationVersion`;
getAstAsCorbaString(fileName)	Print the whole AST on the CORBA format for records. *Inputs*: `String fileName;` *Outputs*: `String res`;
getClassComment(className)	Returns the class comment. *Inputs*: `TypeName className;` *Outputs*: `String comment`;
getClassNames(fileName)	Returns the list of class names defined in the class. *Inputs*: `String fileName;` *Outputs*: `TypeName classNames`;
getClassRestriction(classNam	Returns the type of class. *Inputs*: `TypeName className;`

`e)`	*Outputs*: `String restriction`;
`getClassInformation``(className)`	Returns the list containing the information of the class. *Inputs*: `TypeName className`; *Outputs*: `String information`;
`getClassesInModelicaPath``()`	*Outputs*: `String classesInModelicaPath`;
`getCompileCommand``()`	*Outputs*: `String compileCommand`;
`getIconAnnotation``(className)`	Returns the icon representation of the class. *Inputs*: `TypeName className`; *Outputs*: `String out`;
`getDiagramAnnotation``(className)`	Returns the diagram representation of the class. *Inputs*: `TypeName className`; *Outputs*: `String out`;
`getParameterName``(className)`	Returns the list of parameters present in the class. *Inputs*: `TypeName className`; *Outputs*: `String out`;
`getParameterValue``(className, parameter)`	Returns the parameter value. *Inputs*: `TypeName className; String parameter`; *Outputs*: `String value`;
`getComponentModifierNames``(className)`	Returns the list of component modifiers present in the class. *Inputs*: `TypeName className`; *Outputs*: `String out`;
`getComponentModifierValue``(className, modifier)`	Returns the component modifier value. *Inputs*: `TypeName className; String modifier`; *Outputs*: `String value`;
`getExtendsModifierNames``(className)`	Returns the list of extends modifiers present in the class. *Inputs*: `TypeName className`; *Outputs*: `String out`;
`getExtendsModifierValue``(className, modifier)`	Returns the extends modifier value. *Inputs*: `TypeName className; String modifier`; *Outputs*: `String value`;
`getDocumentationAnnotation``(className)`	Returns the documentation annotation defined in the class. *Inputs*: `TypeName className`; *Outputs*: `String out[2] "{info,revision}"`;
`getEnvironmentVar``(var)`	Returns the value of the environment variable. *Inputs*: `String var`; *Outputs*: `String value`;
`getEquationCount``(className)`	Counts the number of Equation sections in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getEquationItemsCount``(className)`	Counts the number of Equation items in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getErrorString``()`	Returns the current error message. *Outputs*: `String errorString`;
`getImportCount` `(className)`	Counts the number of Import sections in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getInitialAlgorithmCount``(className)`	Counts the number of Initial Algorithm sections in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getInitialAlgorithmItemsCount``(className)`	Counts the number of Initial Algorithm items in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getInitialEquationCount``(className)`	Counts the number of Initial Equation sections in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getInitialEquationItemsCount``(className)`	Counts the number of Initial Equation items in a class. *Inputs*: `TypeName className`; *Outputs*: `Integer count`;
`getInstallationDirectoryPath``()`	Returns `OPENMODELICAHOME` if it is set; on some platforms the default path is returned if it is not set. *Outputs*: `String installationDirectoryPath`;

getLanguageStandard`()`	Returns the current Modelica Language Standard in use. *Outputs*: `String outVersion`;
getMessagesString`()`	Returns the current error message. *Outputs*: `String messagesString`;
getMessagesStringInternal`()`	Returns Error: message, `TRANSLATION`, Error, code. *Outputs*: `ErrorMessage[:] messagesString`;
getModelicaPath`()`	Get the Modelica Library Path. *Outputs*: `String modelicaPath`;
getNoSimplify`()`	Returns true if noSimplify flag is set. *Outputs*: `Boolean res`;
getNthAlgorithm`(className, index)`	Returns the Nth Algorithm section. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthAlgorithmItem`( className, index)`	Returns the Nth Algorithm Item. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthAnnotationString `(className, index)`	Returns the Nth Annotation section as string. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthEquation`(className, index)`	Returns the Nth Equation section. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthEquationItem`( className)`	Returns the Nth Equation Item. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthImport`(className, index)`	Returns the Nth Import as string. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String out[3] "{\"Path\",\"Id\",\"Kind\"}"`;
getNthInitialAlgorithm`( className, index)`	Returns the Nth Initial Algorithm section. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthInitialAlgorithmItem`( className, index)`	Returns the Nth Initial Algorithm Item. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthInitialEquation`( className, index)`	Returns the Nth Initial Equation section. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getNthInitialEquationItem`( className, index)`	Returns the Nth Initial Equation Item. *Inputs:* `TypeName className; Integer index;` *Outputs:* `String res;`
getOrderConnections`()`	Returns true if orderConnections flag is set. *Outputs*: `Boolean isOrderConnections`;
getPackages`(className)`	Returns the list of packages defined in the class. *Inputs*: `TypeName className = $Code(AllLoadedClasses);` *Outputs*: `TypeName classNames[:]`;
getPlotSilent`()`	Returns true if plotSilent flag is set. *Outputs*: `Boolean isPlotSilent`;
getSettings`()`	Returns the settings. *Outputs*: `String settings`;
getShowAnnotations`()`	*Outputs*: `Boolean show`;
getSourceFile`(className)`	Returns the filename of the class. *Inputs*: `TypeName className;` *Outputs*: `String filename`;
getTempDirectoryPath`()`	Returns the current user temporary directory location. *Outputs*: `String tempDirectoryPath`;
getVectorizationLimit`()`	*Outputs*: `Integer vectorizationLimit`;
getVersion`(cl)`	Returns the version of the Modelica compiler. *Inputs*: `TypeName`

	`c1 = $Code(OpenModelica);` *Outputs*: `String version`;
`help``()`	Display the OpenModelica help text. *Outputs*: `String helpText`;
`iconv``(string, from, to)`	Converts one multibyte characters from one character set to another. *Inputs*: `String string; String from; String to;` *Outputs*: `String res`;
`importFMU``(fileName, workDir)`	Imports the Functional Mockup Unit. *Inputs*: `String filename; String workdir "./"` "The output directory for imported FMU files. <default> will put the files to current working directory."`;` *Outputs*: `Boolean res`;
`instantiateModel``(className)`	Instantiate model, resulting in a `.mof` file of flattened Modelica. *Inputs*: `TypeName className;` *Outputs*: `String res;`
`isModel``(className)`	Returns true if the given class has restriction model. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
`isPackage``(className)`	Returns true if the given class is a package. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
`isPartial``(className)`	Returns true if the given class is partial. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
`list``(className)`	Print class definition. *Inputs*: `TypeName className;` *Outputs*: `String classDef;`
`listVariables``()`	Print user defined variables. *Outputs*: `VariableName res`;
`loadFile``(fileName)`	Load models from file. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
`loadFileInteractive``( fileName)`	Outputs the class names in the parsed file (top level only). Used by OpenModelica MDT. *Inputs*: `String fileName;` *Outputs*: `TypeName names[:];`
`loadFileInteractiveQualified``(fileName)`	Output all the class names in the parsed file fully qualified. *Inputs*: `String fileName;` *Outputs*: `TypeName names[:];`
`loadModel``(className)`	Load the file corresponding to the class, using the Modelica class name to file name mapping to locate the file. *Inputs*: `TypeName className;` *Outputs*: `Boolean res;`
`loadString``(data)`	Parses the data and merges the resulting AST with the loaded AST. *Inputs*: `String data;` *Outputs*: `TypeName names[:];`
`parseFile``(filename)`	Parses the file and returns a list of the classes found in the file. *Inputs*: `String filename;` *Outputs*: `TypeName names[:];`
`parseString``(data)`	Parses the string `data` and returns the list of classes found in `data`. *Inputs*: `String data;` *Outputs*: `TypeName names[:];`
`plot``(variable, options)`	Plots `variable`, which is a single variable name. *Inputs*: `VariableName variable; String title;` `Boolean legend; Boolean gridLines;` `Real xrange[2]` i.e. `{xmin,xmax};` `Real yrange[2]` i.e. `{ymin,ymax};` *Outputs*: `Boolean res;`
`plot``(variables, options)`	Plots `variables`, which is a vector of variable names. *Inputs*: `VariableName variables; String title;` `Boolean legend; Boolean gridLines;`

	`Real xrange[2]` i.e., {xmin,xmax}; `Real yrange[2]` i.e., {ymin,ymax}; *Outputs*: `Boolean res;`
plotAll`(options)`	Plot all variables; It does not accept any variable names as input . *Inputs*: `String title; Boolean legend;  Boolean gridLines; Real xrange[2]` i.e., {xmin,xmax}; `Real yrange[2]` i.e., {ymin,ymax}; *Outputs*: `Boolean res;`
plotParametric`(variables1, variables2, options)`	Plot each pair of corresponding variables from the vectors of variables `variables1`, `variables2` as a parametric plot. *Inputs*: `VariableName variables1[:]; VariableName variables2[size(variables1,1)]; String title; Boolean legend;  Boolean gridLines; Real range[2,2];` *Outputs*: `Boolean res;`
readFile`(fileName)`	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 outputs or similar. *Inputs*: `String fileName;` *Outputs*: `String content;`
readFileNoNumeric`(fileName)`	Returns the contents of the file, with anything resembling a (real) number stripped out, and at the end adding: *Inputs*: `String fileName;` *Outputs*: `String content;`
readFilePostprocessLineDirective`(fileName)`	Searches lines for the `#modelicaLine` directive. *Inputs*: `String fileName;` *Outputs*: `String res;`
readFileShowLineNumbers`(fileName)`	Prefixes each line in the file with <n>: where n is the line number. Note: Scales O(n^2) *Inputs*: `String fileName;` *Outputs*: `String res;`
readSimulationResult`(fileName, variables, size)`	Reads the simulation result for a list of variables and returns a matrix of values (each column as a vector or values for a variable.) Size of result is also given as input. *Inputs*: `String fileName; VariableName variables[:]; Integer size;` *Outputs*: `Real res[size(variables,1),size)];`
readSimulationResultSize`(fileName)`	The number of intervals that are present in the output file. *Inputs*: `String fileName;` *Outputs*: `Integer size;`
readSimulationResultVars`(fileName)`	Returns the variables in the simulation file; you can use `val()` and `plot()` commands using these names. *Inputs*: `String fileName;` *Outputs*: `String[:] vars;`
regex`(string, re, options)`	Sets the error buffer and returns -1 if the regular expression does not compile. *Inputs*: `String string; String re; Integer maxMatches; Boolean extended; Boolean caseInsensitive;` *Outputs*: `Integer numMatches; String matchedSubstrings[maxMatches];`
regexBool`(string, re, options)`	Returns true if the string matches the regular expression. *Inputs*: `String string; String re; Boolean extended; Boolean caseInsensitive;` *Outputs*: `Boolean matches;`
regularFileExists`(fileName)`	Returns the content of the given files. Note that if the function fails, the error message is returned as a string instead of multiple outputs or similar. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`

reopenStandardStream(_stream, fileName)	Executes the script file given as argument. *Inputs*: `String fileName; StandardStream _stream;` *Outputs*: `Boolean res;`
runScript(fileName)	Executes the script file given as argument. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
save(className)	Save class definition. *Inputs*: `TypeName className` *Outputs*: `Boolean res;`
saveAll(fileName)	Save the entire loaded AST to file. *Inputs*: `String fileName;` *Outputs*: `Boolean res;`
saveModel(fileName, className)	Save class definition in a file. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
saveTotalModel(fileName, className)	Save total class definition into file of a class. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
saveTotalSCode(fileName, className)	*Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
setAnnotationVersion(annotationVersion)	Sets the annotation version. *Inputs*: `String annotationVersion;` *Outputs*: `Boolean res;`
setCXXCompiler(compiler)	*Inputs*: `String compiler;` *Outputs*: `Boolean res;`
setClassComment(className, fileName)	Sets the class comment. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
saveTotalModel(fileName, className)	Save total class definition into file of a class. *Inputs*: `String fileName; TypeName className` *Outputs*: `Boolean res;`
setCompileCommand(compileCommand)	Sets the default Compilation command. *Inputs*: `String compileCommand;` *Outputs*: `Boolean res;`
setCompiler(compiler)	Sets the default C Compiler. *Inputs*: `String compiler;` *Outputs*: `Boolean res;`
setCompilerFlags(compilerFlags)	Sets the compiler flags that are used while compiling the simulation executable. *Inputs*: `String compilerFlags;` *Outputs*: `Boolean res;`
setCompilerPath(compilerPath)	Sets the default compiler location. *Inputs*: `String compilerPath;` *Outputs*: `Boolean res;`
setComponentModifierValue(className, modifier, value)	Sets the component modifier value. *Inputs*: `TypeName className; String modifier; String value;` *Outputs*: `Boolean result`;
setDebugFlags(debugFlags)	Sets the debug flags. For details run “omc +help=debug” on the command line interface. *Inputs*: `String debugFlags;` *Outputs*: `Boolean res;`
setEnvironmentVar(var, value)	Sets an environment variable. *Inputs*: `String var; String value;` *Outputs*: `Boolean res;`
setExtendsModifierValue(className, modifier, value)	Sets the extends modifier value. *Inputs*: `TypeName className; String modifier; String value;` *Outputs*: `Boolean result`;
setIndexReductionMethod(method)	Sets the index reduction method e.g `setIndexReductionMethod ("dynamicStateSelection").` *Inputs*: `String method;` *Outputs*: `Boolean res;`
setInitXmlStartValue(fileNam	Sets the parameter start value in the `model_init.xml` file. No

`e, variableName, startValue, outputFile)`	need to recompile the model. *Inputs*: `String fileName; String variableName; String startValue; String outputFile;` *Outputs*: `Boolean success;`
`setInstallationDirectoryPath``(debugFlags)`	Sets the `OPENMODELICAHOME` environment variable. *Inputs*: `String installationDirectoryPath;` *Outputs*: `Boolean res;`
`setLanguageStandard``( inVersion)`	Sets the Modelica Language Standard. *Inputs*: `String inVersion;` *Outputs*: `Boolean res;`
`setLinker``(linker)`	Sets the linker. *Inputs*: `String linker;` *Outputs*: `Boolean res;`
`setLinkerFlags``(linkerFlag)`	Sets the linker flag. *Inputs*: `String linkerFlag;` *Outputs*: `Boolean res;`
`setModelicaPath(`` modelicaPath)`	Sets the modelica path. See *loadModel()* for a description of what the `MODELICAPATH` is used for. *Inputs*: `String modelicaPath;` *Outputs*: `Boolean res;`
`setNoSimplify``(noSimplify)`	Sets the noSimplify flag. *Inputs*: `Boolean noSimplify;` *Outputs*: `Boolean res;`
`setOrderConnections``( debugFlags)`	Sets orderering for connect equation. If set, the compiler will order connect equations alphabetically. *Inputs*: `Boolean orderConnections;` *Outputs*: `Boolean res;`
`setParameterValue``(className, parameter, value)`	Sets the parameter value in the class, i.e., updates the parameter definition equation. *Inputs*: `TypeName className; String parameter; String value;` *Outputs*: `Boolean result;`
`setPlotSilent``(silent)`	Sets the `plotSilent` flag. If the flag is true show the OMPlot window and pass the arguments to it. If the flag is false don't show OMPlot and just output the list of arguments. The false case is used in OMNotebook where the plot window is not shown; instead the arguments are read and an embedded plot window is created. *Inputs*: `Boolean silent;` *Outputs*: `Boolean res;`
`setPostOptModules``(modules)`	Sets the post optimization modules to use in the compiler back end. Use the command `"omc +help=optmodules"` for more information. Example: `setPostOptModules("lateInline", "inlineArrayEqn","removeSimpleEquations")` *Inputs*: `String module1, module2, …;` *Outputs*: `Boolean res;`
`setPreOptModules``(module)`	Sets the pre optimization modules to use in the back end. Use the command line command `"omc +help=optmodules"` for more information. *Inputs*: `String module;` *Outputs*: `Boolean res;`
`setShowAnnotations``()`	Show annotations in the flattened code. *Inputs*: `Boolean show;` *Outputs*: `Boolean res;`
`setSourceFile``(className, fileName)`	Sets the output file name for the specified class. *Inputs*: `TypeName className; String fileName;` *Outputs*: `Boolean res;`
`setTempDirectoryPath``(tempDirectoryPath)`	Sets the current user's temporary directory path. This is not the same as the current user's working directory which is set by the `cd()` command. *Inputs*: `String tempDirectoryPath;` *Outputs*: `Boolean res;`
`setVectorizationLimit``(`	Sets scalarization limit. Arrays of size above this limit will not be

`vectorizationLimit)`	scalarized, i.e., expanded to a set of scalar elements. *Inputs*: `Integer scalarizationLimit;` *Outputs*: `Boolean res;`
`simulate``(className, options)`	Simulate model, optionally setting simulation values. *Inputs*: `TypeName className; Real startTime; Real stopTime; Integer numberOfIntervals; Real outputInterval; String method; Real tolerance; Real fixedStepSize;` *Outputs*: `SimulationResult simRes;`
`stringReplace``(str, source, target)`	Relace `source` with `target` within `str` producing `res`. *Inputs*: `String str; String source; String target;` *Outputs*: `String res;`
`strtok``()`	Splits the strings at the places given by the token, for example*:* `strtok("abcbdef","b") => {"a","c","def"}` *Inputs*: `String string; String token;` *Outputs*: `String[:] strings;`
`system``(fileName)`	Execute system command. *Inputs*: `String fileName;` *Outputs*: `Integer res;`
`typeNameString``(cl)`	Converts the qualified class name to string. *Inputs*: `TypeName cl;` *Outputs*: `String out;`
`typeNameStrings``(cl)`	Converts the qualified class name to strings. *Inputs*: `TypeName cl;` *Outputs*: `String out[:];`
`typeOf``(variableName)`	*Inputs*: `VariableName variableName;` *Outputs*: `String res;`
`uriToFilename``(uri)`	Handles modelica:// and file:// URI's. The result is an absolute path on the local system. The result depends on the current MODELICAPATH. Returns the empty string on failure. *Inputs*: `String uri;` *Outputs*: `String fileName;`
`val``(var, time)`	Returns the value of the variable at a certain time point. *Inputs*: `VariableName var; Real time;` *Outputs*: `Real valAtTime;`
`writeFile(``fileName, data, optional`**`)`**	Write the data to file. Returns true on success. *Inputs*: `String fileName; String data; Boolean append;` *Outputs*: `Boolean res;`

15.3.1 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. The system responds with a header line:

```
OpenModelica 1.9.0
```

We type in a very small model:

```
>> model test "Testing OpenModelica Scripts" Real x, y; equation x = 5.0; y = 6.0;
end test;
```

```
   {test}
```

We give the command to flatten a model:

```
 >> instantiateModel(test)
   "class test
       Real x;
       Real y;
     equation
       x = 5.0;
       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)
   "class test \"Testing OpenModelica Scripts\"
       Real x;
       Real y;
     equation
       x = 5.0;
       y = 6.0;
     end test;
  "
```

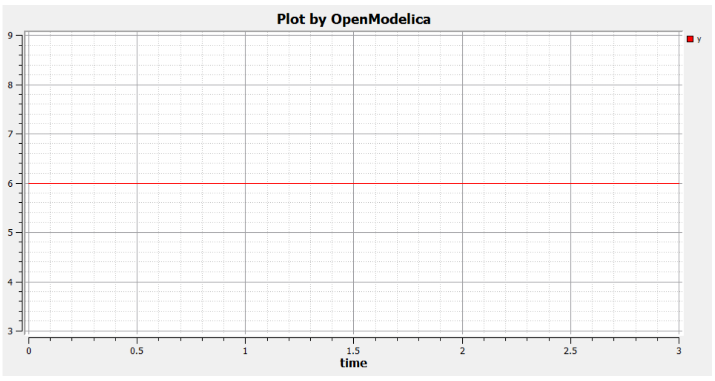
We get the name and other properties of a class:

```
 >> getClassNames()
    {test}

 >> 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).
     "

 >> clear()
   true

 >> getClassNames()
   {}
```

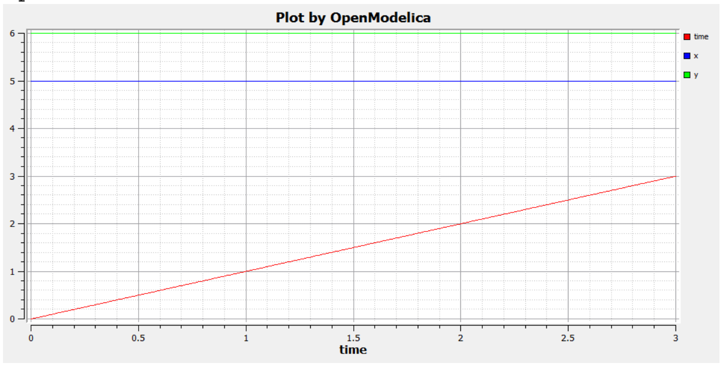
The common combination of a simulation followed by getting a value and doing a plot:

```
 >> simulate(test, stopTime=3.0);
```

```
>> val(x , 2.0)
   5.0

>> plot(y)
```

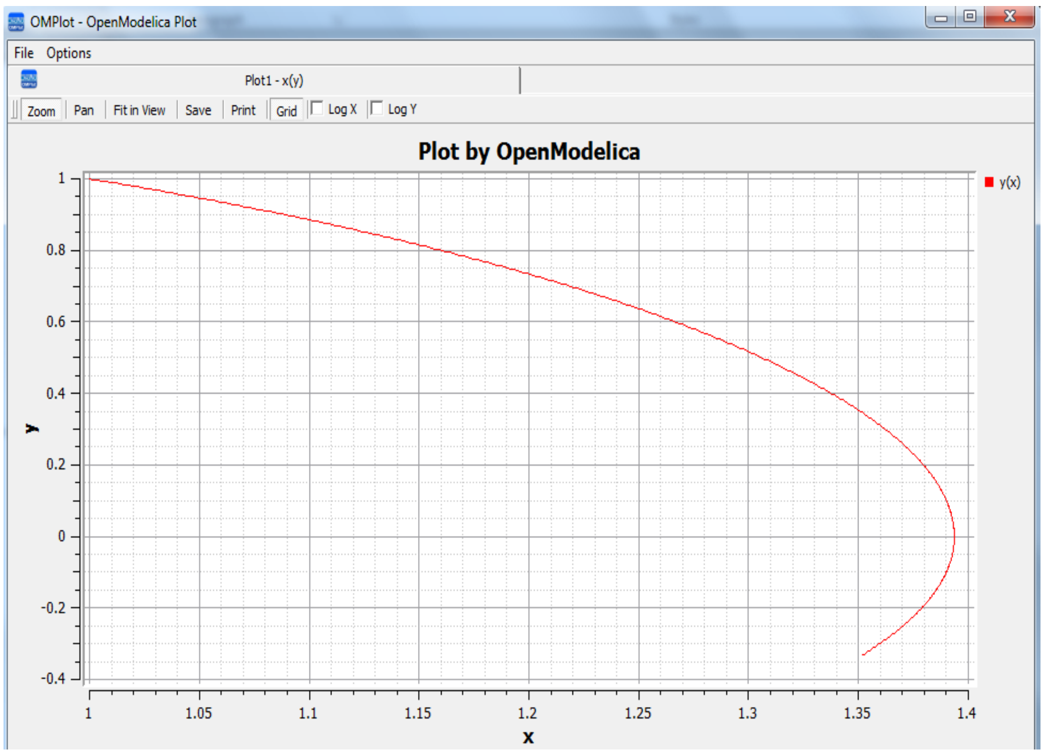


```
  >> plotAll()
```



VanDerPol Model and Parametric Plot

```
>> loadFile("C:/OpenModelica1.9.0/share/doc/omc/testmodels/VanDerPol.mo")
   true

>> 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;
  "

>> simulate(VanDerPol);

>> plotParametric(x,y)
```


OMPlot - OpenModelica Plot
File Options
Plot1 - x(y)
Zoom Pan Fit in View Save Print Grid Log X Log Y
Plot by OpenModelica
y(x)
y
x

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.

```
>> bubblesort(x)
   {12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

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 := read("simplex_in.txt")
   {{-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}}
```

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:

```
>> simplex1(a)
   Tuple: 4
   Real[8]: {9, 9, 4, 5, 0, 0, 33, 0}
   Real: 22
   Integer: 9
   Integer: 3
```

It is possible to compute an expression, e.g. `12:-1:1`, and store the result in a file using the `write` command:

```
>> write(12:-1:1,"test.dat")
```

We can read back the stored result from the file into a variable `y`:

```
>> y := read("test.dat")
   Integer[12]: {12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

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 `system` applied to the Linux command `cat`, which here outputs the contents of the file `bubblesort.mo` to the output stream.

```
>> system("cat 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;
```

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

```
>> cd("..")
"/home/petfr/modelica"
```

15.4 OpenModelica Python Scripting Commands

The OpenModelica Python API Interface—OMPython, attempts to mimic the Modelica scripting commands available in the OpenModelica environment, see the table in Section **Error! Reference source not found.** for available Python scripting commands.

The third party library of OMPython can be created by changing directory to the following:

```
OpenModelicaInstallationDirectory/share/omc/scripts/PythonInterface/
```

and running the command:

```
python setup.py install
```

This will install the OMPython library into the user's Python’s third party libraries directory. Now OMPython can be imported and used within Python.

To test the command outputs, simply import the OMPython library from Python and execute the `run()` method of OMPython. The module allows you to interactively send commands to the OMC server and display their output.
For example:

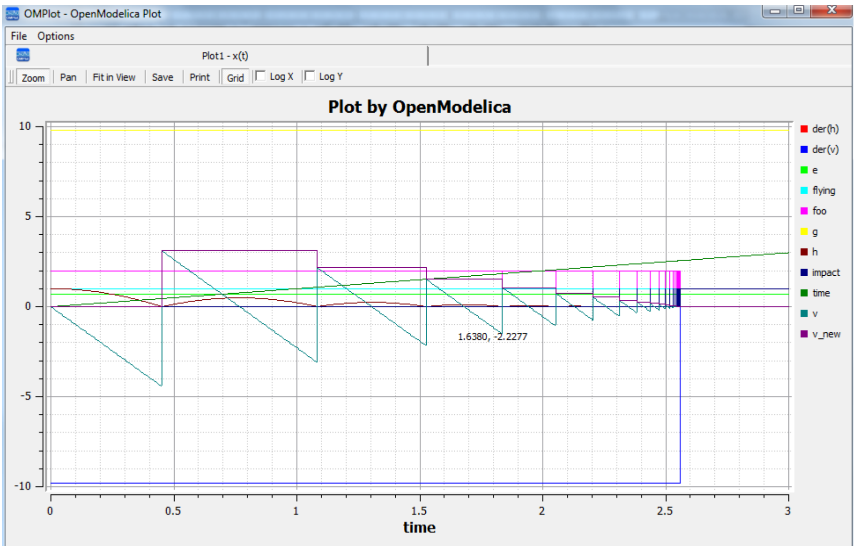
```
C:\>python
  >>> import OMPython
  >>> OMPython.run()
  >> loadModel(Modelica)
```

```
true
```

15.4.1 Examples

The following session in OpenModelica illustrates the use of a few of the above-mentioned functions.

```
>> loadFile("C:/OpenModelica1.9.0/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;
 "
>> getClassNames()
 {'SET1': {'Set1': ['BouncingBall', 'ModelicaServices', 'Complex',
 'Modelica']}}
>> isPartial(BouncingBall)
  False
>> isPackage(BouncingBall)
  False
>> isModel(BouncingBall)
  True
>> checkModel(BouncingBall)
  "Check of BouncingBall completed successfully.
   Class BouncingBall has 6 equation(s) and 6 variable(s).
   1 of these are trivial equation(s).
   "
>> getClassRestriction(BouncingBall)
   "model"
>> getClassInformation(BouncingBall)
'SET1': {'Values': ['"model","","C:\\\\OpenModelica1.9.0\\\\share\\\\doc\\\\omc
\\\\testmodels\\\\BouncingBall.mo"']}, 'SET2': {'Set1': [False, False, False],
'Set2': ['"writable"', 1, 1, 23, 17]}, 'SET3': {'Set1': [False, False, False],
'Set2': ['"writable"', 1, 1, 23, 17]}}
>> existClass(BouncingBall)
True
>> getComponents(BouncingBall)
{'SET1': {}, 'SET2': {'Set4': ['Real', 'v', '"velocity of ball"', '"public"',
'false', 'false', 'false', 'false', '"unspecified"', '"none"',
'"unspecified"'], 'Set5': ['Boolean', 'flying', '"true', 'if ball is flying"',
'"public"', 'false','false', 'false', 'false', '"unspecified"', '"none"',
'"unspecified"'], 'Set6':['Boolean', 'impact', '""', '"public"', 'false',
'false', 'false', 'false', '"unspecified"', '"none"', '"unspecified"'], 'Set7':
['Real', 'v_new', '""', '"public"', 'false', 'false', 'false', 'false',
'"unspecified"', '"none"', '"unspecified"'], 'Set1': ['Real', 'e','"coefficient
of restitution"', '"public"', 'false', 'false', 'false', 'false',
'"parameter"', '"none"', '"unspecified"'], 'Set2':['Real', 'g', '"gravit
acceleration"', '"public"', 'false', 'false', 'false', '
   false', '"parameter"', '"none"', '"unspecified"'], 'Set3': ['Real', 'h',
   '"height of ball"', '"public"', 'false', 'false', 'false', 'false',
  '"unspecified"', '"none"', '"unspecified"'], 'Set8': ['Integer', 'foo', '""',
  '"public"', 'false','false', 'false', 'false', '"unspecified"', '"none"',
  '"unspecified"']}, 'SET3':{}}
>> getConnectionCount(BouncingBall)
   0
>> getNthConnection(BouncingBall)
   1
>> getInheritanceCount(BouncingBall)
   0
>> getComponentModifierValue(BouncingBall,e)
7.5
>> getClassAttributes(BouncingBall)
 {'SET1': {'Elements': {'rec1': {'Properties':
 {'Results': {'comment': '""',
 'restriction': 'MODEL', 'startLine': 1, 'partial': False, 'name':
 '"BouncingBall"','encapsulated': False, 'startColumn': 1, 'readonly':
 '"writable"', 'endColumn': 17, 'file'
 '"C:\\OpenModelica1.9.0\\share\\doc\\omc\\testmodels\\BouncingBall.m
    o"', 'endLine': 23, 'final': False}}}}}}
>> simulate(BouncingBall, stopTime=3.0)
    record SimulationResult
      resultFile =
       "C:/Users/alash325/AppData/Local/Temp/OpenModelica/BouncingBall_res.mat",
       simulationOptions = "startTime = 0.0, stopTime = 3.0,
       numberOfIntervals = 500, tolerance = 0.000001, method = 'dassl',
       fileNamePrefix = 'BouncingBall', options = '',
       outputFormat = 'mat', variableFilter = '.*',
        measureTime = false, cflags = '', simflags = ''", messages = "",
     timeFrontend = 0.02245879828436323б,
     timeBackend = 0.01647555356928777,
     timeSimCode = 0.005109221712579149,
     timeTemplates = 0.01791270548614413З,
     timeCompile = 1.7019944515670637,
     timeSimulation = 0.21892626420791483,
     timeTotal = 1.9834023618418568
     end SimulationResult;
>> plotAll()
   true
```

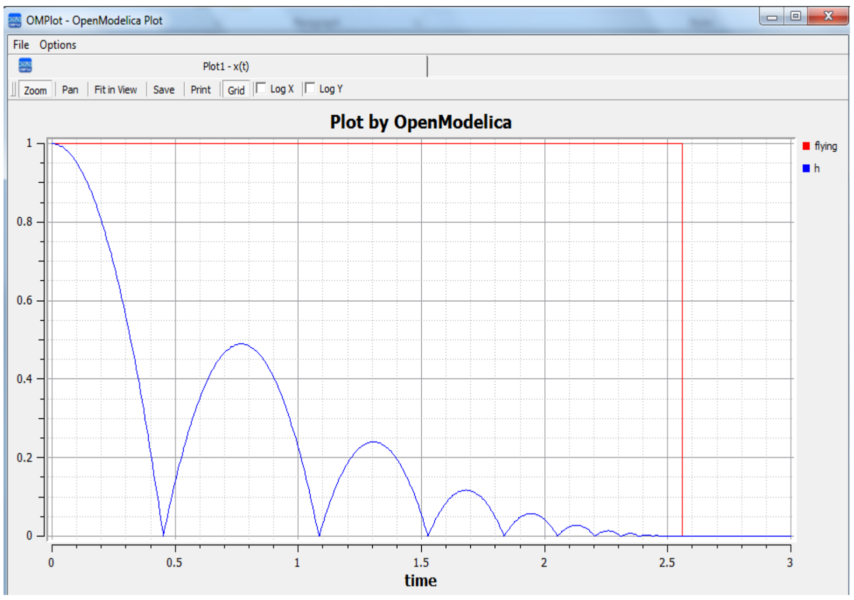


Instead of just giving a `simulate and plot` command, we perform a `runScript` command on a `.mos` (Modelica script) file `sim_BouncingBall.mos` that contains these commands:

```
loadFile("C:/OpenModelica1.9.0/share/doc/omc/testmodels/BouncingBall.mo")
simulate(BouncingBall, stopTime=3.0);
  plot({h,flying});
```

The runScript command:

```
>> runScript("sim_BouncingBall.mos")
    "true
    record SimulationResult
      resultFile =
       "C:/Users/alash325/AppData/Local/Temp/OpenModelica/BouncingBall_res.mat",
       simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals =
       500, tolerance = 0.000001, method = 'dassl', fileNamePrefix =
      'BouncingBall', options = '', outputFormat = 'mat', variableFilter = '.*',
        measureTime = false, cflags = '', simflags = ''",
     messages = "",
     timeFrontend = 0.022458798284363236,
     timeBackend = 0.01647555356928777,
     timeSimCode = 0.005109221712579149,
     timeTemplates = 0.017912705486144133,
     timeCompile = 1.7019944515670637,
     timeSimulation = 0.21892626420791483,
     timeTotal = 1.9834023618418568
    end SimulationResult;
true
"
```



```
>> clear()
 true
>> getClassNames()
     {}
```

Chapter 16

Frequently Asked Questions (FAQ)

Below are some frequently asked questions in three areas, with associated answers.

16.1 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.

16.2 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.

16.3 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.

Appendix A

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 1.9.1 are described.

A.1 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.

A.1.1 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.

A.1.2 OpenModelica Notebook (OMNotebook)

No changes apart from bug fixing.

A.1.3 OpenModelica Shell (OMShell)

No changes.

A.1.4 OpenModelica Eclipse Plug-in (MDT)

No changes apart from bug fixing.

A.1.5 OpenModelica Development Environment (OMDev)

No changes apart from bug fixing.

A.1.6 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 OpenModelica compiler.

A.1.7 Optimization

A builtin integrated Dynamic Optimization module with collocation, using Ipopt, is now available.

A.1.8 FMI Support

Support for FMI 2.0 model exchange import and export has been added. FMI 1.0 support has been further improved.

A.2 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.

A.2.1 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 MSL 3.3 chapter 3.7.2.5 semiLinear.

A.2.2 OpenModelica Notebook (OMNotebook)

The DrModelica interactive document has been updated and the models tested. Almost all models now simulate with OpenModelica.

A.2.3 OpenModelica Shell (OMShell)

No changes.

A.2.4 OpenModelica Eclipse Plug-in (MDT)

Enhanced debugger for algorithmic Modelica code, supporting both standard Modelica algorithmic code called from simulation models, and MetaModelica code.

A.2.5 OpenModelica Development Environment (OMDev)

Migration of version handling and configuration management from CodeBeamer to Trac.

A.2.6 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 is 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.

A.2.7 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.

A.2.8 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.

A.3 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.

A.3.1 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.

A.3.2 OpenModelica Notebook (OMNotebook)

Faster and more stable plottning.

A.3.3 OpenModelica Shell (OMShell)

No changes.

A.3.4 OpenModelica Eclipse Plug-in (MDT)

Small fixes and improvements.

A.3.5 OpenModelica Development Environment (OMDev)

No changes.

A.3.6 Graphic Editor OMEdit

Bug fixes.

A.3.7 New OMOptim Optimization Subsystem

Bug fixes.

A.3.8 FMI Support

Bug fixes.

A.4 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.

A.4.1 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.

A.4.2 OpenModelica Notebook (OMNotebook)

No changes.

A.4.3 OpenModelica Shell (OMShell)

Small performance improvements.

A.4.4 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.

A.4.5 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.

A.4.6 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.

A.4.7 New OMOptim Optimization Subsystem

Bug fixes, especially in the Linux version.

A.4.8 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.

A.5 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 compiler itself. This release also includes two interesting new tools – the OMOpttim optimization subsystem, and a new performance profiler for equation-based Modelica models.

A.5.1 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.

A.5.2 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.

A.5.3 OpenModelica Shell (OMShell)

Same as previously, except the improved 2D plotting through OMPlot.

A.5.4 OpenModelica Eclipse Plug-in (MDT)

Same as previously.

A.5.5 OpenModelica Development Environment (OMDev)

No changes.

A.5.6 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.

A.5.7 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.

A.5.8 New Performance Profiler

A new, low overhead, performance profiler for Modelica models has been developed.

A.6 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.

A.6.1 OpenModelica Compiler (OMC)

This release includes bug fix and performance improvemetns 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.

A.6.2 OpenModelica Notebook (OMNotebook)

A new DrControl electronic notebook for teaching control and modeling with Modelica.

A.6.3 OpenModelica Shell (OMShell)

Same as previously.

A.6.4 OpenModelica Eclipse Plug-in (MDT)

Same as previously.

A.6.5 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.

A.6.6 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.

A.7 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.

A.7.1 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 simForge 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.

A.7.2 OpenModelica Notebook (OMNotebook)

Improvements in platform availability.

- Support for 64-bit Linux.
- Support for Windows 7.
- Better support for MacOS, including 64-bit OSX.

A.7.3 OpenModelica Shell (OMShell)

Same as previously.

A.7.4 OpenModelica Eclipse Plug-in (MDT)

Minor bug fixes.

A.7.5 OpenModelica Development Environment (OMDev)

Minor bug fixes.

A.8 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.

A.8.1 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.

A.8.2 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).

A.8.3 OpenModelica Shell (OMShell)

Same as previously.

A.8.4 OpenModelica Eclipse Plug-in (MDT)

Minor bug fixes.

A.8.5 OpenModelica Development Environment (OMDev)

Same as previously.

A.1 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.

A.8.6 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 consumtion during compilation and interactive evaluation.
- A number of bug-fixes.

A.8.7 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.

A.8.8 OpenModelica Shell (OMShell)

Same as previously.

A.8.9 OpenModelica Eclipse Plug-in (MDT)

This release includes minor bugfixes of MDT and the associated MetaModelica debugger:

A.8.10 OpenModelica Development Environment (OMDev)

Extended test suite with a better structure. Version handling, bug tracking, issue tracking, etc. now available under the integrated Codebeamer

A.9 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.

A.9.1 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.

A.9.2 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.

A.9.3 OpenModelica Shell (OMShell)

Now available also on the Macintosh platform.

A.9.4 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.

A.9.5 OpenModelica Development Environment (OMDev)

Increased compilation speed for MetaModelica. Better if-expression support in MetaModelica.

A.10 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.

A.10.1 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 evalution 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.

A.10.2 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.

A.10.3 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.

A.10.4 OpenModelica Development Environment (OMDev)

Mostly the same as previously.

A.11 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.

A.11.1 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.

A.11.2 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).

A.11.3 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.

A.12 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..

A.12.1 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.

A.12.2 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.

A.12.3 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.

A.12.4 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.

A.12.5 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.

A.13 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 is possible to use OpenModelica for both academic and commercial purposes without restrictions.

A.13.1 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.

A.13.2 OpenModelica Shell (OMShell)

An improved window-based interactive command shell, now including command completion and better editing and font size support.

A.13.3 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.

A.13.4 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.

A.13.5 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.

Appendix B

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.

B.1 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, 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.

B.2 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.

B.3 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.

B.4 OpenModelica Contributors 2011

Peter Fritzson, 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.

B.5 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, IEI, 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.

B.6 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

B.7 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

B.8 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

B.9 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.

B.10 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.

B.11 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.

B.12 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.

B.13 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

B.14 OpenModelica Contributors 2001

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

B.15 OpenModelica Contributors 2000

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

B.16 OpenModelica Contributors 1999

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden

–

Peter Rönnquist, PELAB, Linköping University, Linköping, Sweden.

B.17 OpenModelica Contributors 1998

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

David Kågedal, PELAB, Linköping University, Linköping, Sweden.

Vadim Engelson, PELAB, Linköping University, Linköping, Sweden.

Appendix C

Integration Methods

SimulationRuntime\IntegrationAlgorithms\IntegrationAlgorithms.pdf

Index