OpenModelica User's Guide

Release v1.28.0-dev-838-g4c2a253e5a

Open Source Modelica Consortium

2026

CONTENTS

1 Introduction **3**
- 1.1 System Overview . . . 4
- 1.2 Interactive Session with Examples . . . 5
- 1.3 Summary of Commands for the Interactive Session Handler . . . 24
- 1.4 Running the compiler from command line . . . 25

2 Package Management **27**
- 2.1 Overview of Basic Modelica Package Management Concepts . . . 27
- 2.2 The Package Manager . . . 28
- 2.3 How the package index works . . . 30

3 OMEdit - OpenModelica Connection Editor **31**
- 3.1 Starting OMEdit . . . 31
- 3.2 MainWindow & Browsers . . . 31
- 3.3 Perspectives . . . 37
- 3.4 File Menu . . . 39
- 3.5 Edit Menu . . . 41
- 3.6 View Menu . . . 42
- 3.7 SSP Menu . . . 42
- 3.8 Simulation Menu . . . 42
- 3.9 Data Reconciliation . . . 43
- 3.10 Sensitivity Optimization Menu . . . 43
- 3.11 Debug Menu . . . 43
- 3.12 Tools Menu . . . 43
- 3.13 Help Menu . . . 43
- 3.14 Modeling a Model . . . 43
- 3.15 Simulating a Model . . . 45
- 3.16 2D Plotting . . . 50
- 3.17 Re-simulating a Model . . . 51
- 3.18 3D Visualization . . . 52
- 3.19 Animation of Realtime FMUs . . . 55
- 3.20 Interactive Simulation . . . 55
- 3.21 How to Create User Defined Shapes - Icons . . . 56
- 3.22 Global head section in documentation . . . 57
- 3.23 Options . . . 58
- 3.24 __OpenModelica_commandLineOptions Annotation . . . 66
- 3.25 __OpenModelica_simulationFlags Annotation . . . 67
- 3.26 Global and Local Flags . . . 67
- 3.27 Debugger . . . 68
- 3.28 Editing Modelica Standard Library . . . 68
- 3.29 Install Library . . . 68
- 3.30 Convert Libraries using Conversion Scripts . . . 68
- 3.31 State Machines . . . 68
- 3.32 Using OMEdit as Text Editor . . . 73

3.33 Temporary Directory, Log Files and Working Directory . . . 74
3.34 High DPI Settings . . . 74

4 2D Plotting **81**
4.1 Example . . . 81
4.2 Plot Command Interface . . . 82

5 OpenModelica Compiler **85**
5.1 Frontend Modules . . . 85
5.2 Backend Modules . . . 85
5.3 Code generation . . . 87
5.4 Simulation Runtimes . . . 87

6 Solving Modelica Models **89**
6.1 Integration Methods . . . 89
6.2 DAE Mode Simulation . . . 91
6.3 Initialization . . . 91
6.4 Tearing . . . 97
6.5 Algebraic Solvers . . . 98

7 Automatic Parallelization of Simulations **101**
7.1 Run-time flags . . . 101
7.2 JSON format . . . 101
7.3 External tools . . . 102
7.4 The optimization workflow . . . 103

8 Debugging **105**
8.1 The Equation-based Debugger . . . 105
8.2 The Algorithmic Debugger . . . 107

9 Flattening models to BaseModelica **113**
9.1 BaseModelica . . . 113
9.2 Converting Modelica models in BaseModelica with OpenModelica . . . 113
9.3 Array-preserving BaseModelica output . . . 113

10 Porting Modelica libraries to OpenModelica **115**
10.1 Mapping of the library on the file system . . . 115
10.2 Modifiers for arrays . . . 115
10.3 Access to conditional components . . . 116
10.4 Access to classes defined in partial packages . . . 117
10.5 Equality operator in algorithms . . . 118
10.6 Public non-input non-output variables in functions . . . 118
10.7 Subscripting of expressions . . . 119
10.8 Incomplete specification of initial conditions . . . 119
10.9 Modelica_LinearSystems2 Library . . . 120

11 Backwards Compatibility of OpenModelica Released Versions **121**

12 Generating Graph Representations for Models **123**

13 Functional Mock-up Interface - FMI **125**
13.1 FMI Standards . . . 125
13.2 FMI Export . . . 126
13.3 FMI Import - SSP . . . 133

14 OMSimulator **135**
14.1 Introduction . . . 135
14.2 OMSimulator . . . 135
14.3 OMSimulatorLib . . . 138
14.4 C-API . . . 138

14.5 OMSimulatorLua . . . 154
14.6 OMSimulatorPython . . . 171
14.7 OpenModelicaScripting . . . 189
14.8 Graphical Modelling . . . 201
14.9 SSP Support . . . 213

15 System Identification **215**
15.1 Examples . . . 215
15.2 Python and C API . . . 217

16 OpenModelica Encryption **225**
16.1 Encrypting the Library . . . 225
16.2 Loading an Encrypted Library . . . 225
16.3 Notes . . . 225

17 OMNotebook with DrModelica and DrControl **227**
17.1 Interactive Notebooks with Literate Programming . . . 227
17.2 DrModelica Tutoring System - an Application of OMNotebook . . . 228
17.3 DrControl Tutorial for Teaching Control Theory . . . 234
17.4 OpenModelica Notebook Commands . . . 241
17.5 References . . . 249

18 Optimization with OpenModelica **251**
18.1 Built-in Dynamic Optimization using Annotations . . . 251
18.2 Built-in Dynamic Optimization using Optimica language extensions . . . 261
18.3 Dynamic Optimization with OpenModelica and CasADi . . . 264
18.4 Parameter Sweep Optimization using OMOptim . . . 269

19 Parameter Sensitivities with OpenModelica **277**
19.1 Single Parameter sensitivities with IDA/Sundials . . . 277
19.2 Single and Multi-parameter sensitivities with OMSens . . . 278

20 PDEModelica1 **293**
20.1 PDEModelica1 language elements . . . 293
20.2 Limitations . . . 294
20.3 Viewing results . . . 294

21 MDT - The OpenModelica Development Tooling Eclipse Plugin **295**
21.1 Introduction . . . 295
21.2 Installation . . . 295
21.3 Getting Started . . . 296

22 MDT Debugger for Algorithmic Modelica **309**
22.1 The Eclipse-based Debugger for Algorithmic Modelica . . . 309

23 Modelica Performance Analyzer **315**
23.1 Profiling information for ProfilingTest . . . 316
23.2 Genenerated JSON for the Example . . . 319
23.3 Using the Profiler from OMEdit . . . 320

24 Simulation in Web Browser **323**

25 Interoperability - C and Python **325**
25.1 Calling External C functions . . . 325
25.2 Calling external Python Code from a Modelica model . . . 327
25.3 Calling OpenModelica from Python Code . . . 330

26 OpenModelica Python Interface **333**
26.1 OMPython - OpenModelica Python Interface . . . 333
26.2 Enhanced OMPython Features . . . 336

27 OMMatlab - OpenModelica Matlab Interface **341**
27.1 Features of OMMatlab . . . 341
27.2 Test Commands . . . 341
27.3 WorkDirectory . . . 343
27.4 BuildModel . . . 343
27.5 Standard get methods . . . 343
27.6 Usage of getMethods . . . 343
27.7 Standard set methods . . . 345
27.8 Usage of setMethods . . . 346
27.9 Advanced Simulation . . . 346
27.10 Linearization . . . 346
27.11 Usage of Linearization methods . . . 347

28 OMJulia - OpenModelica Julia Scripting **349**

29 Jupyter-OpenModelica **351**

30 Scripting API **353**
30.1 OpenModelica Scripting Commands . . . 353
30.2 Simulation Parameter Sweep . . . 440
30.3 Examples . . . 441

31 OpenModelica Compiler Flags **447**
31.1 Options . . . 447
31.2 Debug flags . . . 467
31.3 Flags for Optimization Modules . . . 476

32 Simulation Runtime Flags **477**
32.1 C Runtime Simulation Flags . . . 477

33 Technical Details **495**
33.1 The MATv4 Result File Format . . . 495
33.2 The Arrow Result File Format . . . 496

34 Data Reconciliation **497**
34.1 Objective of Data Reconciliation . . . 497
34.2 Defining the Data Reconciliation Problem in OpenModelica . . . 497
34.3 Data Reconciliation Support in OMEdit . . . 501
34.4 Computing the Boundary Conditions from the Reconciled Values . . . 505
34.5 Contacts . . . 508
34.6 References . . . 508

35 Frequently Asked Questions (FAQ) **509**
35.1 OpenModelica General . . . 509
35.2 OMNotebook . . . 509

36 Major OpenModelica Releases **511**
36.1 Release Notes for nightly . . . 511
36.2 Release Notes for OpenModelica 1.27.1 . . . 511
36.3 Patch release for 1.27.0. . . . 511
36.4 Release Notes for OpenModelica 1.27.0 . . . 511
36.5 2026 summer release of OpenModelica version 1.27.0. . . . 511
36.6 Release Notes for v1.26.9 . . . 513
36.7 Release Notes for v1.28.0-dev . . . 514
36.8 Release Notes for OpenModelica v1.26.8 . . . 514
36.9 Release Notes for OpenModelica v1.26.7 . . . 514
36.10 Release Notes for OpenModelica 1.26.6 . . . 514
36.11 Release Notes for OpenModelica 1.26.5 . . . 514
36.12 Release Notes for OpenModelica 1.26.4 . . . 514
36.13 Patch release for 1.26.3. . . . 514

36.14 Release Notes for OpenModelica 1.26.3 . . . 515
36.15 Patch release for 1.26.2. . . . 515
36.16 Release Notes for OpenModelica 1.26.2 . . . 515
36.17 Patch release for 1.26.1. . . . 515
36.18 Release Notes for OpenModelica 1.26.1 . . . 515
36.19 Patch release for 1.26.0. . . . 515
36.20 Release Notes for OpenModelica 1.26.1 Beta 2 . . . 515
36.21 Patch release for 1.26.0. . . . 515
36.22 Release Notes for OpenModelica 1.26.0 . . . 516
36.23 2025 Winter release of OpenModelica version 1.26.0. . . . 516
36.24 Release Notes for OpenModelica 1.26.0-dev.beta.1 . . . 517
36.25 Release Notes for OpenModelica 1.25.7 . . . 519
36.26 Release Notes for OpenModelica 1.25.6 . . . 519
36.27 Release Notes for OpenModelica 1.25.5 . . . 519
36.28 Release Notes for OpenModelica 1.25.4 . . . 519
36.29 Release Notes for OpenModelica 1.25.3 . . . 519
36.30 Release Notes for OpenModelica 1.25.2 . . . 519
36.31 Release Notes for OpenModelica 1.25.1 . . . 520
36.32 Release Notes for OpenModelica 1.25.0 . . . 520
36.33 Release Notes for OpenModelica 1.24.5 . . . 521
36.34 Release Notes for OpenModelica 1.24.4 . . . 521
36.35 Release Notes for OpenModelica 1.24.3 . . . 521
36.36 Release Notes for OpenModelica 1.24.2 . . . 522
36.37 Release Notes for OpenModelica 1.24.0 . . . 522
36.38 Release Notes for OpenModelica 1.23.1 . . . 523
36.39 Release Notes for OpenModelica 1.23.0 . . . 523
36.40 Release Notes for OpenModelica 1.22.3 . . . 524
36.41 Release Notes for OpenModelica 1.22.2 . . . 524
36.42 Release Notes for OpenModelica 1.22.1 . . . 525
36.43 Release Notes for OpenModelica 1.22.0 . . . 525
36.44 Release Notes for OpenModelica 1.21.0 . . . 526
36.45 Release Notes for OpenModelica 1.20.0 . . . 527
36.46 Release Notes for OpenModelica 1.19.2 . . . 529
36.47 Release Notes for OpenModelica 1.19.0 . . . 529
36.48 Release Notes for OpenModelica 1.18.0 . . . 530
36.49 Release Notes for OpenModelica 1.17.0 . . . 532
36.50 Release Notes for OpenModelica 1.16.5 . . . 534
36.51 Release Notes for OpenModelica 1.16.4 . . . 534
36.52 Release Notes for OpenModelica 1.16.2 . . . 534
36.53 Release Notes for OpenModelica 1.16.1 . . . 534
36.54 Release Notes for OpenModelica 1.16.0 . . . 535
36.55 Release Notes for OpenModelica 1.14.2 . . . 536
36.56 Release Notes for OpenModelica 1.14.0 . . . 536

37 Contributors to OpenModelica **559**
37.1 OpenModelica Contributors 2015 . . . 559
37.2 OpenModelica Contributors 2014 . . . 561
37.3 OpenModelica Contributors 2013 . . . 562
37.4 OpenModelica Contributors 2012 . . . 564
37.5 OpenModelica Contributors 2011 . . . 566
37.6 OpenModelica Contributors 2010 . . . 568
37.7 OpenModelica Contributors 2009 . . . 569
37.8 OpenModelica Contributors 2008 . . . 571
37.9 OpenModelica Contributors 2007 . . . 571
37.10 OpenModelica Contributors 2006 . . . 572
37.11 OpenModelica Contributors 2005 . . . 573
37.12 OpenModelica Contributors 2004 . . . 573
37.13 OpenModelica Contributors 2003 . . . 573

37.14 OpenModelica Contributors 2002 . . . 574
37.15 OpenModelica Contributors 2001 . . . 574
37.16 OpenModelica Contributors 2000 . . . 574
37.17 OpenModelica Contributors 1999 . . . 574
37.18 OpenModelica Contributors 1998 . . . 575

Bibliography **577**

Index **579**

Generated on 2026-09-22 at 08:25

Copyright © 1998-2026 Open Source Modelica Consortium (OSMC)
c/o Linköpings universitet, Department of Computer and Information Science
SE-58183 Linköping, Sweden

This work is licensed under a Creative Commons Attribution 4.0 International License.

This document is part of OpenModelica: https://www.openmodelica.org Contact: OpenModelica@ida.liu.se

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

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

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

CHAPTER

ONE

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 long-term goal is to have a complete reference implementation of the Modelica language, including simulation of equation based models and additional facilities in the programming environment, as well as convenient facilities for research and experimentation in language design or other research activities. However, our goal is not to reach the level of performance and quality provided by current commercial Modelica environments that can handle large models requiring advanced analysis and optimization by the Modelica compiler.

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

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

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

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

1.1 System Overview

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

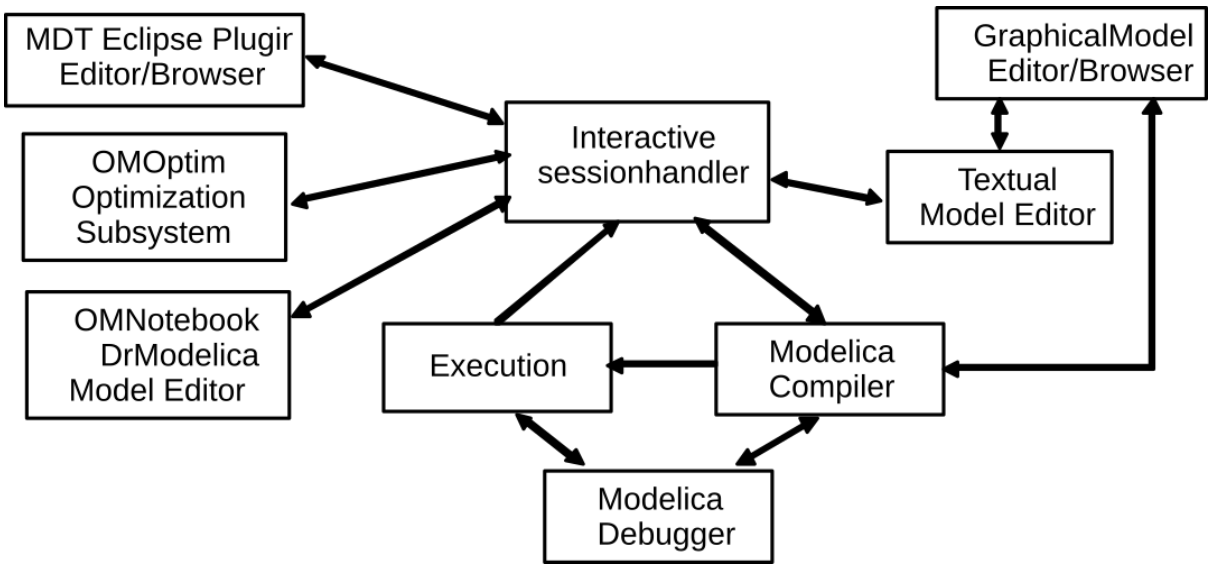


Figure 1.1: The architecture of the OpenModelica environment. Arrows denote data and control flow. The interactive session handler receives commands and shows results from evaluating commands and expressions that are translated and executed. Several subsystems provide different forms of browsing and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

The following subsystems are currently integrated in the OpenModelica environment:

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

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

1.2 Interactive Session with Examples

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

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

The following commands were run using OpenModelica version:

```
>>> getVersion()
"v1.28.0-dev.838+g4c2a253e5a.cmake"
```

1.2.1 Starting the Interactive Session

Under Windows, go to the Start Menu and run OpenModelica->OpenModelica Shell which responds with an interaction window.

Under Linux, run `OMShell-terminal` to start the interactive session at the prompt.

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

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

1.2.2 Using the Interactive Mode

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

Example Session 1

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

(continues on next page)

(continued from previous page)

```
"[<interactive>:1:9-1:23:writable] Error: Type mismatch in binding t = 1.5, expected␣
↪subtype of Integer, got type Real."
```

Example Session 2

If you do not see the error-message when running the example, use the command `getErrorString()`.

```
model C
  Integer a;
  Real b;
equation
  der(a) = b; // der(a) is illegal since a is not a Real number
  der(b) = 12.0;
end C;
```

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

Error

[<interactive>:5:3-5:13:writable] Error: Argument 'a' of der is not differentiable.

1.2.3 Trying the Bubblesort Function

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

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

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

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

Another call:

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

1.2.4 Trying the system and cd Commands

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
"/var/lib/jenkins/ws/OpenModelica/build/share/doc/omc/testmodels"
>>> cd(dir)
"«DOCHOME»"
```

1.2.5 Modelica Library and DCMotor Model

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

```
>>> loadModel(Modelica, {"3.2.3"})
true
```

We also load a file containing the dcmotor model:

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

Note

Notification: dcmotor requested package Modelica of version 3.2.2. Modelica 3.2.3 is used instead which states that it is fully compatible without conversion script needed.

It is simulated:

```
>>> simulate(dcmotor, startTime=0.0, stopTime=10.0)
record SimulationResult
    resultFile = "«DOCHOME»/dcmotor_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'dcmotor', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.007431358000000001,
    timeBackend = 0.011042376000000001,
    timeSimCode = 0.003166753,
    timeTemplates = 0.0027330270000000003,
    timeCompile = 0.18618481399999998,
    timeSimulation = 0.006637016000000001,
    timeTotal = 0.2173000519999999
end SimulationResult;
```

We list the source code of the model:

```
>>> list(dcmotor)
model dcmotor
  import Modelica.Electrical.Analog.Basic;
  Basic.Resistor resistor1(R = 10);
  Basic.Inductor inductor1(L = 0.2, i.fixed = true);
  Basic.Ground ground1;
  Modelica.Mechanics.Rotational.Components.Inertia load(J = 1, phi.fixed = true, w.
↪fixed = true);
  Basic.EMF emf1(k = 1.0);
```

(continues on next page)

(continued from previous page)

```
  Modelica.Blocks.Sources.Step step1;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1;
equation
  connect(step1.y, signalVoltage1.v);
  connect(signalVoltage1.p, resistor1.p);
  connect(resistor1.n, inductor1.p);
  connect(inductor1.n, emf1.p);
  connect(emf1.flange, load.flange_a);
  connect(signalVoltage1.n, ground1.p);
  connect(ground1.p, emf1.n);
  annotation(
    uses(Modelica(version = "3.2.2")));
end dcmotor;
```

We test code instantiation of the model to flat code:

```
>>> instantiateModel(dcmotor)
class dcmotor
  parameter Real resistor1.R(quantity = "Resistance", unit = "Ohm", start = 1.0) = 10.
↪0 "Resistance at temperature T_ref";
  parameter Real resistor1.T_ref(quantity = "ThermodynamicTemperature", unit = "K",␣
↪displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) = 300.15
↪"Reference temperature";
  parameter Real resistor1.alpha(quantity = "LinearTemperatureCoefficient", unit = "1/
↪K") = 0.0 "Temperature coefficient of resistance (R_actual = R*(1 + alpha*(T_
↪heatPort - T_ref))";
  Real resistor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop of the␣
↪two pins (= p.v - n.v)";
  Real resistor1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from␣
↪pin p to pin n";
  Real resistor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin
↪";
  Real resistor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  Real resistor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin
↪";
  Real resistor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  final parameter Boolean resistor1.useHeatPort = false "=true, if heatPort is enabled
↪";
  parameter Real resistor1.T(quantity = "ThermodynamicTemperature", unit = "K",␣
↪displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) = resistor1.T_ref
↪"Fixed device temperature if useHeatPort = false";
  Real resistor1.LossPower(quantity = "Power", unit = "W") "Loss power leaving␣
↪component via heatPort";
  Real resistor1.T_heatPort(quantity = "ThermodynamicTemperature", unit = "K",␣
↪displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) "Temperature of␣
↪heatPort";
  Real resistor1.R_actual(quantity = "Resistance", unit = "Ohm") "Actual resistance =␣
↪R*(1 + alpha*(T_heatPort - T_ref))";
  Real inductor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop of the␣
↪two pins (= p.v - n.v)";
  Real inductor1.i(quantity = "ElectricCurrent", unit = "A", start = 0.0, fixed =␣
↪true) "Current flowing from pin p to pin n";
  Real inductor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin
↪";
```

(continues on next page)

(continued from previous page)

```
  Real inductor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  Real inductor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin
↪";
  Real inductor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  parameter Real inductor1.L(quantity = "Inductance", unit = "H", start = 1.0) = 0.2
↪"Inductance";
  Real ground1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real ground1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  Real load.flange_a.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
  Real load.flange_a.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange
↪";
  Real load.flange_b.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
  Real load.flange_b.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange
↪";
  parameter Real load.J(quantity = "MomentOfInertia", unit = "kg.m2", min = 0.0,␣
↪start = 1.0) = 1.0 "Moment of inertia";
  final parameter enumeration(never, avoid, default, prefer, always) load.stateSelect␣
↪= StateSelect.default "Priority to use phi and w as states";
  Real load.phi(quantity = "Angle", unit = "rad", displayUnit = "deg", fixed = true,␣
↪stateSelect = StateSelect.default) "Absolute rotation angle of component";
  Real load.w(quantity = "AngularVelocity", unit = "rad/s", fixed = true, stateSelect␣
↪= StateSelect.default) "Absolute angular velocity of component (= der(phi))";
  Real load.a(quantity = "AngularAcceleration", unit = "rad/s2") "Absolute angular␣
↪acceleration of component (= der(w))";
  final parameter Boolean emf1.useSupport = false "= true, if support flange enabled,␣
↪otherwise implicitly grounded";
  parameter Real emf1.k(quantity = "ElectricalTorqueConstant", unit = "N.m/A", start␣
↪= 1.0) = 1.0 "Transformation coefficient";
  Real emf1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop between the␣
↪two pins";
  Real emf1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from␣
↪positive to negative pin";
  Real emf1.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Angle of␣
↪shaft flange with respect to support (= flange.phi - support.phi)";
  Real emf1.w(quantity = "AngularVelocity", unit = "rad/s") "Angular velocity of␣
↪flange relative to support";
  Real emf1.tau(quantity = "Torque", unit = "N.m") "Torque of flange";
  Real emf1.tauElectrical(quantity = "Torque", unit = "N.m") "Electrical torque";
  Real emf1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the␣
↪pin";
  Real emf1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the␣
↪pin";
  Real emf1.flange.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
  Real emf1.flange.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange";
  protected final parameter Real emf1.fixed.phi0(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") = 0.0 "Fixed offset angle of housing";
  protected Real emf1.fixed.flange.phi(quantity = "Angle", unit = "rad", displayUnit␣
↪= "deg") "Absolute rotation angle of flange";
```

(continues on next page)

(continued from previous page)

```
  protected Real emf1.fixed.flange.tau(quantity = "Torque", unit = "N.m") "Cut torque␣
↪in the flange";
  protected Real emf1.internalSupport.tau(quantity = "Torque", unit = "N.m") = -emf1.
↪tau "External support torque (must be computed via torque balance in model where␣
↪InternalSupport is used; = flange.tau)";
  protected Real emf1.internalSupport.phi(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") "External support angle (= flange.phi)";
  protected Real emf1.internalSupport.flange.phi(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") "Absolute rotation angle of flange";
  protected Real emf1.internalSupport.flange.tau(quantity = "Torque", unit = "N.m")
↪"Cut torque in the flange";
  parameter Real step1.height = 1.0 "Height of step";
  Real step1.y "Connector of Real output signal";
  parameter Real step1.offset = 0.0 "Offset of output signal y";
  parameter Real step1.startTime(quantity = "Time", unit = "s") = 0.0 "Output y =␣
↪offset for time < startTime";
  Real signalVoltage1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at␣
↪the pin";
  Real signalVoltage1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
  Real signalVoltage1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at␣
↪the pin";
  Real signalVoltage1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
  Real signalVoltage1.v(unit = "V") "Voltage between pin p and n (= p.v - n.v) as␣
↪input signal";
  Real signalVoltage1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪from pin p to pin n";
equation
  emf1.internalSupport.flange.phi = emf1.fixed.flange.phi;
  step1.y = signalVoltage1.v;
  signalVoltage1.p.v = resistor1.p.v;
  resistor1.n.v = inductor1.p.v;
  inductor1.n.v = emf1.p.v;
  emf1.flange.phi = load.flange_a.phi;
  ground1.p.v = emf1.n.v;
  ground1.p.v = signalVoltage1.n.v;
  inductor1.p.i + resistor1.n.i = 0.0;
  emf1.p.i + inductor1.n.i = 0.0;
  load.flange_b.tau = 0.0;
  emf1.flange.tau + load.flange_a.tau = 0.0;
  emf1.internalSupport.flange.tau + emf1.fixed.flange.tau = 0.0;
  signalVoltage1.p.i + resistor1.p.i = 0.0;
  signalVoltage1.n.i + emf1.n.i + ground1.p.i = 0.0;
  assert(1.0 + resistor1.alpha * (resistor1.T_heatPort - resistor1.T_ref) >= 1e-15,
↪"Temperature outside scope of model!");
  resistor1.R_actual = resistor1.R * (1.0 + resistor1.alpha * (resistor1.T_heatPort -␣
↪resistor1.T_ref));
  resistor1.v = resistor1.R_actual * resistor1.i;
  resistor1.LossPower = resistor1.v * resistor1.i;
  resistor1.T_heatPort = resistor1.T;
  resistor1.v = resistor1.p.v - resistor1.n.v;
  0.0 = resistor1.p.i + resistor1.n.i;
  resistor1.i = resistor1.p.i;
  inductor1.L * der(inductor1.i) = inductor1.v;
  inductor1.v = inductor1.p.v - inductor1.n.v;
```

(continues on next page)

(continued from previous page)

```
  0.0 = inductor1.p.i + inductor1.n.i;
  inductor1.i = inductor1.p.i;
  ground1.p.v = 0.0;
  load.phi = load.flange_a.phi;
  load.phi = load.flange_b.phi;
  load.w = der(load.phi);
  load.a = der(load.w);
  load.J * load.a = load.flange_a.tau + load.flange_b.tau;
  emf1.fixed.flange.phi = 0.0;
  emf1.internalSupport.flange.tau = emf1.internalSupport.tau;
  emf1.internalSupport.flange.phi = emf1.internalSupport.phi;
  emf1.v = emf1.p.v - emf1.n.v;
  0.0 = emf1.p.i + emf1.n.i;
  emf1.i = emf1.p.i;
  emf1.phi = emf1.flange.phi - emf1.internalSupport.phi;
  emf1.w = der(emf1.phi);
  emf1.k * emf1.w = emf1.v;
  emf1.tau = -emf1.k * emf1.i;
  emf1.tauElectrical = -emf1.tau;
  emf1.tau = emf1.flange.tau;
  step1.y = step1.offset + (if time < step1.startTime then 0.0 else step1.height);
  signalVoltage1.v = signalVoltage1.p.v - signalVoltage1.n.v;
  0.0 = signalVoltage1.p.i + signalVoltage1.n.i;
  signalVoltage1.i = signalVoltage1.p.i;
end dcmotor;
```

We plot part of the simulated result:

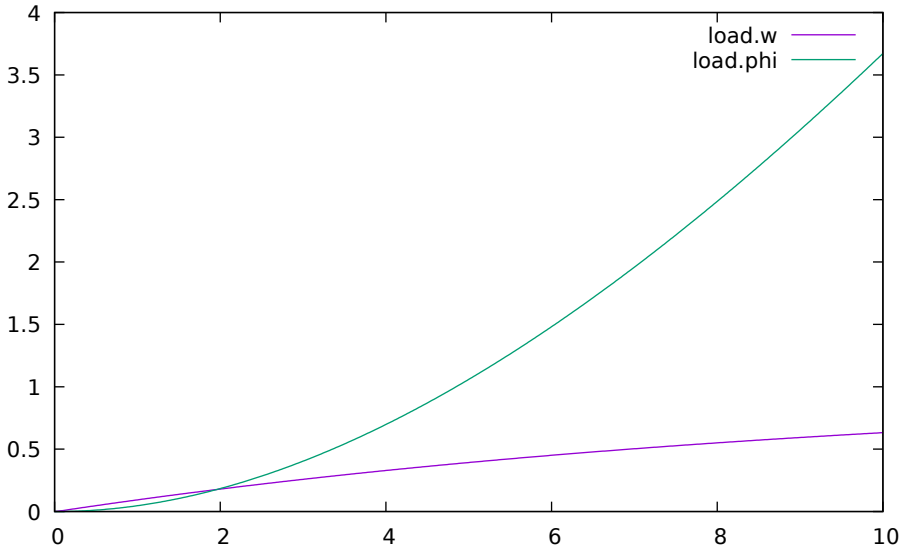


Figure 1.2: Rotation and rotational velocity of the DC motor

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

```
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(fixed = true, start = 1) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(fixed = true, start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  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;
  annotation(
    Documentation(figures = {Figure(title = "Bouncing ball", identifier = "height",␣
↪preferred = true, plots = {Plot(curves = {Curve(y = h, legend = "Height of ball")})}
↪)}));
end BouncingBall;
```

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

```
>>> writeFile("sim_BouncingBall.mos", "
  loadFile(getInstallationDirectoryPath() + \"/share/doc/omc/testmodels/BouncingBall.
↪mo\");
  simulate(BouncingBall, stopTime=3.0);
  /* plot({h,flying}); */
")
true
>>> runScript("sim_BouncingBall.mos")
"true
record SimulationResult
    resultFile = \"«DOCHOME»/BouncingBall_res.mat\",
    simulationOptions = \"startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'BouncingBall', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''\",
    messages = \"LOG_SUCCESS       | info    | The initialization finished␣
```

(continues on next page)

(continued from previous page)

```
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
\",
    timeFrontend = 0.0012986710000000002,
    timeBackend = 0.005585579,
    timeSimCode = 0.0017168370000000001,
    timeTemplates = 0.002481014,
    timeCompile = 0.175896506,
    timeSimulation = 0.007683774000000001,
    timeTotal = 0.19470688400000002
end SimulationResult;
"
```

```
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;
```

```
>>> simulate(Switch, startTime=0, stopTime=1)
record SimulationResult
    resultFile = "«DOCHOME»/Switch_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'Switch', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 9.80002e-4,
    timeBackend = 0.009669466,
    timeSimCode = 0.001717598,
    timeTemplates = 0.002113583,
    timeCompile = 0.158802043,
    timeSimulation = 0.0067139700000000005,
    timeTotal = 0.18004868100000002
end SimulationResult;
```

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

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

Plot itot and open:

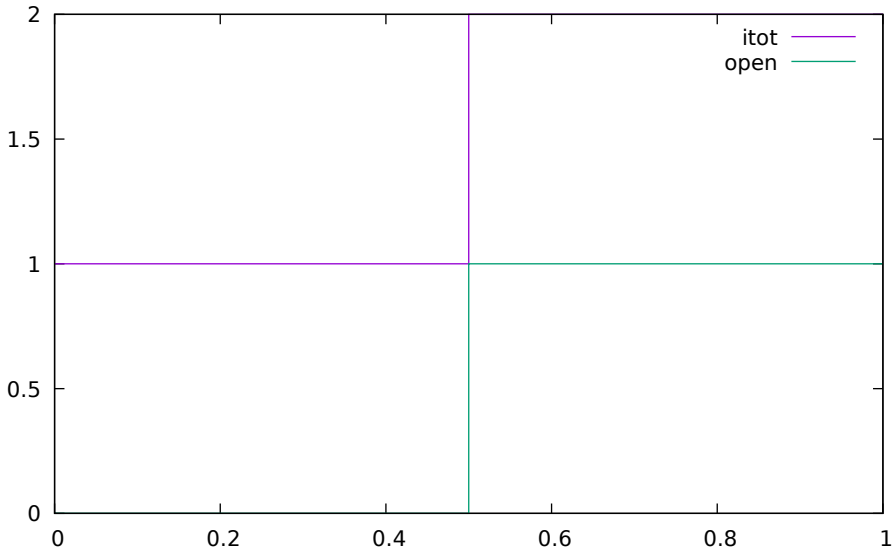


Figure 1.3: Plot when the switch opens

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

It is simulated:

```
>>> simulate(VanDerPol, stopTime=80)
record SimulationResult
    resultFile = "«DOCHOME»/VanDerPol_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 80.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'VanDerPol', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0010160500000000001,
    timeBackend = 0.0034888280000000002,
```

(continues on next page)

(continued from previous page)

```
    timeSimCode = 8.416220000000001e-4,
    timeTemplates = 0.001599797,
    timeCompile = 0.16946951500000002,
    timeSimulation = 0.0065658520000000005,
    timeTotal = 0.18304051400000002
end SimulationResult;
```

It is plotted:

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

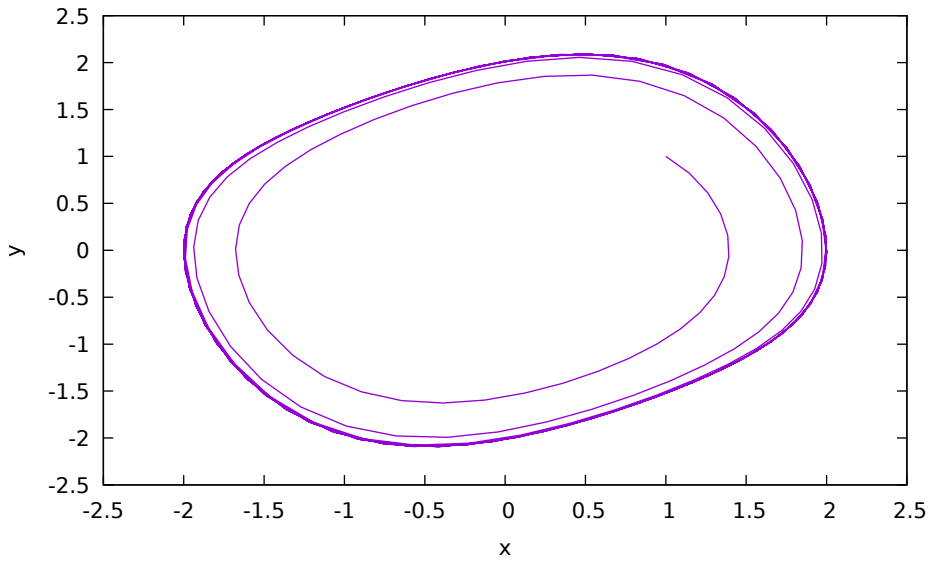


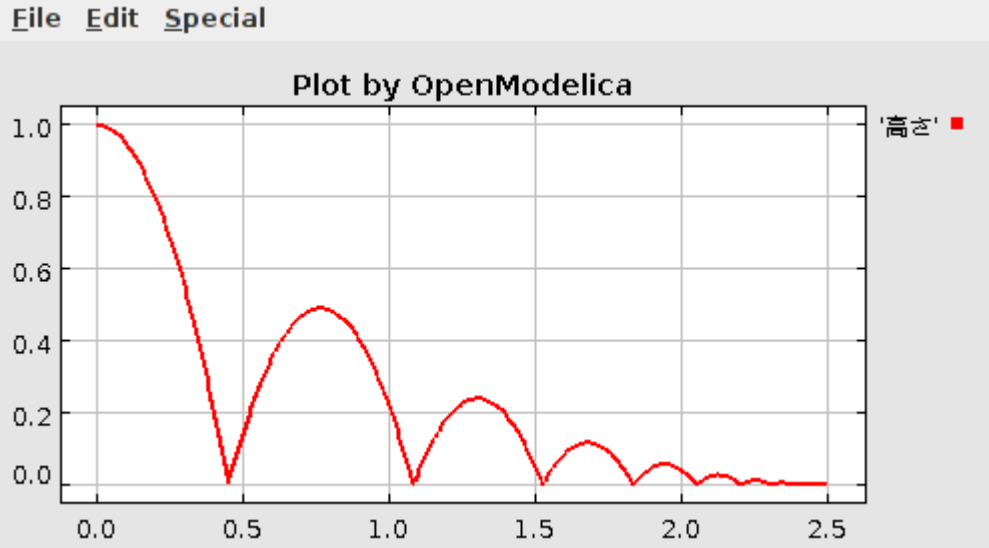
Figure 1.4: VanDerPol plotParametric(x,y)

Perform code instantiation to flat form of the VanDerPol model:

```
>>> instantiateModel(VanDerPol)
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1.0, fixed = true);
  Real y(start = 1.0, fixed = true);
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = lambda * (1.0 - x * x) * y - x;
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
1997.4500000000003
1479.0900000000001
```

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;
```

(continues on next page)

(continued from previous page)

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

Normal while-loop with concatenation of 10 "abc " strings:

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

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

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

An if-then-else statement with elseif:

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

Take a look at the variables a and b:

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

1.2.12 Variables, Functions, and Types of Variables

Assign a vector to a variable:

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

Type in a function:

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

Call the function:

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

Look at the value of variable a:

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

Look at the type of a:

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

Retrieve the type of b:

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

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

```
>>> typeOf(mySqr)
```

List the available variables:

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

Clear again:

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

1.2.13 Getting Information about Error Cause

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

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

1.2.14 Alternative Simulation Output Formats

There are several output format possibilities, with mat being the default. plt and mat are the only formats that allow you to use the val() or plot() functions after a simulation. Compared to the speed of plt, mat is roughly 5 times 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 or Octave.

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

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

// Default, match everything

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

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

// of the letters 1 through 3

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

// match x or y or z

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

1.2.15 Using External Functions

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

1.2.16 Using Parallel Simulation via OpenMP Multi-Core Support

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

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

1.2.17 Loading Specific Library Version

There exist many different 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;
```

Note

Notification: Automatically loaded package Complex 4.1.0 due to uses annotation from Modelica.

Notification: Automatically loaded package ModelicaServices 4.1.0 due to uses annotation from Modelica.

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

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

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

Note

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

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

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

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

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

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

Note

Notification: Automatically loaded package Complex 4.1.0 due to uses annotation from Modelica.

Notification: Automatically loaded package ModelicaServices 4.1.0 due to uses annotation from Modelica.

Notification: Automatically loaded package Modelica default due to usage.

1.2.18 Calling the Model Query and Manipulation API

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

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

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.
↪mo");
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
```

(continues on next page)

(continued from previous page)

```
  parameter Real g = 9.81 "gravity acceleration";
  Real h(fixed = true, start = 1) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(fixed = true, start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  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;
  annotation(
    Documentation(figures = {Figure(title = "Bouncing ball", identifier = "height",␣
↪preferred = true, plots = {Plot(curves = {Curve(y = h, legend = "Height of ball")})}
↪)}));
end BouncingBall;
```

Different kinds of calls with returned results:

```
>>> getClassRestriction(BouncingBall)
"model"
>>> getClassInformation(BouncingBall)
("model", "", false, false, false, "/var/lib/jenkins/ws/OpenModelica/build/share/doc/
↪omc/testmodels/BouncingBall.mo", false, 1, 1, 26, 17, {}, false, false, "", "",␣
↪false, "", "", "", "", "")
>>> isFunction(BouncingBall)
false
>>> existClass(BouncingBall)
true
>>> getComponents(BouncingBall)
{{Real, e, "coefficient of restitution", "public", false, false, false, false,
↪"parameter", "none", "unspecified", {}}, {Real, g, "gravity acceleration", "public",
↪ false, false, false, false, "parameter", "none", "unspecified", {}}, {Real, h,
↪"height of ball", "public", false, false, false, false, "unspecified", "none",
↪"unspecified", {}}, {Real, v, "velocity of ball", "public", false, false, false,␣
↪false, "unspecified", "none", "unspecified", {}}, {Boolean, flying, "true, if ball␣
↪is flying", "public", false, false, false, false, "unspecified", "none",
↪"unspecified", {}}, {Boolean, impact, "", "public", false, false, false, false,
↪"unspecified", "none", "unspecified", {}}, {Real, v_new, "", "public", false, false,
↪ false, false, "unspecified", "none", "unspecified", {}}, {Integer, foo, "", "public
↪", false, false, false, false, "unspecified", "none", "unspecified", {}}}
>>> getConnectionCount(BouncingBall)
0
>>> getInheritanceCount(BouncingBall)
0
>>> getComponentModifierValue(BouncingBall,e)
"0.7"
>>> getComponentModifierNames(BouncingBall,"e")
{}
>>> getClassRestriction(BouncingBall)
```

(continues on next page)

(continued from previous page)

```
"model"
>>> getVersion() // Version of the currently running OMC
"v1.28.0-dev.838+g4c2a253e5a.cmake"
```

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:

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

```
% Adjacency Matrix
% ====================================
% number of rows: 6
IM={{4,1},{6,{'if', 'true','==' {4},{},}},{{'if', 'true','==' {3},{},}},{2},{5,{'if',
↪'edge(impact)' {4},{2},}},{3,5}};
VL = {'h','v','flying','impact','v_new','foo'};


EqStr = {'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; end when;','when {h <= 0.0 and v <= 0.
↪0, impact} then flying = v_new > 0.0; end when;'};


OldEqStr={'class BouncingBall','  parameter Real e = 0.7 "coefficient of restitution";
↪','  parameter Real g = 9.81 "gravity acceleration";','  Real h(start = 1.0, fixed␣
↪= true) "height of ball";','  Real v(fixed = true) "velocity of ball";','  Boolean␣
↪flying(start = true, fixed = true) "true, if ball is flying";','  Boolean impact;',
↪'  Real v_new(fixed = true);','  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)␣
```

(continues on next page)

(continued from previous page)

```
↪= 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", "cvode", gbode and "euler" are also available. Output format "mat" is default. "plt" and "mat" (MATLAB) are the only ones that work with the val() command, "csv" (comma separated values) and "empty" (no output) are also available (see section *Alternative Simulation Output Formats*).

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

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

cd() Return the current directory.

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

clear() Clear all loaded definitions.

clearVariables() Clear all defined variables.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

quit() Leave and quit the OpenModelica environment

1.4 Running the compiler from command line

The OpenModelica compiler can also be used from command line, in Windows cmd.exe or a Unix shell. The following examples assume omc is on the PATH; if it is not, you can run `C:\OpenModelica 1.16.0\build\bin\omc.exe` or similar (depending on where you installed OpenModelica).

1.4.1 Example Session 1 - obtaining information about command line parameters

```
$ omc --help
OpenModelica Compiler v1.28.0-dev.838+g4c2a253e5a.cmake
Copyright © 2019 Open Source Modelica Consortium (OSMC)
Distributed under OSMC-PL and AGPL3, see www.openmodelica.org

Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]
* Libraries: Fully qualified names of libraries to load before processing Model or␣
↪Script.
...
Documentation is available in the built-in package OpenModelica.Scripting or
online <https://build.openmodelica.org/Documentation/OpenModelica.Scripting.html>.
```

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

```
model TestModel
  parameter Real x = 1;
end TestModel;
```

```
$ omc TestModel.mo
class TestModel
  parameter Real x = 1.0;
end TestModel;
```

1.4.3 Example Session 3 - create a mos-script and run omc on it

```
loadModel(Modelica);
getErrorString();
simulate(Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum);
getErrorString();
```

```
$ omc TestScript.mos
true
""
record SimulationResult
    resultFile = "/var/lib/jenkins/ws/OpenModelica/build_usersguide/source/Modelica.
↪Mechanics.MultiBody.Examples.Elementary.Pendulum_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 5.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'Modelica.Mechanics.MultiBody.
```

(continues on next page)

(continued from previous page)

```
↪Examples.Elementary.Pendulum', options = '', outputFormat = 'mat', variableFilter =
↪'.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.049697207,
    timeBackend = 0.259693761,
    timeSimCode = 0.028150204,
    timeTemplates = 0.025421115,
    timeCompile = 0.242029668,
    timeSimulation = 0.020041292000000002,
    timeTotal = 0.6251408270000001
end SimulationResult;
""
```

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

```
$ omc --showErrorMessages -d=failtrace TestScript.mos
InstFunction.getRecordConstructorFunction failed for OpenModelica.Scripting.loadModel
- Static.elabCrefSubs failed on: [top:<Prefix.NOPRE()>].<Prefix.NOPRE()>.Modelica␣
↪env: <global scope>
- Static.elabCref failed: Modelica in env: <global scope>
- Static.elabCrefSubs failed on: [top:<Prefix.NOPRE()>].<Prefix.NOPRE()>.Modelica␣
↪env: <global scope>
...
    timeTotal = 0.639681179
end SimulationResult;
"[/var/lib/jenkins/ws/OpenModelica/OMCompiler/Compiler/BackEnd/ExpressionSolve.
↪mo:195:9-195:216:writable] Error: Internal error Failed to solve \"world.z_label.
↪cylinders[2].lengthDirection[1] = world.z_label.cylinders[1].lengthDirection[1]\" w.
↪r.t. \"world.z_label.R_lines[1,1]\"
"
```

CHAPTER

TWO

PACKAGE MANAGEMENT

2.1 Overview of Basic Modelica Package Management Concepts

The Modelica language promotes the orderly reuse of component models by means of packages that contain structured libraries of reusable models. The most prominent example is the Modelica Standard Library (MSL), that contains basic models covering many fields of engineering. Other libraries, both open-source and commercial, are available to cover specific applications domains.

When you start a simulation project using Modelica, it is common practice to collect all related system models in a project-specific package that you develop. The models in this package are often instantiated (e.g. by drag-and-drop in OMEdit) from released libraries, which are read-only for your project. This establishes a dependency between your project package and a certain version of a read-only package (or library), which is the one you have loaded in OMEdit and that you drag-and-drop components from.

This dependency is automatically marked in your package by adding a uses annotation at the top level. For example, if you drag and drop components from MSL 4.0.0 into models of your package, the `annotation(uses(Modelica(version="4.0.0")));` will be added automatically to it. This information allows OpenModelica to automatically load all the libraries that are required to compile the models in your own package next time you (or someone else, possibly on a different computer) loads your package, provided they are installed in places on the computer's file system where OpenModelica can find them.

The default place where OpenModelica looks for packages is set by environment variable `OPENMODELICALIBRARY`. The variable MODELICAPATH from the language specification is ignored by OpenModelica, see *FAQ OpenModelica General*. You can check where it is by typing `getModelicaPath()` in the Interactive Environment (Tools | OpenModelica Compiler CLI in OMEdit), or by browsing the General group under Tools|Options|Libraries. Installed read-only libraries are all placed by default in the `OPENMODELICALIBRARY`.

However, when you open a package directly from the file system, OpenModelica will also look for packages it depends upon in the same directory that contains the package you just opened. For example, if you open `/home/John/ModelicaPackages/MoonShot/package.mo`, and your MoonShot package contains `annotation(uses(Rockets));`, OpenModelica will also check `/home/John/ModelicaPackages/Rockets/package.mo` and `/home/John/ModelicaPackages/Rockets.mo`. So, if you are developing several packages with dependencies among them, you can place them in the same common root directory to make sure that all the dependencies are loaded automatically, without the need of putting them in the `OPENMODELICALIBRARY`, or to change it to include that directory.

Please note that if the `uses` annotation refers to a specific version of a package, that package will only be loaded if the name of the directory or of the single file that contains it also indicates the version number, as allowed by the Modelica Specification, Section 18.8.3. For example, if MoonShot contains `annotation(uses(Rockets(version = "2.0.0"));`, OpenModelica will try to load `/home/John/ModelicaPackages/Rockets 2.0.0/package.mo` or `/home/John/ModelicaPackages/Rockets 2.0.0.mo`; in this case, packages without the version number in their root directory, such as `/home/John/ModelicaPackages/Rockets/package.mo`, will be ignored. All installed packages in the `OPENMODELICALIBRARY` include version numbers in their directory name, which also allows to install multiple versions of the same library.

When a new version of certain package comes out, conversion annotations in it declare whether your models using a certain older version of it can be used as they are with the new one, which is then 100% backwards-compatible, or whether they need to be upgraded by running a conversion script, provided with the new version

of the package. The former case is declared explicitly by a `conversion(noneFromVersion)` annotation. For example, a `conversion(noneFromVersion="3.0.0")` annotation in version `3.1.0` of a certain package means that all packages using version `3.0.0` can use `3.1.0` without any change. Of course it is preferable to use a newer, backwards-compatible version, as it contains bugfixes and possibly new features.

Hence, if you install a new version of a library which is 100% backwards-compatible with the previous ones, all your models that used the old one will automatically load and use the new one, without the need of any further action.

If the new version is not backwards-compatible, instead, you will need to create a new version of your library that uses it, by running the provided conversion scripts.

OpenModelica has a package manager that can be used to install and update libraries on your computer, and is able to run conversion scripts. Keep in mind there are three stages in package usage: *available* packages are indexed on the OSMC servers and can be downloaded from public repositories; *installed* packages are stored in the `OPENMODELICALIBRARY` of your computer; *loaded* packages are loaded in memory in an active OMC session, either via the Interactive Environment, or via the OMEdit GUI, where they are shown in the Libraries Browser. When you load a package, OpenModelica tries to load the best possible installed versions of all the dependencies declared in the uses annotation.

2.2 The Package Manager

The Open Source Modelica Consortium (OSMC) maintains a collection of publicly available, open-source Modelica libraries on its servers, see https://github.com/OpenModelica/OMPackageManager. These libraries are routinely tested with past released versions of OpenModelica, as well as with the current development version on the master branch, see the overview report. Based on the testing results and on information gathered from the library developers, these packages are classified in terms of level of support in OpenModelica. Backwards-compatibility information is also collected from the conversion annotations.

The OpenModelica Package Manager relies on this information to install the best versions of the library dependencies of your own, locally developed Modelica packages and models. It can be run both from the OMEdit GUI and from the command-line interactive environment. The libraries and their `index.json` index file with all the library metadata are installed in the `~/.openmodelica/libraries` directory under Linux and in the `%AppData%\.openmodelica\libraries` directory on Windows. Note that these directories are user-specific, so if there are multiple users on the same computer, each of them will install and manage his/her own set of libraries independently from the others.

The Package Manager may install multiple builds of the same library version in your own package manager directory, if they are indexed on the OSMC servers. When this happens, they are distinguished among each other by means of semver-style pre- or post-release metadata in the top directory name on the file system. Post-release builds are denoted by a plus sign (e.g. `2.0.0+build.02`) and have higher priority over the corresponding plain release (e.g. `2.0.0`), while pre-release builds are denoted by a minus sign (e.g. `2.0.0-dev.30`) and have a lower priority.

When loading a certain version of a library, unless a specific build is explicitly referenced, the one with higher precedence will always be loaded. For example, if the versions `2.0.0-beta.01`, `2.0.0`, and `2.0.0+build.01` are installed, the latter is loaded by libraries with uses annotation requiring version `2.0.0`. Unless, of course, there are later backwards-compatible versions installed, e.g., `2.0.1`, in which case the one with the highest release number and priority is installed.

In any case, semver version semantics is only used to order the releases, while backwards-compatibility is determined exclusively on the basis of `noneFromVersion` annotations.

When installing OpenModelica, a cached version of the latest versions of the Modelica Standard Library is included in the installation files. As soon as a user starts any OpenModelica tool (e.g., OMEdit, OMNotebook, OMShell, or direct command-line invocation of omc), if the user's `.openmodelica` directory is empty the Modelica Standard Library will be installed automatically using this cached version. This happens when using OpenModelica for the first time, or if the contents of the `.openmodelica` directory have been deleted to get rid of all installed libraries. This automatic installation needs no Internet connection, so it also works behind firewalls or in set-ups with limited available bandwidth. Therefore, the Modelica Standard Library is immediately available without the need of using

the package manager explicitly. It is then possible to install and manage other libraries using the package manager, as explained previously.

As a final remark, please note that the version numbers of the various Modelica packages have no relation with the version numbers of the OpenModelica tool itself. Since version 1.19.0, OpenModelica is no longer shipped with built-in installed libraries, that are instead managed independently by the user with the online Package Manager. You can install and use old and new versions of a certain open source Modelica library using the latest released version of OpenModelica, by using the Package Manager. We strive to make sure that new released versions of OpenModelica are backwards-compatible, meaning that you should always be able to run the same models/libraries with a new version of OpenModelica if you could with an older version of the tool. Hence, we strongly advise you to always use the latest released version of OpenModelica, even if you are running old models; by doing so, you benefit from faster performance, more robust numerical performance, new tool features, and a lot of bug fixes.

You should never find yourself in a situation where you are forced to stick to an old version of OpenModelica to run your models. If that happens to you, please open a ticket on the issue tracker, so we can hopefully fix the problem and allow you to keep using the latest OpenModelica release.

2.2.1 Package Management in OMEdit

Installing a new library in OMEdit.

2.2.2 Running Conversion Scripts in OMEdit

Converting a library in OMEdit.

2.2.3 Automatically Loaded Packages in OMEdit

When you start OMEdit, some packages can be automatically loaded into the environment, and shown in the Libraries Browser. You can configure which ones are loaded from the Tools|Options|Libraries menu.

Please note that automatically loaded libraries may be in conflict with the dependencies of packages that you may later load from the File menu. For example, if you automatically load Modelica `4.0.0`, and then load a library XYZ that still uses MSL `3.2.3`, you get a conflict, because Modelica `4.0.0` is not backwards-compatible with Modelica `3.2.3`, so XYZ cannot be used.

In this case you have two options:

- Cancel Operation: this means XYZ is not actually loaded, and all previously loaded libraries remain in place.
- Unload all and Reload XYZ: in this case, all previously loaded libraries, that may generate conflicts, are unloaded first; then XYZ is loaded, and finally the right versions of the libraries XYZ uses, as declared in its `uses` annotation, will be loaded automatically.

If you are normally working with only one version of the Modelica standard library, you can set it to be automatically loaded from the Tools|Options|Libraries menu; in case you need to work with a library that uses a previous, non-backwards compatible version, the Unload all and Reload option comes handy. Otherwise, you can avoid loading the Modelica library automatically upon starting OMEdit, and let the right version of the Modelica library be loaded automatically when you open the library you want to work with. In this case, if you want to get the Modelica library into the Package Browser to start developing a new library, you can do so easily from the Welcome tab, by clicking on the System Libraries button and selecting the version that you want to load.

2.2.4 Manually Loading Packages

If you want to maintain full control over which library dependencies are loaded, you can use the File | Open Model/Library Files(s) menu command in OMEdit to open the libraries one by one from specific locations in your file system. Note, however, that whenever a library is loaded, its dependencies, that are declared in its `uses` annotation, will automatically be loaded. If you want to avoid that, you need to load the library dependencies in reverse order, so that the intended library dependencies are already loaded when you open the library that needs them.

If you are using the Interactive Environment, you can use the `loadFile()` command to load libraries from specific locations on the file system, also in reverse dependency order, unless you also set the optional `uses = false` input argument to disable the automatic loading of dependencies.

2.2.5 Using the Package Manager from the Interactive Environment

The Package Manager can also be used from the Interactive Environment command line shell. Here is a list of examples of relevant commands; please type them followed by *getErrorString()*, e.g., *updatePackageIndex()*; *getErrorString()*, in order to get additional information, notifications and error messages.

- *updatePackageIndex()* - this command puts the Package Manager in contact with the OSMC servers and updates the internally stored list of available packages;
- *getAvailablePackageVersions(Building, "")* - lists all available versions of the Buildings library on the OSMC server, starting from the most recent one, in descending order of priority. Note that pre-release versions have lower priority than all other versions;
- *getAvailablePackageVersions(Building, "7.0.0")* - lists all available versions of the Buildings library on the OSMC server that are backwards-compatible with version `7.0.0`, in descending order of priority;
- *installPackage(Buildings, "")* - install the most recent version of the Building libraries, *and all its dependencies*;
- *installPackage(Buildings, "7.0.0")* - install the most recent version of the Building libraries which is backwards-compatible with version `7.0.0`, *and all its dependencies*;
- *installPackage(Buildings, "7.0.0", exactMatch = true)* - install version `7.0.0` even if there are more recent backwards-compatible versions available, *and all its dependencies*;
- *upgradeInstalledPackages(installNewestVersions = true)* - installs the latest available version of all installed packages.

2.3 How the package index works

The package index is generated by OMPackageManager on an OSMC server, based on these settings. See its documentation to see how to add new packages to the index, change support level, and so on.

The index is generated by scanning git repositories on github. All tags and optionally some specific branches are scanned. The tag name is parsed as if it was a semantic version, with prerelease and metadata of the tag added to the version of Modelica packages in the repository. If the tag name is not a semantic version, it is sorted differently.

Packages are sorted as follows:

- Support level: each package is given a level of support in the index
- Semantic version: according to the semver specification, but build metadata is also considered (sorted the same way as pre-releases)
- Non-semantic versions: alphabetically

CHAPTER

THREE

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 graphical user interface library and supports the Modelica Standard Library 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.

3.1 Starting OMEdit

A splash screen similar to the one shown in Figure 3.1 will appear indicating that it is starting OMEdit. The executable is found in different places depending on the platform (see below).

3.1.1 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.

3.1.2 Linux

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

3.1.3 Mac OS X

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

3.2 MainWindow & Browsers

The MainWindow contains several dockable browsers,

- Libraries Browser
- Documentation Browser

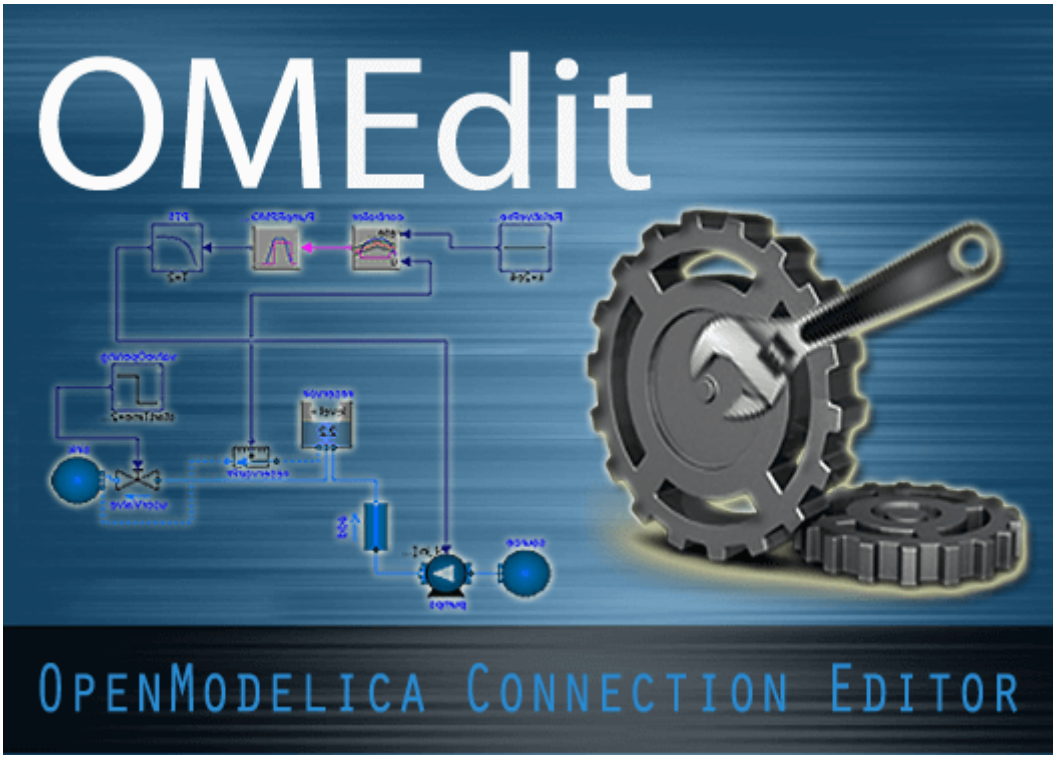


Figure 3.1: OMEdit Splash Screen.

- Variables Browser
- Messages Browser

Figure 3.2 shows the MainWindow and browsers.

The default location of the browsers are shown in Figure 3.2. All browsers except for Message Browser can be docked into left or right column. The Messages Browser can be docked into top or bottom areas. If you want OMEdit to remember the new docked position of the browsers then you must enable Preserve User's GUI Customizations option, see section *General Options*.

3.2.1 Filter Classes

To filter a class click Edit > Filter Classes or press keyboard shortcut Ctrl+Shift+F. The loaded Modelica classes can be filtered by typing any part of the class name.

3.2.2 Libraries Browser

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

3.2.3 Documentation Browser

Displays the HTML documentation of Modelica classes. It contains the navigation buttons for moving forward and backward. It also contains a WYSIWYG editor which allows writing class documentation in HTML format. To view the Documentation Browser click View > Windows > Documentation Browser.

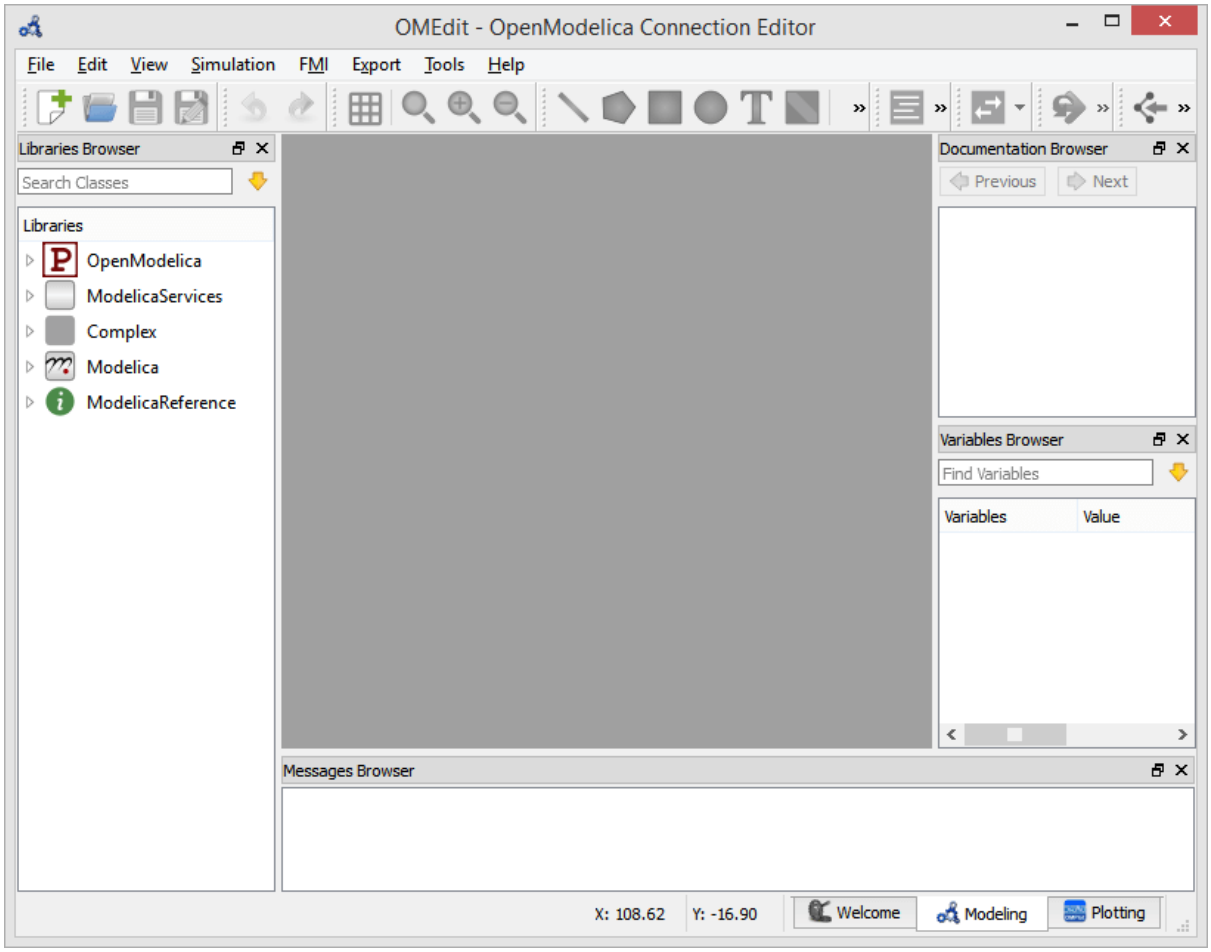


Figure 3.2: OMEdit MainWindow and Browsers.

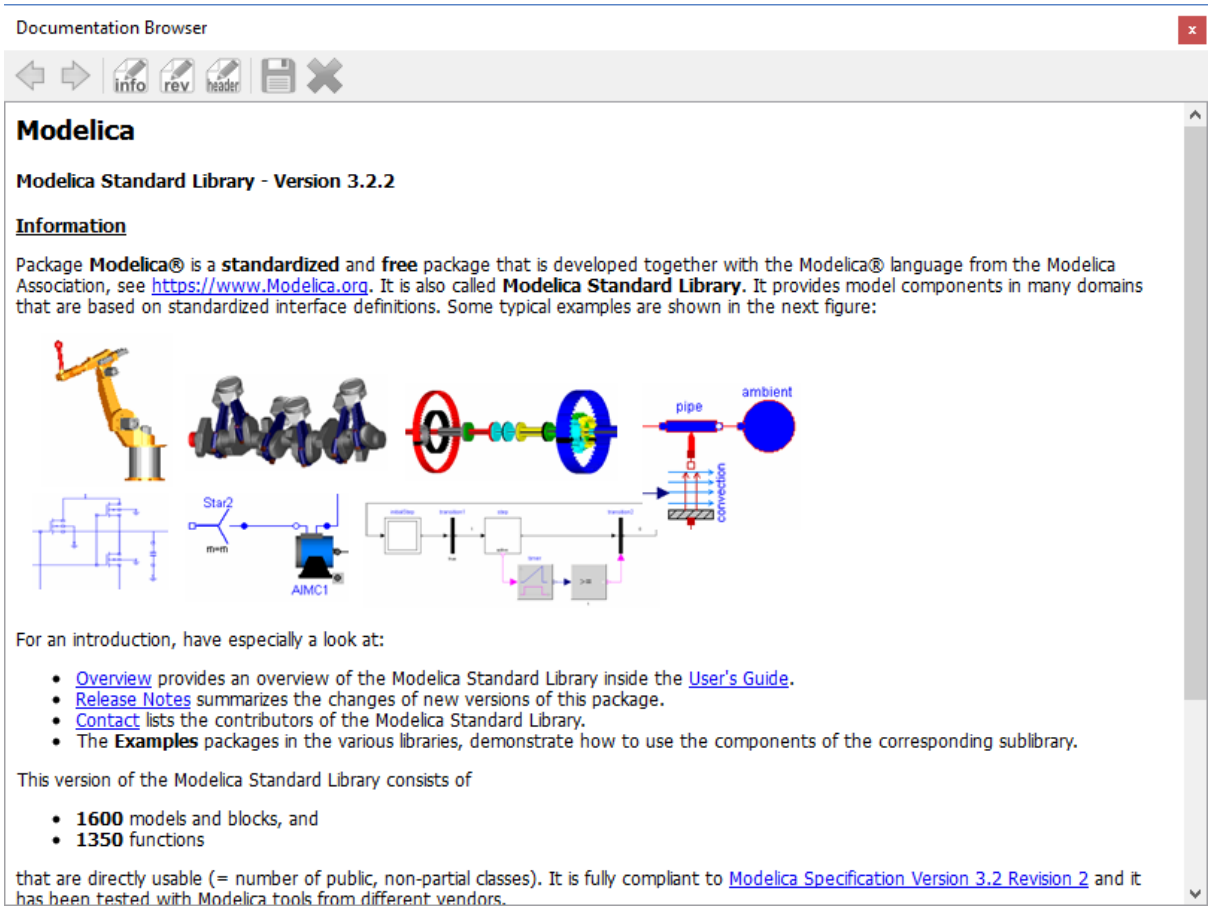


Figure 3.3: Documentation Browser.

3.2.4 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. The complete Variables Browser can be collapsed and expanded using the Collapse All and Expand All buttons.

There is a find box for filtering the variable in the tree. By clicking the yellow down arrow you can set all the filtering options. The filtering can be done using Regular Expression, Wildcard and Fixed String; in all three cases, all variables whose full name *contains* a string corresponding to the filter string will be displayed.

Fixed String: shows all variables whose name contains the string verbatim

- `abc` shows `abc`, `abc.def`, `xyz.abc`, `der(abc)` etc.
- `a.b` shows `a.b`, `a.bcd`, `a.b.c`, `x.a.b`, `x.a.b.c`, etc.

Wildcard: same as Fixed String; additionally, asterisks match any number of characters

- `der(*)` shows all derivatives, e.g. `der(x)`, `der(abc)`, `abc.der(xyz)`, etc.
- `a*c` shows `ac`, `abc`, `abdc`, `xyz.adefc`, etc.

Regular expression: shows all variables whose name contain a string that matches the regexp; if the regexp ends with `$`, then the name must end with a string matching the regexp

- `abc` shows `abc`, `abc.def`, `xyz.abc`, `der(abc)` etc.
- `abc$` shows `abc`, `xyz.abc` only
- `a.c` shows `abc`, `abc.def`, `azc`, `xyz.adc` etc. (`.` matches any character)
- `a.*c` shows `abc`, `abc.def`, `axyc`, `xyz.axxxxdc` etc. (`.*` matches any number of character)
- `body\.a_0\[1\]` shows variables containing `body.a_0[1]`. Note that `.`, `[`, and `]` are special regexp characters, so they must be escaped
- `der\(.*\)` shows all derivatives in the model. Note that `(` and `)` must be escaped
- `x\[[2-4]\]` shows elements 2, 3, and 4 of arrays `x[:]`, `abc.x[:]`, `x[:].abc`
- `x\[.*\]` shows all elements of arrays `x[:]`, `abc.x[:]`, `x[:].abc`
- `abc|def` shows all variables with names containing either `abc` or `def`

The browser allows manipulation of changeable parameters for *Plot Window*. It also displays the unit and description of the variable.

The browser also contains the slider and animation buttons. These controls are used for variable graphics and schematic animation of models i.e., DynamicSelect annotation. They are also used for debugging of state machines. Open the *Diagram Window* for animation. It is only possible to animate one model at a time.

3.2.5 Messages Browser

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

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

See section *Messages Options* for Messages Browser options.

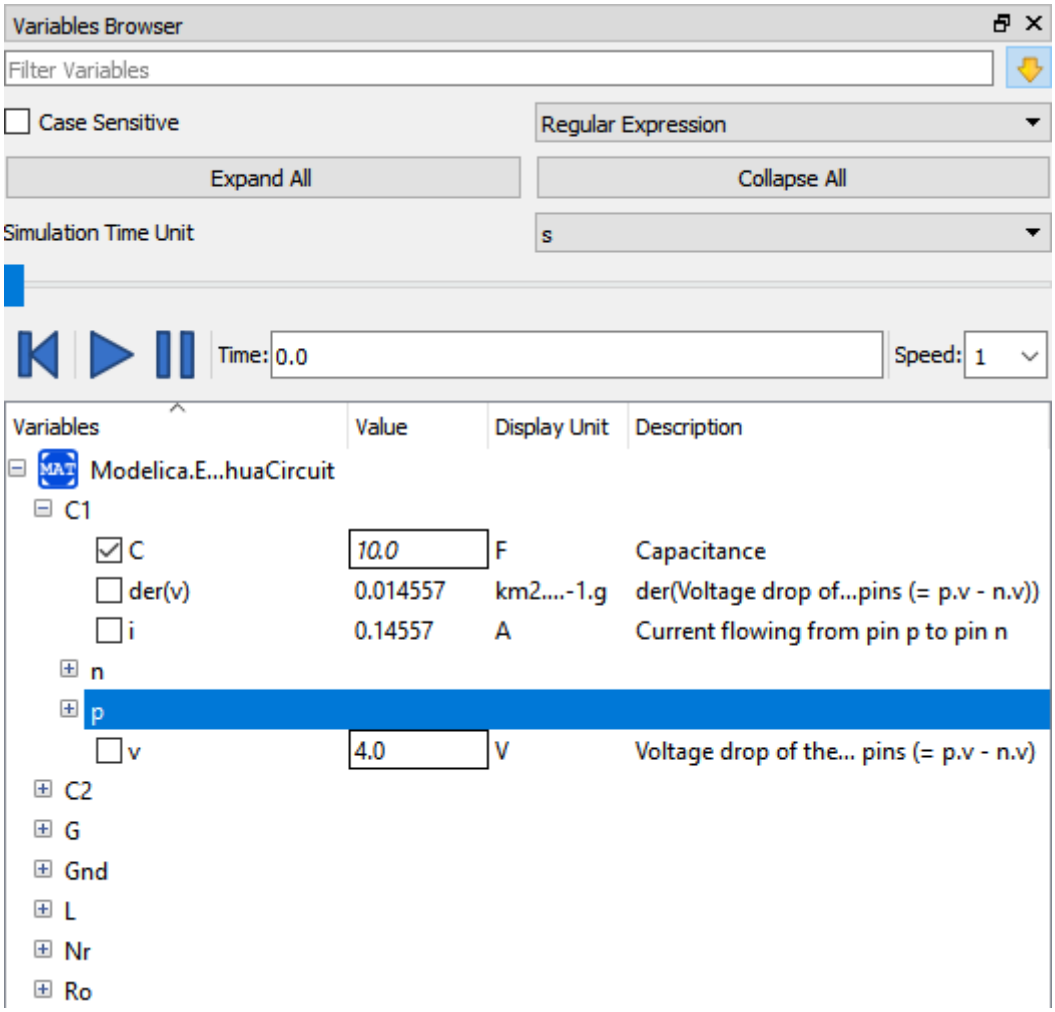


Figure 3.4: Variables Browser.

3.3 Perspectives

The perspective tabs are located at the bottom right of the Main Window:

- Welcome Perspective
- Modeling Perspective
- Plotting Perspective
- Debugging Perspective

3.3.1 Welcome Perspective

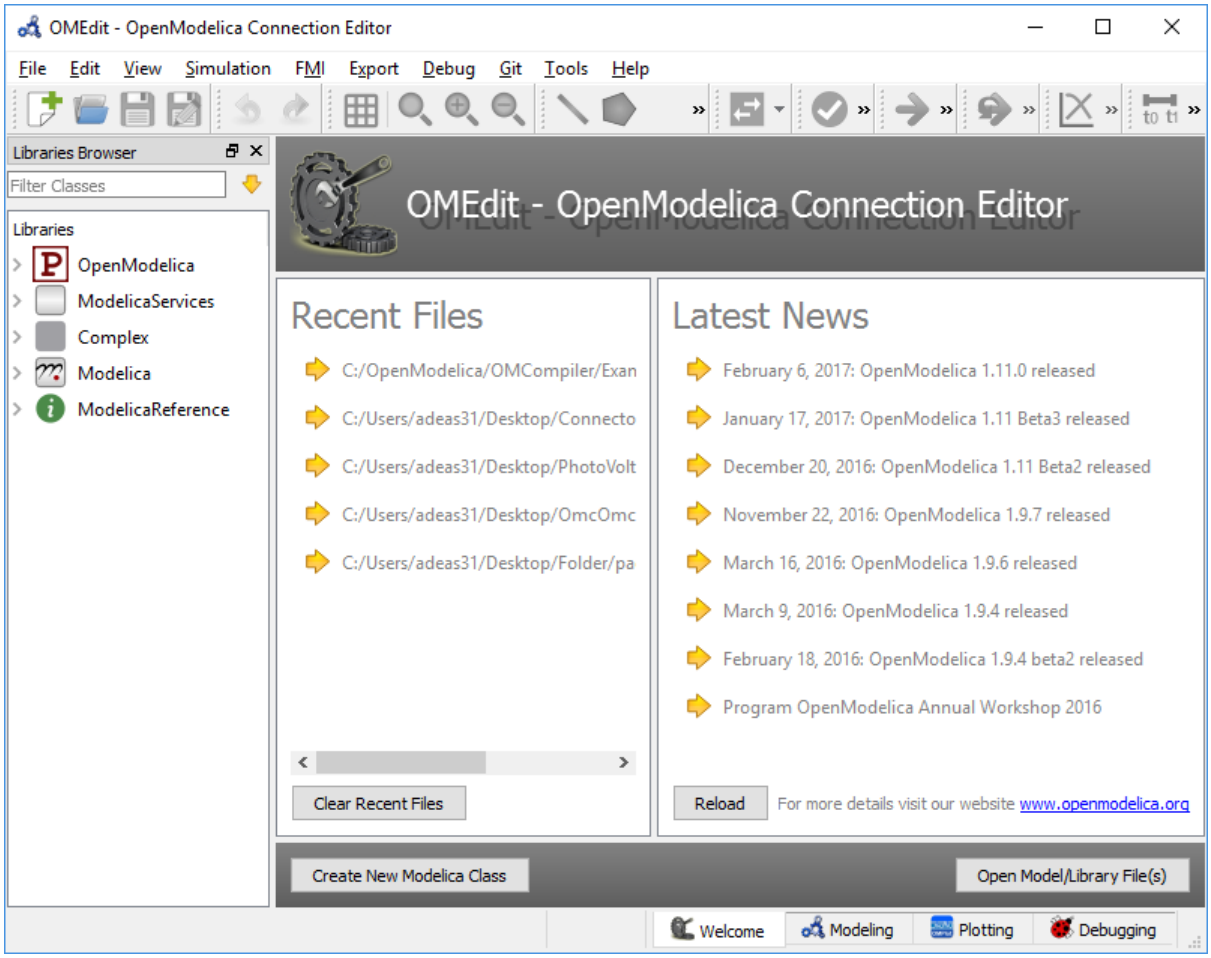


Figure 3.5: OMEdit Welcome Perspective.

The Welcome Perspective shows the list of recent files and the list of latest news from https://www.openmodelica.org. See Figure 3.5. The orientation of recent files and latest news can be horizontal or vertical. User is allowed to show/hide the latest news. See section *General Options*.

3.3.2 Modeling Perspective

The Modeling Perspective provides the interface where user can create and design their models. See Figure 3.6.

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

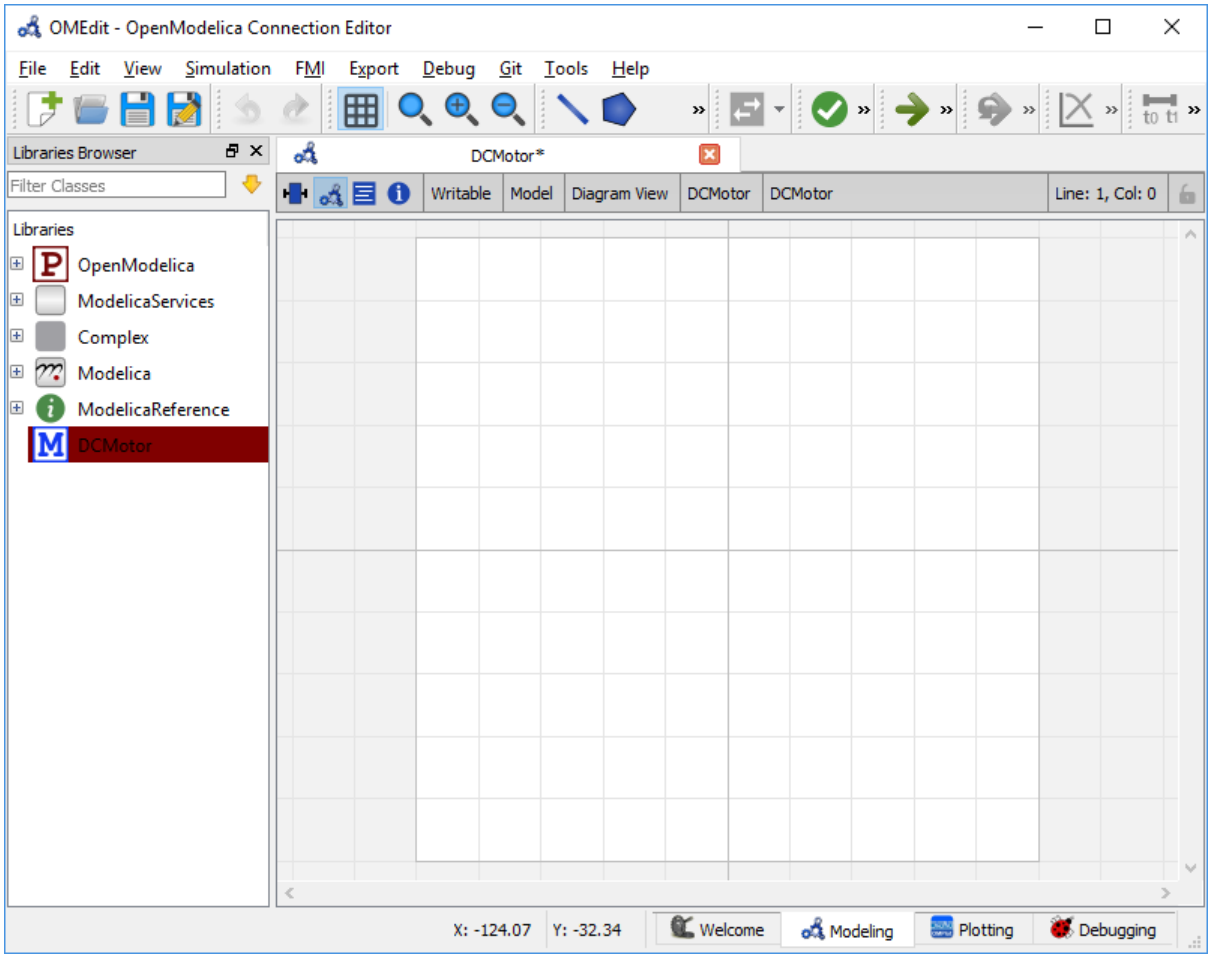


Figure 3.6: OMEdit Modeling Perspective.

3.3.3 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 sub-window view, see section *General Options*.

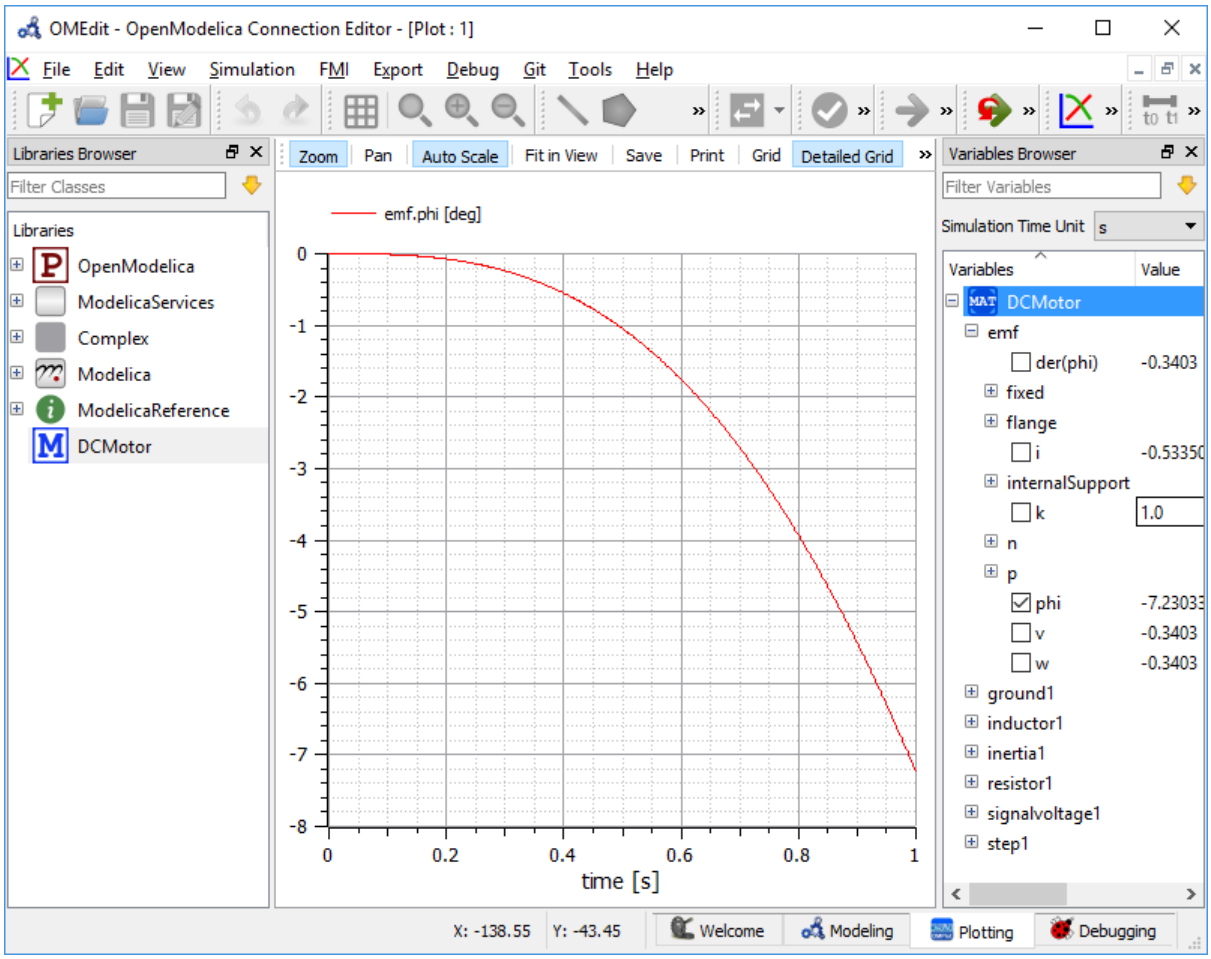


Figure 3.7: OMEdit Plotting Perspective.

3.3.4 Debugging Perspective

The application automatically switches to Debugging Perspective when user simulates the class with algorithmic debugger. The perspective shows the list of stack frames, breakpoints and variables.

3.4 File Menu

- *New*
- *New Modelica Class* - Creates a new Modelica class.
- *New SSP Model* - Creates a new SSP model.
- *Open Model/Library File(s)* - Opens the Modelica file or a library.
- *Open/Convert Modelica File(s) With Encoding* - Opens the Modelica file or a library with a specific encoding. It is also possible to convert to UTF-8.
- *Load Library* - Loads a Modelica library. Allows the user to select the library path assuming that the path contains a package.mo file.

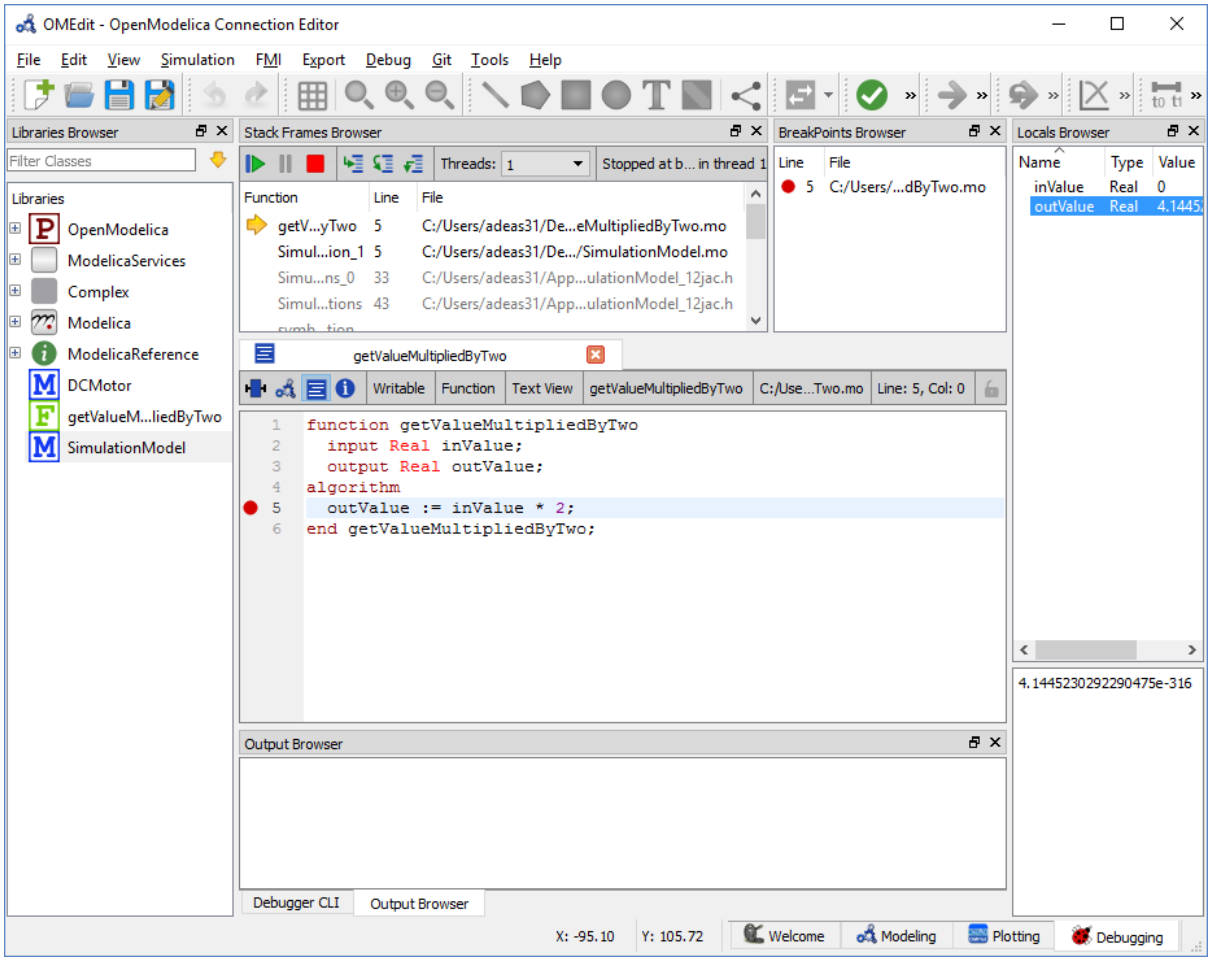


Figure 3.8: OMEdit Debugging Perspective.

- *Load Encrypted Library* - Loads an encrypted library. see *OpenModelica Encryption*
- *Open Result File(s)* - Opens a result file.
- *Open Transformations File* - Opens a transformational debugger file.
- *Unload All* - Unloads all loaded classes.
- *Open Directory* - Loads the files of a directory recursively. The files are loaded as text files.
- *Save* - Saves the class.
- *Save As* - Save as the class.
- *Save Total* - Saves the class and all the classes it uses in a single file. The class and its dependencies can only be loaded later by using the *loadFile()* API function in a script. Allows third parties to reproduce an issue with a class without worrying about library dependencies.
- *Import*
- *FMU* - Imports the FMU.
- *FMU Model Description* - Imports the FMU model description.
- *From OMNotbook* - Imports the Modelica models from OMNotebook.
- *Ngspice netlist* - Imports the ngspice netlist to Modelica code.
- *Export*
- *To Clipboard* - Exports the current model to clipboard.
- *Image* - Exports the current model to image.
- *FMU* - Exports the current model to FMU.
- *Read-only Package* - Exports a zipped Modelica library with file extension .mol
- *Encrypted Package* - Exports an encrypted package. see *OpenModelica Encryption*
- *XML* - Exports the current model to a xml file.
- *Figaro* - Exports the current model to Figaro.
- *To OMNotebook* - Exports the current model to a OMNotebook file.
- *System Libraries* - Contains a list of system libraries.
- *Manage Libraries*
- *Install Library* - Opens a dialog to select and install a new library, see *Install Library*
- *Upgrade Installed Libraries* - Opens a dialog to upgrade the installed libraries.
- *Update Library Index* - Updates the library index.
- *Recent Files* - Contains a list of recent files.
- *Clear Recent Files* - Clears the list of recent files.
- *Print* - Prints the current model.
- *Quit* - Quit the OpenModelica Connection Editor.

3.5 Edit Menu

- *Undo* - Undoes the last change.
- *Redo* - Redoes the last undone change.
- *Filter Classes* - Filters the classes in Libraries Browser, see *Filter Classes*

3.6 View Menu

- *Toolbars* - Toggle visibility of toolbars.
- *Windows* - Toggle visibility of windows.
- *Close Window* - Closes the current model window.
- *Close All Windows* - Closes all the model windows.
- *Close All Windows But This* - Closes all the model windows except the current.
- *Cascade Windows* - Arranges all the child windows in a cascade pattern.
- *Tile Windows Horizontally* - Arranges all child windows in a horizontally tiled pattern.
- *Tile Windows Vertically* - Arranges all child windows in a vertically tiled pattern.
- *Toggle Tab/SubWindow View* - Switches between tab and sub-window view.
- *Grid Lines* - Toggle grid lines of the current model.
- *Reset Zoom* - Resets the zoom of the current model.
- *Zoom In* - Zoom in the current model.
- *Zoom Out* - Zoom out the current model.
- *Fit to Diagram* - Fit the current model diagram in the view.

3.7 SSP Menu

- *Add System* - Adds the system to a model.
- *Add/Edit Icon* - Add/Edit the system/submodel icon.
- *Delete Icon* - Deletes the system/submodel icon.
- *Add Connector* - Adds a connector to a system/submodel.
- *Add Bus* - Adds a bus to a system/submodel.
- *Add TLM Bus* - Adds a TLM bus to a system/submodel.
- *Add SubModel* - Adds a submodel to a system.

3.8 Simulation Menu

- *Check Model* - Checks the current model.
- *Check All Models* - Checks all the models of a library.
- *Instantiate Model* - Instantiates the current model.
- *Simulation Setup* - Opens the simulation setup window.
- *Simulate* - Simulates the current model.
- *Simulate with Transformational Debugger* - Simulates the current model and opens the transformational debugger.
- *Simulate with Algorithmic Debugger* - Simulates the current model and opens the algorithmic debugger.
- *Simulate with Animation* - Simulates the current model and open the animation.
- *Archived Simulations* - Shows the list of simulations already finished or running. Double clicking on any of them opens the simulation output window.

3.9 Data Reconciliation

- *Calculate Data Reconciliation* - Opens the dialog to run the data reconciliation algorithm.

3.10 Sensitivity Optimization Menu

- *Run Sensitivity Analysis and Optimization* - Runs the sensitivity analysis and optimization.

3.11 Debug Menu

- *Debug Configurations* - Opens the debug configurations window.
- *Attach to Running Process* - Attaches the algorithmic debugger to a running process.

3.12 Tools Menu

- *OpenModelica Compiler CLI* - Opens the OpenModelica Compiler command line interface window.
- *OpenModelica Command Prompt* - Opens the OpenModelica Command Prompt (Only available on Windows).
- *Open Temporary Directory* - Opens the current temporary directory.
- *Open Working Directory* - Opens the current working directory.
- *Open Terminal* - Runs the terminal command set in *General Options*.
- *Options* - Opens the options window.

3.13 Help Menu

- *OpenModelica User's Guide* - Opens the OpenModelica User's Guide.
- *OpenModelica User's Guide (PDF)* - Opens the OpenModelica User's Guide (PDF).
- *OpenModelica System Documentation* - Opens the OpenModelica System Documentation.
- *OpenModelica Scripting Documentation* - Opens the OpenModelica Scripting Documentation.
- *Modelica Documentation* - Opens the Modelica Documentation.
- *OMSimulator User's Guide* - Opens the OMSimulator User's Guide.
- *About OMEdit* - Shows the information about OpenModelica Connection Editor.

3.14 Modeling a Model

3.14.1 Creating a New Modelica Class

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

- Select File > New > 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.

3.14.2 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.

(Note, for editing Modelica system files like MSL (not recommended), see *Editing Modelica Standard Library*)

3.14.3 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.

3.14.4 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 3.9.

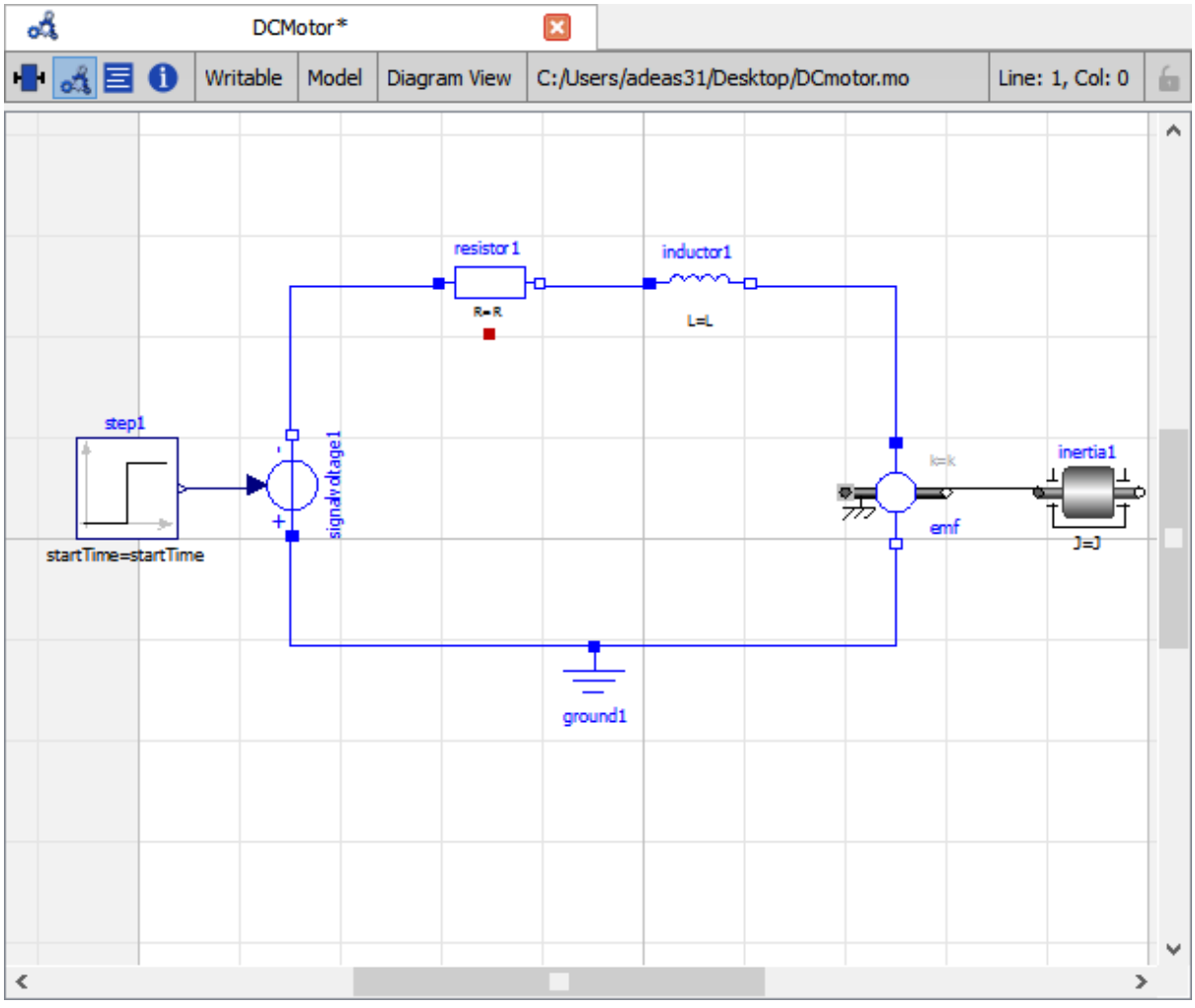


Figure 3.9: Model Widget showing the Diagram View.

3.14.5 Adding Component Models

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

3.14.6 Making Connections

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

Move the mouse over the connector. The mouse cursor will change from arrow cursor to cross cursor. To start the connection press left button and move while keeping the button pressed. Now release the left button. Move towards the end connector and click when cursor changes to cross cursor.

3.15 Simulating a Model

The simulation process in OMEdit is split into three main phases:

1. The Modelica model is translated into C/C++ code. The model is first instantiated by the frontend, which turns it into a flat set of variables, parameters, equations, algorithms, and functions. The backend then analyzes the mathematical structure of the flat model, applies symbolic simplifications and determines how the equations can be solved efficiently. Finally, based on this information, model-specific C/C++ code is generated. This part of the process can be influenced by setting *Translation Flags* (a.k.a. *Command Line Options*), e.g. deciding which kind of structural simplifications should be performed during the translation phase.
2. The C/C++ code is compiled and linked into an executable simulation code. Additional *C/C++ compiler flags* can be given to influence this part of the process, e.g. by setting compiler optimizations such as `-O3`. Since multiple C/C++ source code files are generated for a given model, they are compiled in parallel by OMEdit, exploiting the power of multi-core CPUs.
3. The simulation executable is started and produces the simulation results in a *.mat* or *.csv* file. The runtime behavior can be influenced by *Simulation Flags*, e.g. by choosing specific solvers, or changing the output file name. Note that it it possible to re-simulate a model multiple times, changing parameter values from the Variables Browser and/or changing some Simulation Flags. In this case, only Phase 3. is repeated, skipping Phases 1. and 2., which enables much faster iterations.

The simulation options for each model are stored inside the OMEdit data structure. They are set according to the following sequence,

- Each model has its own translation and simulation options.
- If the model is opened for the first time then the translation and simulation options are set to defaults, that can be customized in Tools | Options | Simulation.
- `experiment`, `__OpenModelica_commandLineOptions` and `__OpenModelica_simulationFlags` annotations are applied if the model contains them.
- After that all the changes done via Simulation Setup window for a certain model are preserved for the whole session. If you want to use the same settings in future sessions then you should store them inside `experiment`, `__OpenModelica_commandLineOptions`, and `__OpenModelica_simulationFlags` annotations.

The OMEdit Simulation Setup 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.

3.15.1 General

- Simulation Interval
- *Start Time* - the simulation start time.
- *Stop Time* - the simulation stop time.
- *Number of Intervals* - the simulation number of intervals.
- *Interval* - the length of one interval (i.e., stepsize)
- Integration
  - *Method* - the simulation solver. See section *Integration Methods* for solver details.
  - *Tolerance* - the simulation tolerance.
  - *Jacobian* - the jacobian method to use.
  - DASSL/IDA Options
  - *Root Finding* - Activates the internal root finding procedure of dassl.
  - *Restart After Event* - Activates the restart of dassl after an event is performed.
  - *Initial Step Size*
  - *Maximum Step Size*
  - *Maximum Integration Order*
- *C/C++ Compiler Flags (Optional)* - the optional C/C++ compiler flags.
- *Number of Processors* - the number of processors used to build the simulation.
- *Build Only* - only builds the class.
- *Launch Transformational Debugger* - launches the transformational debugger.
- *Launch Algorithmic Debugger* - launches the algorithmic debugger.
- *Launch Animation* - launches the 3d animation window.

3.15.2 Interactive Simulation

- Simulate with steps (makes the interactive simulation synchronous; plots nicer curves at the expense of performance)
- Simulation server port

3.15.3 Translation Flags

3.15.4 Simulation Flags

- *Model Setup File (Optional)* - specifies a new setup XML file to the generated simulation code.
- *Initialization Method (Optional)* - specifies the initialization method.
- *Equation System Initialization File (Optional)* - specifies an external file for the initialization of the model.
- *Equation System Initialization Time (Optional)* - specifies a time for the initialization of the model.
- *Clock (Optional)* - the type of clock to use.
- *Linear Solver (Optional)* - specifies the linear solver method.
- *Non Linear Solver (Optional)* - specifies the nonlinear solver.
- *Linearization Time (Optional)* - specifies a time where the linearization of the model should be performed.
- *Output Variables (Optional)* - outputs the variables a, b and c at the end of the simulation to the standard output.

- *CPU Time* - dumps the cpu-time into the result file.
- *Enable All Warnings* - outputs all warnings.
- *Logging (Optional)*
- *LOG_STDOUT* - standard output stream. This stream is always active, can be disabled with -lv=-LOG_STDOUT
- *LOG_ASSERT* - This stream is always active, can be disabled with -lv=-LOG_ASSERT
- *LOG_DASSL* - additional information about dassl solver.
- *LOG_DASSL_STATES* - outputs the states at every dassl call.
- *LOG_DEBUG* - additional debug information.
- *LOG_DELAY* - Debug information for delay operator.
- *LOG_DIVISION* - Log division by zero.
- *LOG_DSS* - outputs information about dynamic state selection.
- *LOG_DSS_JAC* - outputs jacobian of the dynamic state selection.
- *LOG_DT* - additional information about dynamic tearing.
- *LOG_DT_CONS* - additional information about dynamic tearing (local and global constraints).
- *LOG_EVENTS* - additional information during event iteration.
- *LOG_EVENTS_V* - verbose logging of event system.
- *LOG_GBODE* - Information about GBODE solver.
- *LOG_GBODE_V* - Verbose information about GBODE solver.
- *LOG_GBODE_NLS* - Log non-linear solver process of GBODE solver.
- *LOG_GBODE_NLS_V* - Verbose log non-linear solver process of GBODE solver.
- *LOG_GBODE_STATES* - Output states at every GBODE call.
- *LOG_INIT* - additional information during initialization.
- *LOG_INIT_HOMOTOPY* - Log homotopy initialization.
- *LOG_INIT_V* - Verbose information during initialization.
- *LOG_IPOPT* - information from Ipopt.
- *LOG_IPOPT_FULL* - more information from Ipopt.
- *LOG_IPOPT_JAC* - check jacobian matrix with Ipopt.
- *LOG_IPOPT_HESSE* - check hessian matrix with Ipopt.
- *LOG_IPOPT_ERROR* - print max error in the optimization.
- *LOG_JAC* - Outputs the jacobian matrix used by ODE solvers.
- *LOG_LS* - logging for linear systems.
- *LOG_LS_V* - verbose logging of linear systems.
- *LOG_NLS* - logging for nonlinear systems.
- *LOG_NLS_V* - verbose logging of nonlinear systems.
- *LOG_NLS_HOMOTOPY* - logging of homotopy solver for nonlinear systems.
- *LOG_NLS_JAC* - outputs the jacobian of nonlinear systems.
- *LOG_NLS_JAC_TEST* - tests the analytical jacobian of nonlinear systems.

- *LOG_NLS_NEWTON_DIAGNOSTICS* - Log Newton diagnostics. A Diagnostic method to figure out which individual initial guess values are more likely to be causing the convergence failure of Newton-type iterative nonlinear solvers.
- *LOG_NLS_RES* - outputs every evaluation of the residual function.
- *LOG_NLS_EXTRAPOLATE* - outputs debug information about extrapolate process.
- *LOG_RES_INIT* - outputs residuals of the initialization.
- *LOG_RT* - additional information regarding real-time processes.
- *LOG_SIMULATION* - additional information about simulation process.
- *LOG_SOLVER* - additional information about solver process.
- *LOG_SOLVER_V* - verbose information about the integration process.
- *LOG_SOLVER_CONTEXT* - context information during the solver process.
- *LOG_SOTI* - final solution of the initialization.
- *LOG_SPATIALDISTR* - logging of internal operations for spatialDistribution.
- *LOG_STATS* - additional statistics about timer/events/solver.
- *LOG_STATS_V* - additional statistics for LOG_STATS.
- *LOG_SUCCESS* - This stream is always active, can be disabled with -lv=-LOG_SUCCESS.
- *LOG_SYNCHRONOUS* - Log clocks and sub-clocks for synchronous features.
- *LOG_ZEROCROSSINGS* - additional information about the zero-crossings.
- *Additional Simulation Flags (Optional)* - specify any other simulation flag.

3.15.5 Output

- *Output Format* - the simulation result file output format.
- *Single Precision* - Output results in single precision (only for mat output format).
- *File Name Prefix (Optional)* - the name is used as a prefix for the output files.
- *Result File (Optional)* - the simulation result file name.
- *Variable Filter (Optional)* - only output variables with names fully matching the regular expression.
- *Protected Variables if not encrypted* - adds the protected variables in result file.
- *Equidistant Time Grid* - output the internal steps given by dassl instead of interpolating results into an equidistant time grid as given by stepSize or numberOfIntervals.
- *Store Variables at Events* - adds the variables at time events.
- *Show Generated File* - displays the generated files in a dialog box.

The Variable Filter takes a regular expression input and only saves in the simulation results file those variables whose names fully match it. Here are some simple examples:

- `.*` matches any variable (default choice)
- `xy.*` matches variables starting with `xy`
- `.*yz` matches variables ending with `yz`
- `abc\.def.*` matches variables starting with `abc.def`. Note that the `.` character is a regex meta-character, so it must be escaped by a `\`
- `.*body\.a_0\[1\]` matches variables ending with `body.a_0[1]`. Note that `.`, `[`, and `]` must be escaped
- `x\[.*\]` matches all elements of array `x`
- `x\[[2-4]\]` matches elements 2, 3, and 4 of array `x`

- `abc.*|def.*` matches variables starting with `abc` or `def`
- `.*der\(.*\)` matches all derivatives in the model. Note that `(` and `)` must be escaped

Please note that all the model variables will still be shown in the Variables Browser tree; however, only those for which results were actually saved will have a checkbox to plot them.

3.15.6 CSV-File Data Input

When simulating Modelica models with top-level inputs (input variables or input connectors), these inputs are assumed to be equal to their start value by default. However, it is possible to feed them with input signals obtained from CSV (Comma-Separated Value) input data files, by means of the *-csvInput* simulation flag, that can be set in the *Additional Simulation Flags (Optional)* field of the Simulation Flags tab. For example, setting `-csvInput=myinput.csv` causes the runtime executable to read such input data from the `myinput.csv` file.

CSV files should contain the names of the input variables in the first row, beginning with `time` on the first column, and the values of such variables for each point in time in subsequent rows, with non-decreasing time values. The variable names should be enclosed by quotation marks in case they contain spaces, to avoid ambiguities. The default separator for data items within each row is the comma, but it is also possible to use other separators, e.g., space, tab, or semi-colon; in this case, the file should start with the separator specification `"sep=x"` (including the quotation marks), where `x` is the separator character.

For example, assume your model has three top-level inputs named `u1`, `u2`, and `u3`. These are valid CSV input files:

```
time, u3, u2, u1
0.0, 0.0, 0.0, 0.0
1.0, 0.0, 0.0, 0.0
2.0, 0.0, 0.0, 1.0

"sep=;" time; u3; u2; u1
0.0; 0.0; 0.0; 0.0
1.0; 0.0; 0.0; 0.0
2.0; 0.0; 0.0; 1.0

"sep= " "time" "u3" "u2" "u1"
0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
2.0 0.0 0.0 1.0
```

Note that input labels need not be lexicographically ordered, the association between the columns and the inputs is given by the first row.

The CSV-file provides the values of the top level inputs at the specified points in time; linear interpolation is used to provide intermediate values between any two subsequent data points. Discontinuous inputs can be obtained by providing two consecutive rows with the same time value, containing the left limit values and the right limit values.

Unless an absolute pathname is provided for the CSV-files, OMEdit will load it from the sub-directory of the working directory which has the same name of the model, where all the other input and output data files are located.

3.15.7 Data Reconciliation

- *Algorithm* - data reconciliation algorithm.
- *Measurement Input File* - measurement input file.
- *Correlation Matrix Input File* - correlation matrix file.
- *Epsilon*

3.16 2D Plotting

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 3.7. To get several plot windows tiled horizontally or vertically use the menu items *Tile Windows Horizontally* or *Tile Windows Vertically* under *View Menu*.

3.16.1 Types of Plotting

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

Time Plot

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

Plot Parametric

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

Select the x-axis variable while holding down the shift key, release the shift key and then select y-axis variables. One or many y-axis variables can be selected against one x-axis variable. To select a new x-axis variable press and hold the shift key again.

Unchecking the x-axis variable will uncheck all y-axis variables linked to it.

Array Plot

Plots an array variable so that the array elements' indexes are on the x-axis and corresponding elements' values are on the y-axis. The time is controlled by the slider above the variable tree. Right click the result file in variable browser and select *Enable Time Controls* to enable the slider. When an array is present in the model, it has a principal array node in the variable tree. To plot this array as an Array Plot, match the principal node. The principal node may be expanded into particular array elements. To plot a single element in the Time Plot, match the element. A new Array Plot window is opened using the New Array Plot Window toolbar button ( ).

Array Parametric Plot

Plots the first array elements' values on the x-axis versus the second array elements' values on the y-axis. The time is controlled by the slider above the variable tree. Right click the result file in variable browser and select *Enable Time Controls* to enable the slider. To create a new Array Parametric Plot, press the New Array Parametric Plot Window toolbar button ( ), then match the principle array node in the variable tree view to be plotted on the x-axis and match the principle array node to be plotted on the y-axis.

Diagram Window

Shows the active ModelWidget as a read only diagram. You can only have one Diagram Window. To show it click on Diagram Window toolbar button ( ).

3.16.2 Plot Window

A plot window shows the plot curve of instance variables. Several plot curves can be plotted in the same plot window. See Figure 3.7.

Plot Window Menu

- *Auto Scale* - Automatically scales the horizontal and vertical axes.
- *Fit in View* - Adjusts the plot canvas to according to the size of plot curves.
- *Save* - Saves the plot to file system as .png, .svg or .bmp.
- *Print* - Prints the plot.
- *Grid* - Shows grid lines.
- *Detailed Grid* - Shows detailed grid lines.
- *No Grid* - Hides grid lines.
- *Log X* - Logarithmic scale of the horizontal axis.
- *Log Y* - Logarithmic scale of the vertical axis.
- *Setup* - Shows a setup window.
  - *Variables* - List of all plotted variables.
    - *General* - Variable general information.
    - *Legend* - Display name for legend.
    - *File* - File name where variable data is stored.
    - *Appearance* - Visual settings of variable.
    - *Color* - Display color.
    - *Pattern* - Line pattern of curve.
    - *Thickness* - Line thickness of curve.
    - *Hide* - Hide/Show the curve.
    - *Toggle Sign* - Toggles the sign of curve.
    - *Plot on Right Y-Axis* - Display curve on right-side y-axis.
  - *Titles* - Plot, axes and footer titles settings.
  - *Legend* - Sets legend position and font.
  - *Range* - Automatic or manual axes range.
    - *Auto Scale* - Automatically scales the horizontal and vertical axes.
    - *X-Axis*
    - *Minimum* - Minimum value for x-axis.
    - *Maximum* - Maximum value for x-axis.
    - *Y-Axis*
    - *Minimum* - Minimum value for y-axis.
    - *Maximum* - Maximum value for y-axis.
    - *Prefix Units* - Automatically pick the right prefix for units.

3.17 Re-simulating a Model

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

3.18 3D Visualization

Since OpenModelica 1.11 , OMEdit has built-in 3D visualization, which replaces third-party libraries (such as Modelica3D) for 3D visualization.

3.18.1 Running a Visualization

The 3d visualization is based on Qt Quick 3D. In order to run the visualization simply right click the class in Libraries Browser an choose "**Simulate with Animation**" as shown in Figure 3.10.

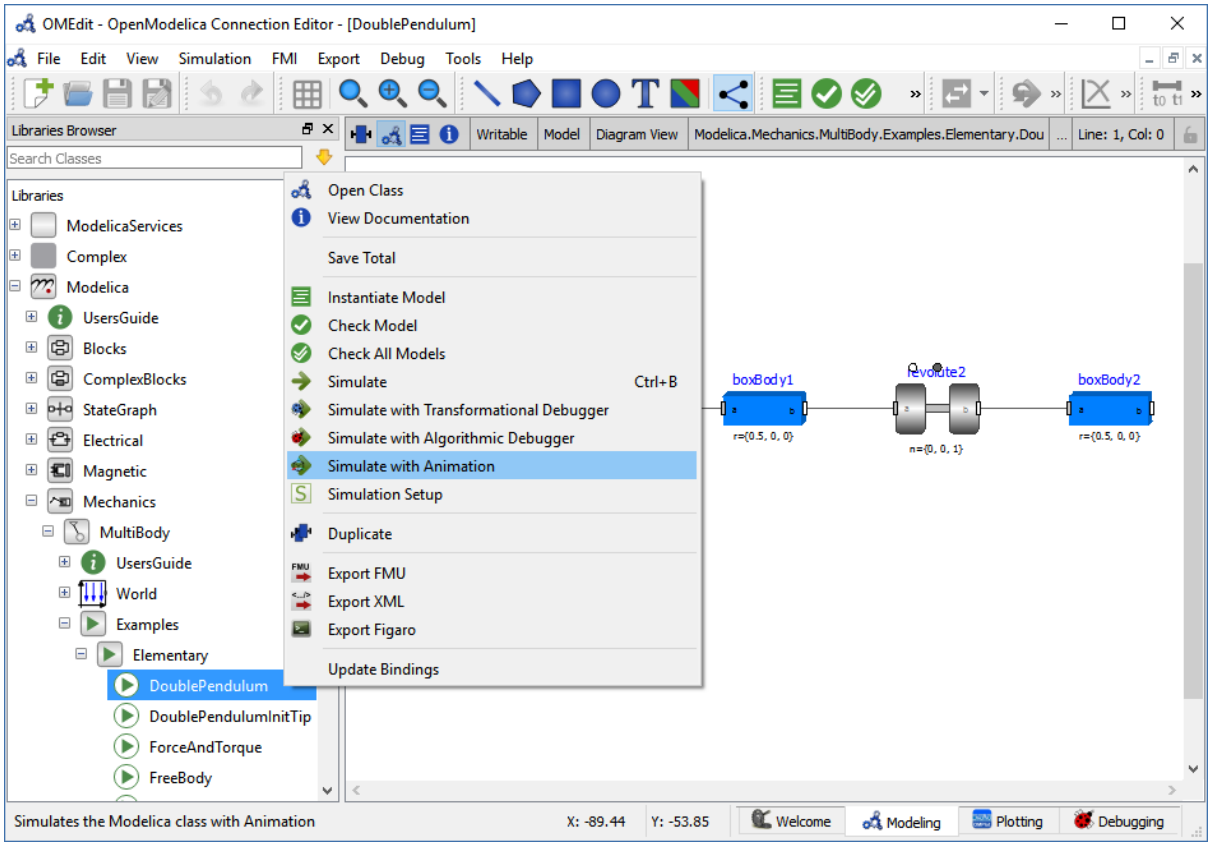


Figure 3.10: OMEdit Simulate with Animation.

One can also run the visualization via Simulation > Simulate with Animation from the menu.

When simulating a model in animation mode, the flag *+d=visxml* is set. Hence, the compiler will generate a scene description file *_visual.xml* which stores all information on the multibody shapes. This scene description references all variables which are needed for the animation of the multibody system. When simulating with *+d=visxml*, the compiler will always generate results for these variables.

3.18.2 Viewing a Visualization

After the successful simulation of the model, the visualization window will show up automatically as shown in Figure 3.11.

The animation starts with pushing the *play* button. The animation is played until stopTime or until the *pause* button is pushed. By pushing the *previous* button, the animation jumps to the initial point of time. Points of time can be selected by moving the *time slider* or by inserting a simulation time in the *Time-box*. The speed factor of animation in relation to realtime can be set in the *Speed-dialog*. Other animations can be opened by using the *open file* button and selecting a result file with a corresponding scene description file.

The 3D camera view can be manipulated as follows:

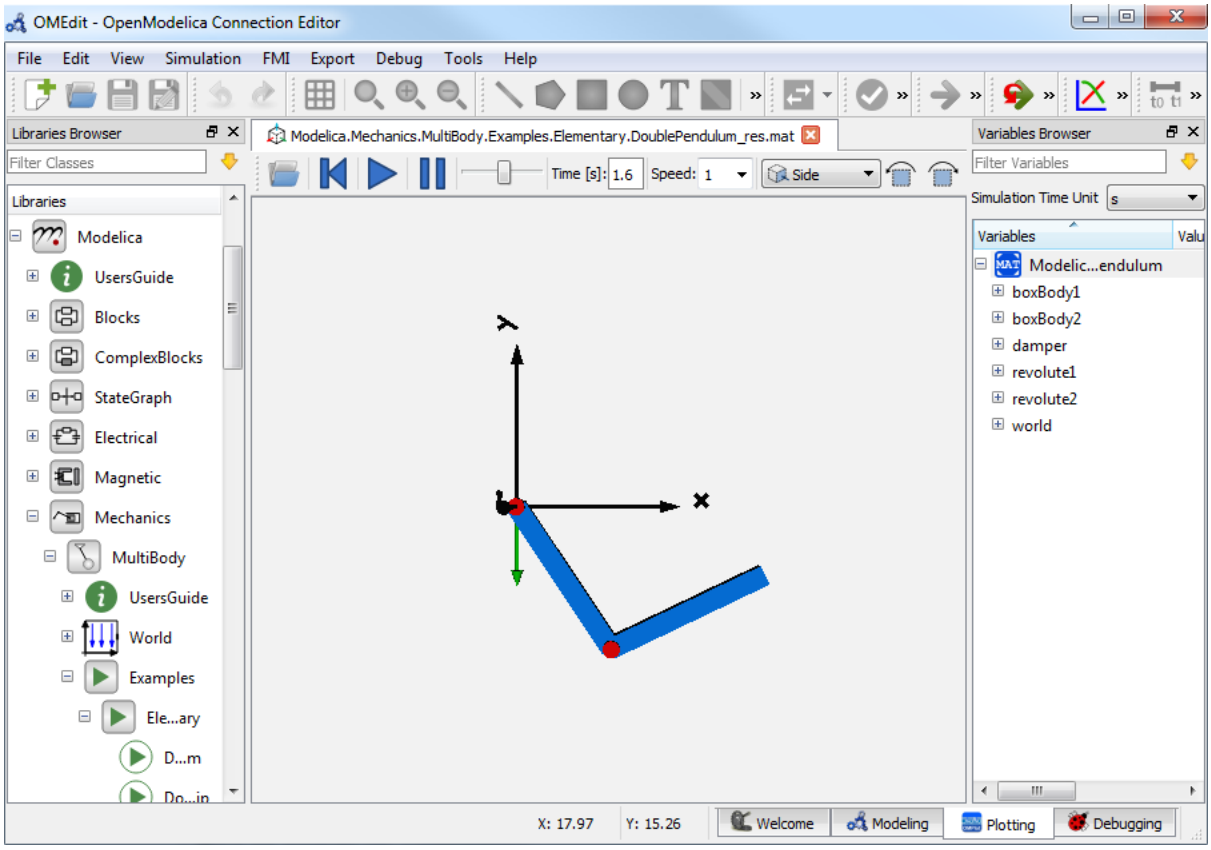


Figure 3.11: OMEdit 3D Visualization.

Operation	Key	Mouse Action
Move Closer/Further	none	Wheel
Move Closer/Further	Right Mouse Hold	Up/Down
Move Up/Down/Left/Right	Middle Mouse Hold	Move Mouse
Move Up/Down/Left/Right	Ctrl + Left Mouse Hold	Move Mouse
Rotate	Left Mouse Hold	Move Mouse
Shape context menu	Right Mouse + Shift	

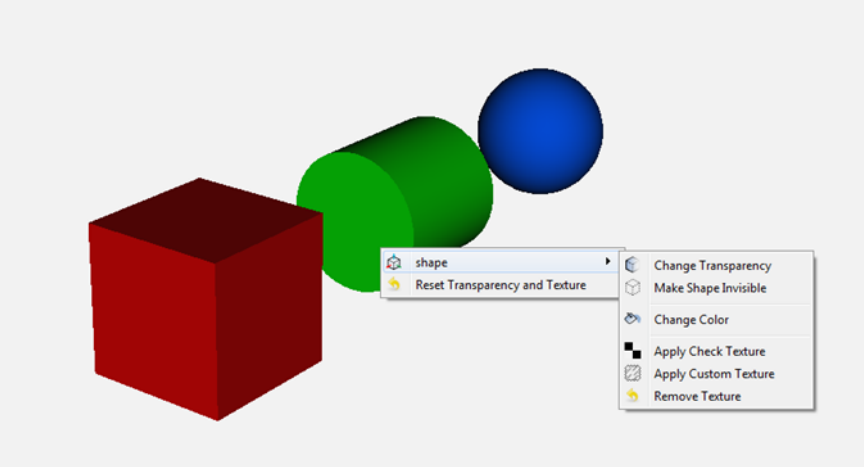
Predefined views (Isometric, Side, Front, Top) can be selected and the camera can be orbited left or right in steps with the rotation buttons.

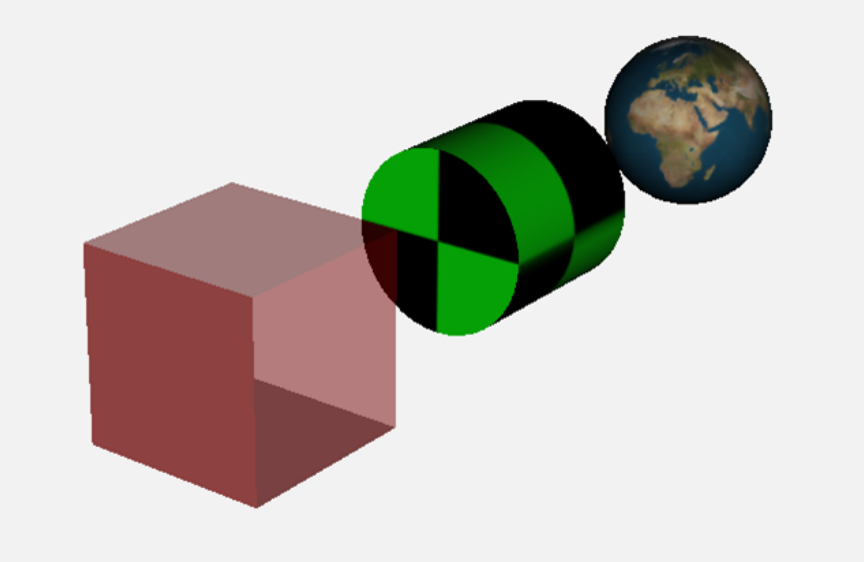
3.18.3 Additional Visualization Features

The shapes that are displayed in the viewer can be selected with shift + right click. If a shape is selected, a context menu pops up that offers additional visualization features

The following features can be selected:

Menu	Description
Change Transparency	The shape becomes either transparent or intransparent.
Make Shape Invisible	The shape becomes invisible.
Change Color	A color dialog pops up and the color of the shape can be set.
Apply Check Texture	A checked texture is applied to the shape.
Apply Custom Texture	A file selection dialog pops up and an image file can be selected as a texture.
Remove Texture	Removes the current texture of the shape.


shape
Reset Transparency and Texture
Change Transparency
Make Shape Invisible
Change Color
Apply Check Texture
Apply Custom Texture
Remove Texture

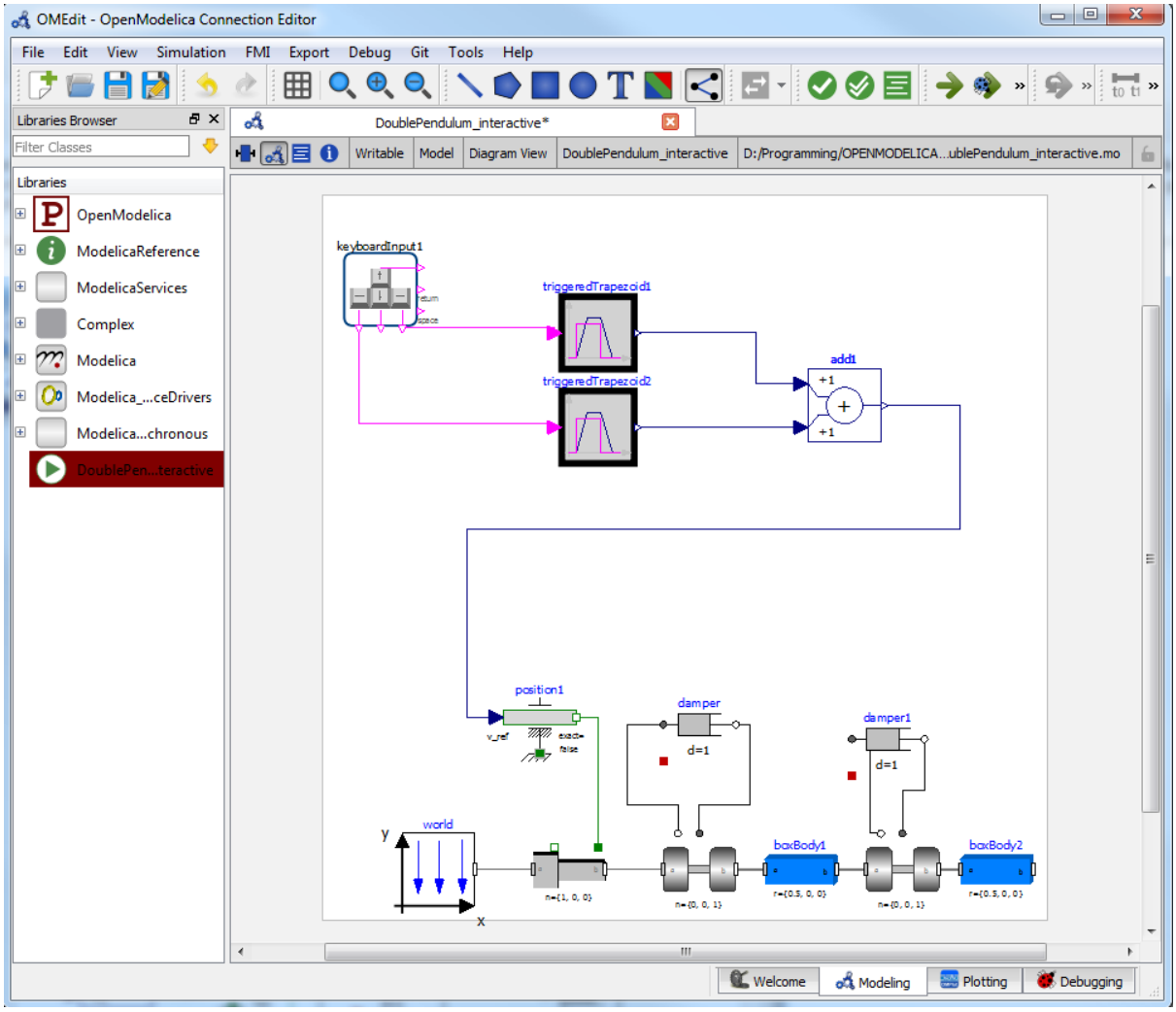


3.19 Animation of Realtime FMUs

Instead of a result file, OMEdit can load Functional Mock-up Units to retrieve the data for the animation of multi-body systems. Just like opening a mat-file from the animation-plotting view, one can open an FMU-file. Necessarily, the FMU has to be generated with the *+d=visxml* flag activated, so that a scene description file is generated in the same directory as the FMU. Currently, only FMU 1.0 and FMU 2.0 model exchange are supported. When choosing an FMU, the simulation settings window pops up to choose solver and step size. Afterwards, the model initializes and can be simulated by pressing the play button.

3.19.1 Interactive Realtime Animation of FMUs

FMUs can be simulated with realtime user interaction. A possible solution is to equip the model with an interaction model from the Modelica_DeviceDrivers library (https://github.com/modelica/Modelica_DeviceDrivers). The realtime synchronization is done by OMEdit so no additional time synchronization model is necessary.



3.20 Interactive Simulation

Warning

Interactive simulation is an experimental feature.

Interactive simulation is enabled by selecting interactive simulation in the simulation setup.

There are two main modes of execution: asynchronous and synchronous (simulate with steps). The difference is that in synchronous (step mode), OMEdit sends a command to the simulation for each step that the simulation should take. The asynchronous mode simply tells the simulation to run and samples variables values in real-time; if the simulation runs very fast, fewer values will be sampled.

When running in asynchronous mode, it is possible to simulate the model in real-time (with a scaling factor just like simulation flag *-rt*, but with the ability to change the scaling factor during the interactive simulation). In the synchronous mode, the speed of the simulation does not directly correspond to real-time.

3.21 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 3.12.

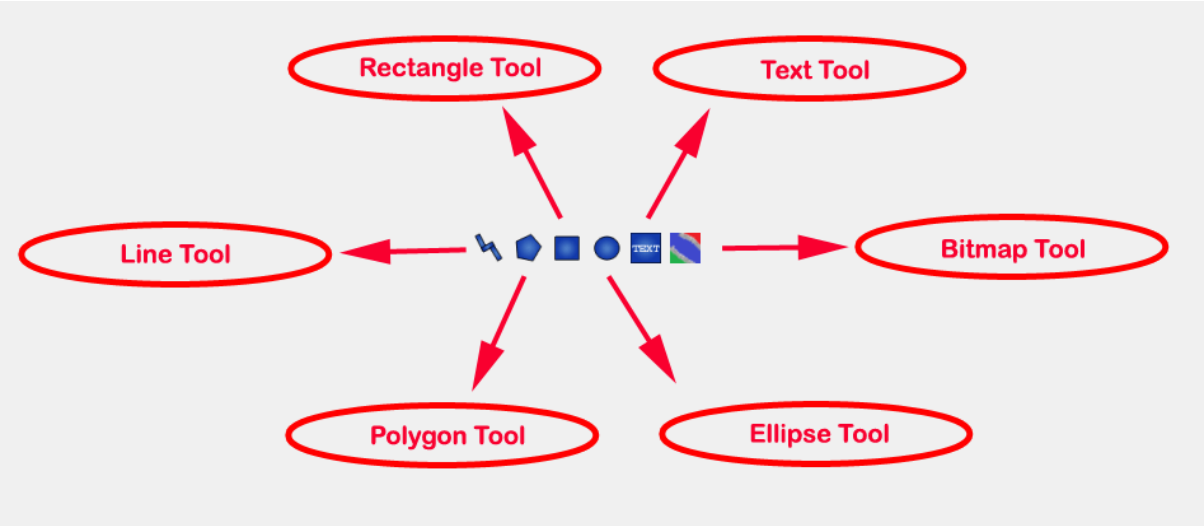


Figure 3.12: User defined shapes.

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.

3.22 Global head section in documentation

If you want to use same styles or same JavaScript for the classes contained inside a package then you can define `__OpenModelica_infoHeader` annotation inside the `Documentation` annotation of a package. For example,

```
package P
  model M
    annotation(Documentation(info="<html>
      <a href=\"javascript:HelloWorld()\">Click here</a>
    </html>"));
  end M;
 annotation(Documentation(__OpenModelica_infoHeader="
     <script type=\"text/javascript\">
       function HelloWorld() {
         alert(\"Hello World!\");
       }
     </script>"));
end P;
```

In the above example model `M` does not need to define the javascript function `HelloWorld`. It is only defined once at the package level using the `__OpenModelica_infoHeader` and then all classes contained in the package can use it.

In addition styles and JavaScript can be added from file locations using Modelica URIs. Example:

```
package P
  model M
    annotation(Documentation(info="<html>
      <a href=\"javascript:HelloWorld()\">Click here</a>
    </html>"));
  end M;
 annotation(Documentation(__OpenModelica_infoHeader="
     <script type=\"text/javascript\">
        src=\"modelica://P/Resources/hello.js\">
       }
     </script>"));
end P;
```

Where the file `Resources/hello.js` then contains:

```
function HelloWorld() {
  alert("Hello World!");
```

(continues on next page)

(continued from previous page)

```
}
```

3.23 Options

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

3.23.1 General Options

- General
- *Language* - Sets the application language.
- *Working Directory* - Sets the application working directory. All files are generated in this directory.
- *Toolbar Icon Size* - Sets the size for toolbar icons.
- *Preserve User's GUI Customizations* - If true then OMEdit will remember its windows and toolbars positions and sizes.
- *Terminal Command* - Sets the terminal command. When user clicks on Tools > Open Terminal then this command is executed.
- *Terminal Command Arguments* - Sets the terminal command arguments.
- *Hide Variables Browser* - Hides the variable browser when switching away from plotting perspective.
- *Activate Access Annotations* - Activates the access annotations for the non-encrypted libraries. Access annotations are always active for encrypted libraries.
- *Create a model.bak-mo backup file when deleting a model*
- *Display errors/warnings when instantiating the graphical annotations* - if true then the errors/warnings are shown when using OMC API for graphical annotations.
- *Enable CRML Support* - Enables the CRML support. The user can create/open .crml files and translate them to Modelica.
- Libraries Browser
- *Library Icon Size* - Sets the size for library icons.
- *Max. Library Icon Text Length to Show* - Sets the maximum text length that can be shown in the icon in Libraries Browser.
- *Show Protected Classes* - If enabled then Libraries Browser will also list the protected classes.
- *Show Hidden Classes if not encrypted* - If enabled then Libraries Browser will also list the hidden classes. Ignores the annotation(Protection(access = Access.hide))
- *Synchronize with Model Widget* - If enabled then Libraries Browser will scroll automatically to the active Model Widget i.e., the current model.
- Enable Auto Save - Enables/disables the auto save feature.
- *Auto Save interval* - Sets the auto save interval value. The minimum possible interval value is 60 seconds.
- Welcome Page
- *Horizontal View/Vertical View* - Sets the view mode for welcome page.
- *Show Latest News* - If enabled then the latest news from https://openmodelica.org are shown.
- *Recent Files and Latest News Size* - Sets the display size for recent files and latest news items.
- Optional Features

- *Disable new instance-based graphical editing of models* - Enables/disables the use of instance-based graphical editing. The instance-based graphical editing enables features like parameter-dependent conditional connectors, conditional dialog enable, replaceable classes and models, etc. It also provides much faster rendering than the previously implemented graphical editing framework. This feature has been thoroughly tested, but it could still have some issues; in case the graphical rendering of models fails (blank screen) or is not correct, you can disable the instance-based editing and fall back to the old editing framework. In that case, please open a ticket on the OpenModelica issue tracker so we can fix the issue for the next release.

3.23.2 Libraries Options

- General
- *OPENMODELICALIBRARY* - The list of paths searched while loading a library, i.e. the OPENMODELICALIBRARY environment variable (MODELICAPATH in the language specification). Paths are separated by `;` on Windows and `:` on Linux and macOS.
- System libraries loaded automatically on startup - The list of system libraries that are loaded on startup.
- *Load latest Modelica version on startup* - Is true then the latest available version of the Modelica Standard Library is always loaded along with its dependencies.
- User libraries loaded automatically on startup - The list of user libraries/files that are loaded on startup.

3.23.3 Text Editor Options

- Format
- *Line Ending* - Sets the file line ending.
- *Byte Order Mark (BOM)* - Sets the file BOM.
- Tabs and Indentation
- *Tab Policy* - Sets the tab policy to either spaces or tabs only.
- *Tab Size* - Sets the tab size.
- *Indent Size* - Sets the indent size.
- Syntax Highlight and Text Wrapping
  - *Enable Syntax Highlighting* - Enable/Disable the syntax highlighting.
  - *Enable Code Folding* - Enable/Disable the code folding. When code folding is enabled multi-line annotations are collapsed into a compact icon (a rectangle containing "...)"). A marker containing a "+" sign becomes available at the left-side of the involved line, allowing the code to be expanded/re-collapsed at will.
  - *Match Parentheses within Comments and Quotes* - Enable/Disable the matching of parentheses within comments and quotes.
  - *Enable Line Wrapping* - Enable/Disable the line wrapping.
- Autocomplete
- *Enable Autocomplete* - Enables/Disables the autocomplete.
- Font
- *Font Family* - Shows the names list of available fonts. Sets the font for the editor.
- *Font Size* - Sets the font size for the editor.

3.23.4 Modelica Editor Options

- *Preserve Text Indentation* - If true then uses *diffModelicaFileListings* API call otherwise uses the OMC pretty-printing.
- Colors

- *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.

3.23.5 Modelica Script Editor Options

- Colors
- *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.

3.23.6 MetaModelica Editor Options

- Colors
- *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.

3.23.7 SSP Editor Options

- Colors
- *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.

3.23.8 CRML Editor Options

- Colors
- *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.

3.23.9 C/C++ Editor Options

- Colors
- *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.

3.23.10 HTML Editor Options

- Colors
- *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.

3.23.11 Graphical Views Options

- General
  - Modeling View Mode
  - *Tabbed View/SubWindow View* - Sets the view mode for modeling.
  - 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.
  - *Move connectors together on both icon and diagram layers*

3.23.12 Simulation Options

- Simulation
  - Translation Flags
  - *Matching Algorithm* - sets the matching algorithm for simulation.
  - *Index Reduction Method* - sets the index reduction method for simulation.
  - *Show additional information from the initialization process* - prints the information from the initialization process
  - *Evaluate all parameters (faster simulation, cannot change them at runtime)* - makes the simulation more efficient but you have to recompile the model if you want to change the parameter instead of re-simulate.
  - *Enable analytical jacobian for non-linear strong components* - enables analytical jacobian for non-linear strong components without user-defined function calls.
  - *Enable parallelization of independent systems of equations (Experimental)*
  - *Enable old frontend for code generation*
  - *Enable FMU Import* - See *FMI Import - SSP*.
  - *Profiling* - creates a profiling HTML file.
  - *Additional Translation Flags* - sets the translation flags see *Options*
  - *Target Language* - sets the target language in which the code is generated.
  - *Target Build* - sets the target build that is used to compile the generated code.
  - *C Compiler* - sets the C compiler for compiling the generated code.
  - *CXX Compiler* - sets the CXX compiler for compiling the generated code.
  - *Use static linking* - if true then static linking is used for simulation executable. The default is dynamic linking. This option is only available on Windows.
  - *Post compilation command* - if not empty allows to run a command after the compilation step. A possible use-case is to be able to sign the binaries before execution to comply with the security policy. The command is run in the same folder where the simulation executable is created. The interpreter executable must be passed to run shell scripts, eg on Windows: *powershell.exe -File C:script.ps1*
  - *Ignore __OpenModelica_commandLineOptions annotation* - if true then ignores the __OpenModelica_commandLineOptions annotation while running the simulation.
  - *Ignore __OpenModelica_simulationFlags annotation* - if true then ignores the __OpenModelica_simulationFlags annotation while running the simulation.
  - *Save class before simulation* - if true then always saves the class before running the simulation.

- *Switch to plotting perspective after simulation* - if true then GUI always switches to plotting perspective after the simulation.
- *Close completed simulation output windows before simulation* - if true then the completed simulation output windows are closed before starting a new simulation.
- *Delete intermediate compilation files* - if true then the files generated during the compilation are deleted automatically.
- *Delete entire simulation directory of the model when OMEdit is closed* - if true then the entire simulation directory is deleted on quit.
- Output
- *Structured* - Shows the simulation output in the form of tree structure.
- *Formatted Text* - Shows the simulation output in the form of formatted text.
- *Display Limit* - Sets the display limit for simulation output. A link to log file is shown once the limit is reached.

3.23.13 Messages Options

- General
- *Output Size* - Specifies the maximum number of rows the Message Browser may have. If there are more rows then the rows are removed from the beginning.
- *Reset messages number before simulation* - Resets the messages counter before starting the simulation.
- *Clear messages browser before checking, instantiation & simulation* - If enabled then the message browser is cleared before checking, instantiation & simulation of model.
- *Do not automatically enlarge message browser when a new message is available* - If enabled then the message browser will not be enlarged instead the tabbar shown will start blinking indicating that a new message is available.
- Font and Colors
- *Font Family* - Sets the font for the messages.
- *Font Size* - Sets the font size for the messages.
- *Notification Color* - Sets the text color for notification messages.
- *Warning Color* - Sets the text color for warning messages.
- *Error Color* - Sets the text color for error messages.

3.23.14 Notifications Options

- 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 partial and component is 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.
  - *Always ask for the dragged component name* - If true then a message will pop-up when user drag & drop the component on the graphical view.

- *Always ask for what to do with the text editor error* - If true then a message will always pop-up when there is an error in the text editor.
- If new frontend for code generation fails
- *Always ask for old frontend*
- *Try with old frontend once*
- *Switch to old frontend permanently*
- *Keep using new frontend*

3.23.15 Line Style Options

- 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.

3.23.16 Fill Style Options

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

3.23.17 Plotting Options

- General
- *Auto Scale* - Sets whether to auto scale the plots or not.
- *Prefix Units* - Automatically pick the right prefix for units for the new plot windows. For existing plot windows use the *Plot Window Menu*.
- Plotting View Mode
- *Tabbed View/SubWindow View* - Sets the view mode for plotting.
- Curve Style
- *Pattern* - Sets the curve pattern.
- *Thickness* - Sets the curve thickness.
- Variable filter
- *Filter Interval* - Delay in filtering the variables. Set the value to 0 if you don't want any delay.
- Font Size - sets the font size for plot window items
- *Title*
- *Vertical Axis Title*
- *Vertical Axis Numbers*
- *Horizontal Axis Title*
- *Horizontal Axis Numbers*

- *Footer*
- *Legend*

3.23.18 Figaro Options

- Figaro
- *Figaro Library* - the Figaro library file path.
- *Tree generation options* - the Figaro tree generation options file path.
- *Figaro Processor* - the Figaro processor location.

3.23.19 CRML Options

- CRML
- *Compiler Jar* - the compiler jar file of CRML.
- *Compiler Arguments* - arugments to the compiler process.
- *Processor* - the CRML translator.
- *Modelica Library Paths* - list of Modelica library paths that are needed to run the translated model.

3.23.20 Debugger Options

- Algorithmic Debugger
- *GDB Path* - the gnu debugger path
- *GDB Command Timeout* - timeout for gdb commands.
- *GDB Output Limit* - limits the GDB output to N characters.
- *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.

3.23.21 FMI Options

- Export
  - Version
    - *1.0* - Sets the FMI export version to 1.0.

> **Warning**
>
> Deprecated and will be removed in a future version of OpenModelica

    - *2.0* - Sets the FMI export version to 2.0.x
    - *3.0* - Sets the FMI export version to 3.0.x

> **Warning**
>
> Experimental support

- Type
- *Model Exchange* - Sets the FMI export type to Model Exchange (ME).
- *Co-Simulation* - Sets the FMI export type to Co-Simulation (CS).
- *Scheduled Execution* - Sets the FMI export type to Scheduled Execution (SE), needs at least FMI version 3.0.
- *Model Exchange and Co-Simulation* - Sets the FMI export type to Model Exchange and Co-Simulation.
- *FMU Name* - Sets a prefix for generated FMU file.
- *Move FMU* - Moves the generated FMU to a specified path.
- Platforms: See *Platforms*.
- Solver for Co-Simulation
- *Explicit Euler*
- *CVODE*
- *Model Description Filters* - Sets the variable filter for model description file, see *--fmifilter*.
- *Include Modelica based resources via loadResource*
- *Include Source Code* - Sets if the exported FMU can contain source code. Model Description Filter "blackBox" will override this, because black box FMUs do never contain their source code.
- *Generate Debug Symbols* - Generates a FMU with debug symbols.

- Import
- *Delete FMU directory and generated model when OMEdit is closed* - If true then the temporary FMU directory that is created for importing the FMU will be deleted.

3.23.22 OMSimulator/SSP Options

- General
- *Command Line Options* - sets the OMSimulator command line options.
- *Logging Level* - OMSimulator logging level.

3.23.23 Sensitivity Optimization Options

- General
- *OMSens Backend Path* - sets the OMSens backend.
- *python* - sets the Python executable to run OMSens scripts.

3.23.24 Language Server Options

OMEdit can connect to an external Modelica language server that speaks the Language Server Protocol (LSP) over stdin/stdout. When enabled, the language server provides:

- **Hover** - hover the mouse over a symbol in the text editor to see its documentation in a tooltip.
- **Go to definition** - Ctrl+Click a symbol, or use *Go to Definition* from the right-click menu, to navigate to where it is defined (including across files). When the language server cannot resolve the symbol, OMEdit falls back to its built-in class navigation.

The feature is on by default and can be turned off here.

- Language Server Protocol (LSP)
  - *Language Server Protocol (LSP)* - when this group is checked, OMEdit starts the language server process and connects to it.
  - *Server Executable* - path to the language server executable. Leave it blank (recommended) to run the server installed with OpenModelica. Use *Download...* below to fetch a different release, *Browse* to select a server you already have, or *Auto Detect* to search for one.
  - *Restart Server* - stops the language server and starts it again. The library list is kept in step with OMC automatically, so this is only needed if the server stopped after repeated crashes, or if a library changed on disk. It acts immediately; changes to the fields above are applied when you click *OK*.
  - *Download...* - fetches a standalone language server and points *Server Executable* at it. The drop-down next to the button chooses the release: the version marked *(recommended)* is the one installed with OMEdit, and *Latest release* takes the newest release published on GitHub.

  The libraries the server searches are the ones loaded in OMEdit; there is no separate list to maintain. Loading a library makes it resolvable for *Go to Definition* and hover, and unloading it removes it again.
  - *Log language server messages to the Messages Browser* - when checked, messages from the language server are shown in the Messages Browser, prefixed with `LSP`.

Hover information is shown as a tooltip. Navigation works both with Ctrl+Click and from the editor right-click menu (*Go to Definition*, marked with the language server icon while the server is running). Enabling or disabling the language server takes effect immediately; a restart is not required.

The language server installed with OpenModelica

OpenModelica installs a Modelica language server alongside OMEdit, and it is what runs when *Server Executable* is left blank - which is the default, so there is nothing to set up. It brings its own runtime, so nothing else has to be installed.

It is installed in `share/omedit/ls/modelica` as three files: the server itself and the two `.wasm` files it parses Modelica with. They belong together - a server without them starts, and then answers nothing.

Using a different server

Download... on the options page fetches another release of the same standalone server and points *Server Executable* at it. It is installed under your user configuration directory, beside `omedit.ini`, so no administrator rights are needed.

You can also point *Server Executable* at a server you built yourself - which is how to try a change to the language server before it is released. Keep `tree-sitter-modelica.wasm` and `web-tree-sitter.wasm` in the same directory as the binary; without them the server starts but reports nothing, and OMEdit says so in the Messages Browser.

If the server stops

If the server process exits unexpectedly, OMEdit restarts it automatically after a short delay, up to 5 times within a 3 minute window. Past that it stops retrying and reports the failure as an error message. Every crash and restart attempt is appended to `languageserver_crash.log` in OMEdit's temporary directory (*Tools* > *Open Temporary Directory*), so the log can be attached to a bug report.

3.24 __OpenModelica_commandLineOptions Annotation

OpenModelica specific annotation to define the command line options needed to simulate the model. For example if you always want to simulate the model with a specific matching algorithm and index reduction method instead of the default ones then you can write the following code,

```
model Test
  annotation(__OpenModelica_commandLineOptions = "--matchingAlgorithm=BFSB --
↪indexReductionMethod=dynamicStateSelection");
end Test;
```

The annotation is a space separated list of options where each option is either just a command line flag or a flag with a value.

In OMEdit open the Simulation Setup and set the Translation Flags then in the bottom check *Save translation flags inside model i.e., __OpenModelica_commandLineOptions annotation* and click on OK.

If you want to ignore this annotation then use *setCommandLineOptions("--ignoreCommandLineOptionsAnnotation=true")*. In OMEdit *Tools* > *Options* > *Simulation* check *Ignore __OpenModelica_commandLineOptions annotation*.

3.25 __OpenModelica_simulationFlags Annotation

OpenModelica specific annotation to define the simulation options needed to simulate the model. For example if you always want to simulate the model with a specific solver instead of the default DASSL and would also like to see the cpu time then you can write the following code,

```
model Test
  annotation(__OpenModelica_simulationFlags(s = "ida", cpu = "()"));
end Test;
```

The annotation is a comma separated list of options where each option is a simulation flag with a value. For flags that doesn't have any value use *()* (See the above code example).

In OMEdit open the Simulation Setup and set the Simulation Flags then in the bottom check *Save simulation flags inside model i.e., __OpenModelica_simulationFlags annotation* and click on OK.

If you want to ignore this annotation then use *setCommandLineOptions("--ignoreSimulationFlagsAnnotation=true")*. In OMEdit *Tools* > *Options* > *Simulation* check *Ignore __OpenModelica_simulationFlags annotation*.

3.26 Global and Local Flags

There is a large number of optional settings and flags to influence the way OpenModelica generates the simulation code (*Compiler flags*, a.k.a. Translation flags or Command Line Options) and the way the simulation executable is run (*Simulation Flags*).

The global default settings can be accessed and changed with the *Tools* > *Options* menu. It is also possible to reset them to factory state by clicking on the `Reset` button of the *Tools* > *Options* dialog window.

When you start OMEdit and you simulate a model for the first time, the model-specific simulation session settings are initialized by copying the global default settings, and then by applying any further settings that are saved in the model within OpenModelica-specific `__OpenModelica_commandLineOptions` and `__OpenModelica_simulationFlags` annotations. Note that the latter may partially override the former, if they give different values to the same flags.

You can change those model-specific settings at will with the Simulation Setup window. Any change you make will be remembered until the end of the simulation session, i.e. until you close OMEdit. This is very useful to experiment with different settings and find the optimal ones, or to investigate bugs by turning on logging options, etc. If you check the `Save translation flags` and `Save simulation flags` options in the simulation setup, those settings will be saved in the model within the corresponding OpenModelica-specific annotations, so that you can get the same behavior when you start a new session later on, or if someone else loads the model on a different computer. Otherwise, all of those changes will be forgotten when you exit OMEdit.

If you change the global default settings after running some models, the simulation settings of those models will be reset as if you closed OMEdit and restarted a new session: the new global options will first be applied, and then

any further setting saved in the OpenModelica-specific annotations will be applied, possibly overriding the global options if the same flags get different values from the annotations. Any model-specific settings that you may have changed with Simulation Setup up to that point will be lost, unless you saved them in the OpenModelica-specific annotations before changing the global default settings.

3.27 Debugger

For debugging capability, see *Debugging*.

3.28 Editing Modelica Standard Library

By default OMEdit loads the Modelica Standard Library (MSL) as a system library. System libraries are read-only. If you want to edit MSL you need to load it as user library instead of system library. We don't recommend editing MSL but if you really need to and understand the consequences then follow these steps,

- Go to *Tools* > *Options* > *Libraries*.
- Remove Modelica & ModelicaReference from list of system libraries.
- Uncheck *force loading of Modelica Standard Library*.
- Add *$OPENMODELICAHOME/lib/omlibrary/Modelica X.X/package.mo* under user libraries.
- Restart OMEdit.

3.29 Install Library

A new library can be installed with the help of the *package manager*. Click *File->Manage Libraries->Install Library* to open the install library dialog. OMEdit lists the libraries that are available for installation through the package manager.

3.30 Convert Libraries using Conversion Scripts

In order to convert the libraries right-click the model/package in the *Libraries Browser* and choose *Convert to newer versions of used libraries*. OMEdit will read the used libraries from the uses-annotation and list any new version of the library that provide the conversion using the conversion script.

3.31 State Machines

3.31.1 Creating a New Modelica State Class

Follow the same steps as defined in *Creating a New Modelica Class*. Additionally make sure you check the *State* checkbox.

3.31.2 Making Transitions

In order to make a transition from one state to another the user first needs to enable the transition mode ( ) from the toolbar.

Move the mouse over the state. The mouse cursor will change from arrow cursor to cross cursor. To start the transition press left button and move while keeping the button pressed. Now release the left button. Move towards the end state and click when cursor changes to cross cursor.

A *Create Transition* dialog box will appear which allows you to set the transition attributes. Cancelling the dialog will cancel the transition.

Double click the transition or right click and choose *Edit Transition* to modify the transition attributes.

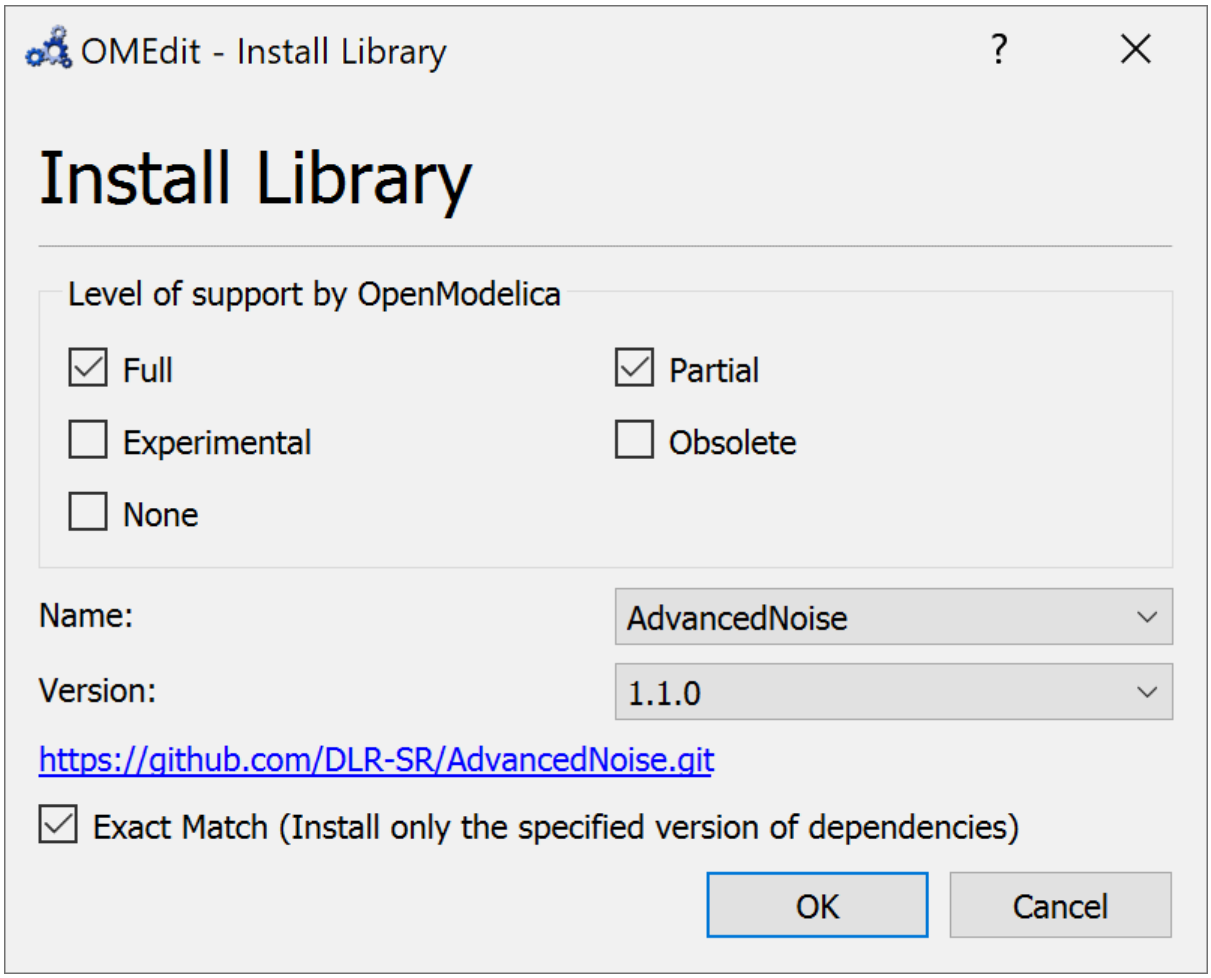


Figure 3.13: Install Library.

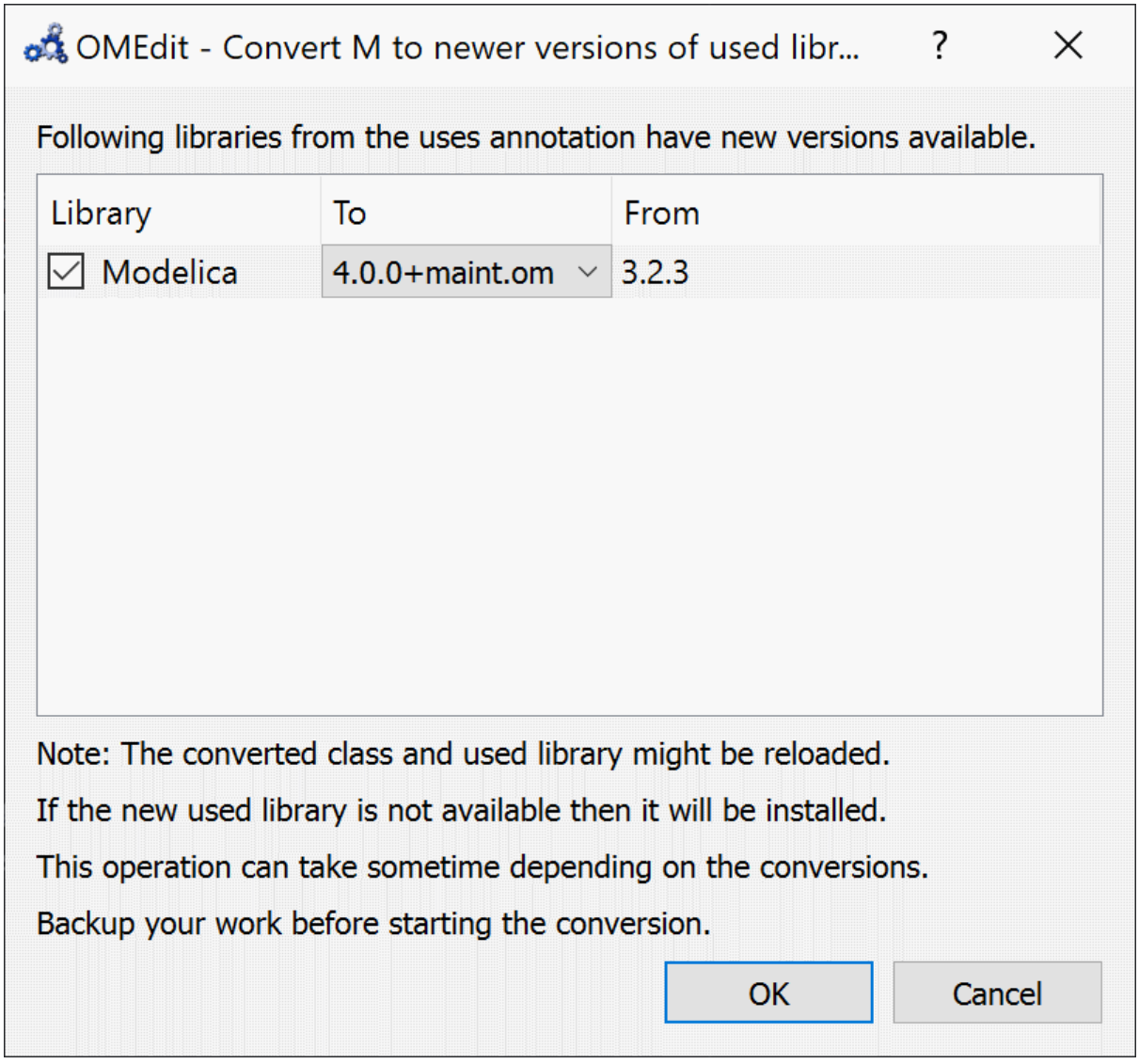


Figure 3.14: Converts the model/package to newer version of used libraries.

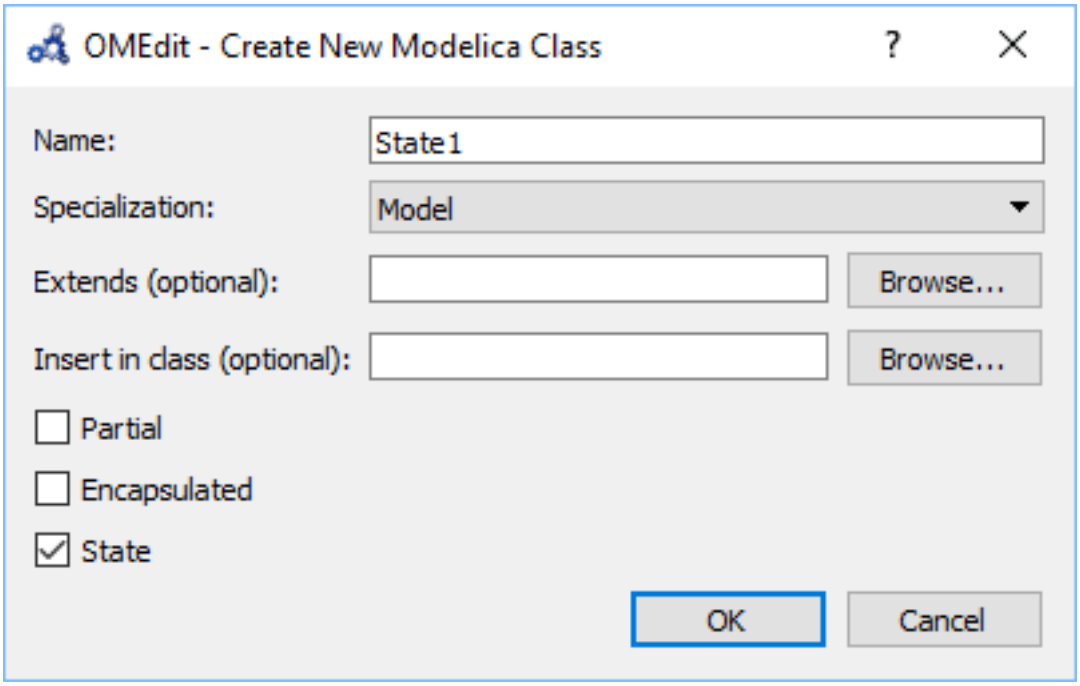


Figure 3.15: Creating a new Modelica state.

3.31.3 State Machines Simulation

Support for Modelica state machines was added in the Modelica Language Specification v3.3. A subtle problem can occur if Modelica v3.2 libraries are loaded, e.g., the Modelica Standard Library v3.2.2, because in this case OMC automatically switches into Modelica v3.2 compatibility mode. Trying to simulate a state machine in Modelica v3.2 compatibility mode results in an error. It is possible to use the OMC flag *--std=latest* in order to ensure (at least) Modelica v3.3 support. In OMEdit this can be achieved by setting that flag in the *Tools* > *Options* > *Simulation* dialog.

3.31.4 State Machines Debugger

Modelica state machines debugger is implemented as a visualization, which allows the user to run the state machines simulation as an animation.

A special Diagram Window is developed to visualize the active and inactive states. The active and inactive value of the states are stored in the OpenModelica simulation result file. After the successful simulation, of the state machine model, OMEdit reads the start, stop time values, and initializes the visualization controls accordingly.

The controls allows the easy manipulation of the visualization,

- Rewind - resets the visualization to start.
- Play - starts the visualization.
- Pause - pauses the visualization.
- Time - allows the user to jump at any specific time.
- Speed - speed of the visualization.
- Slider - controls the time.

The visualization is based on the simulation result file. All three formats of the simulation result file are supported i.e., mat, csv and plt where mat is a matlab file format, csv is a comma separated file and plt is an ordered text file.

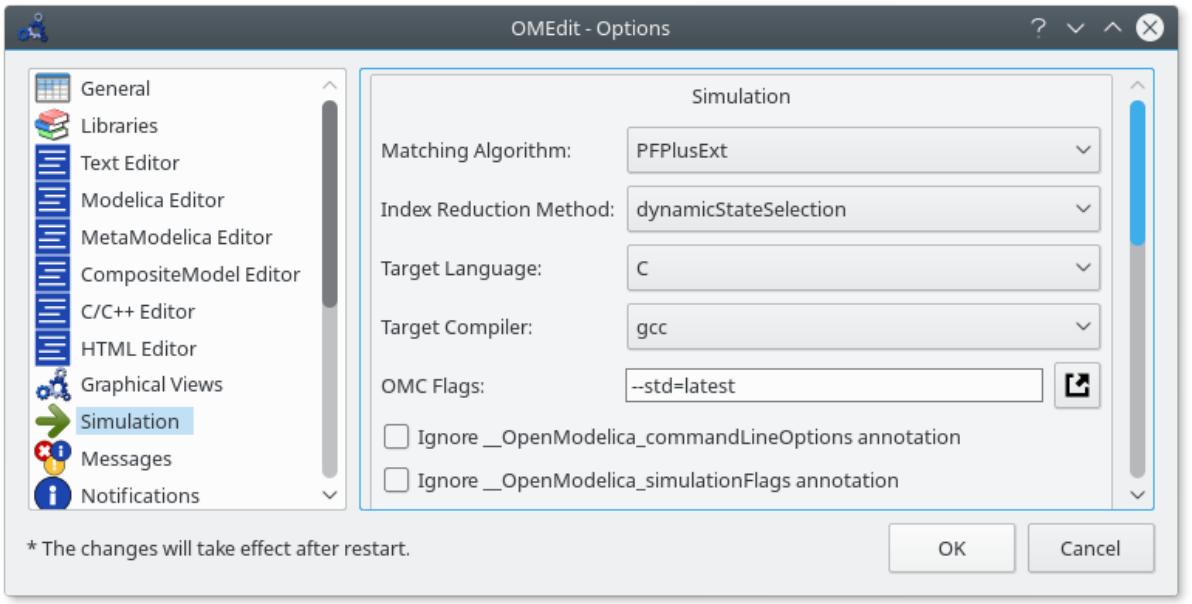


Figure 3.16: Ensure (at least) Modelica v3.3 support.

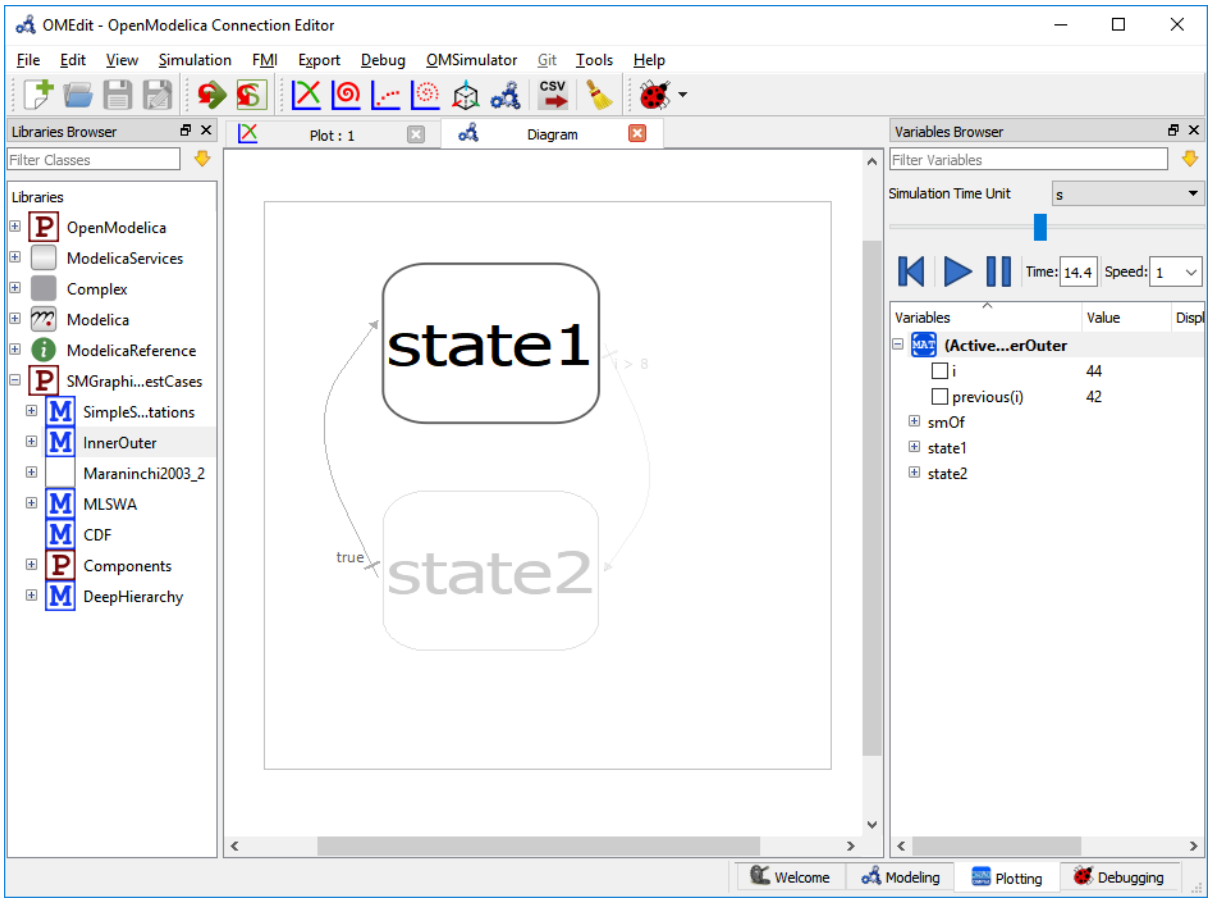


Figure 3.17: State machine debugger in OMEdit.

3.32 Using OMEdit as Text Editor

OMEdit can be be used as a Text editor. Currently support for editing MetaModelica,Modelica and C/C++ are available with syntax highlighting and autocompletion of keywords and types. Additionaly the Modelica and MetaModelica files are provided with autocompletion of code-snippets along with keywords and types. The users can load the directory from file menu *File > Open Directory*. which opens the Directory structure in the Libraries-browser.

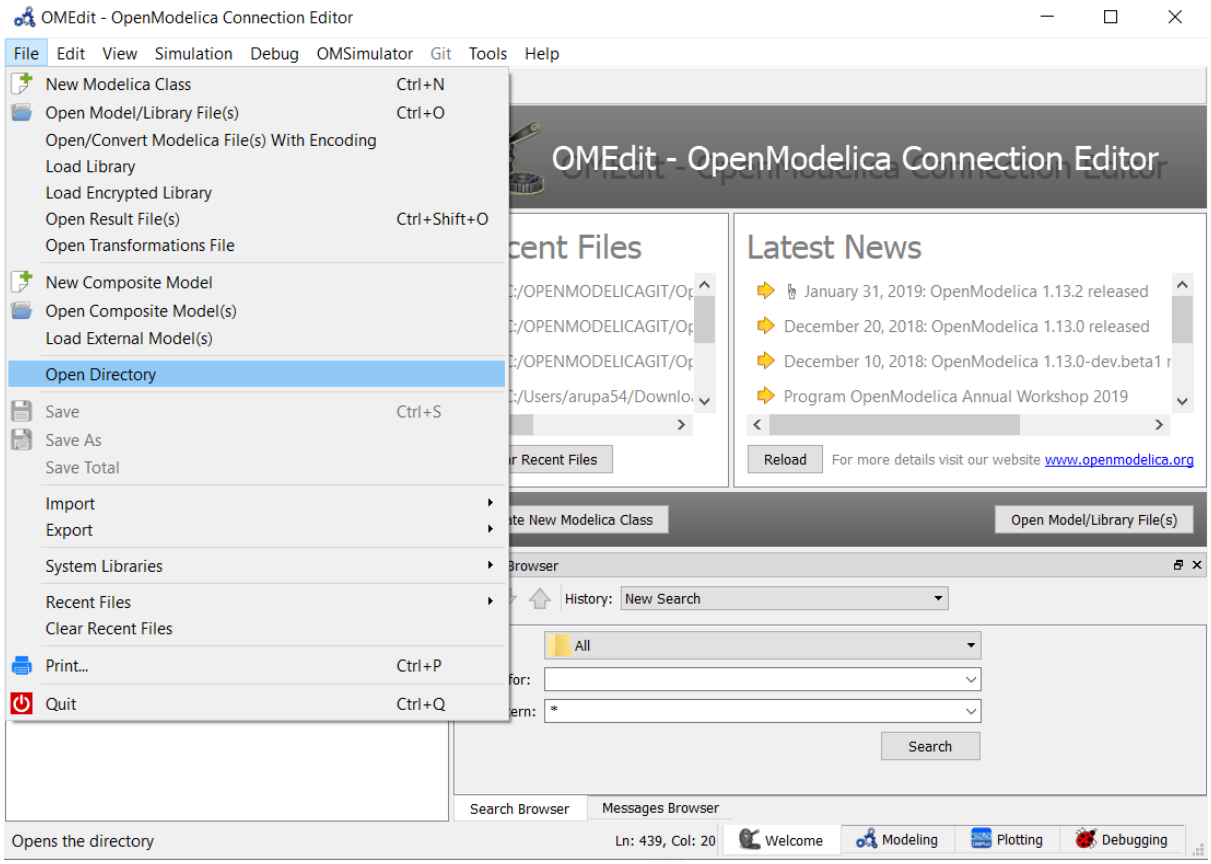


Figure 3.18: open-directory

After the directory is opened in the Libraries-browser, the users can expand the directory structure and click the file which opens in the texteditor.

3.32.1 Advanced Search

Support to search in OMEdit texteditor is available. The search browser can be enabled by selecting View > Windows > Search browser or through shortcut keys (ctrl+h).

The users can start the search by loading the directory they want to search and fill in the text to be searched for and file pattern if needed and click the search button.

After the search is completed the results are presented to the users in a separate window, The search results contains the following

1) The name of the files where the searched word is matched

2) The line number and text of the matched word.

The users can click the line number or the matched text and it will automatically open the file in the texteditor and move the cursor to matched line number of the text.

The users can perform multiple searches and go back to old search results using search histroy option.

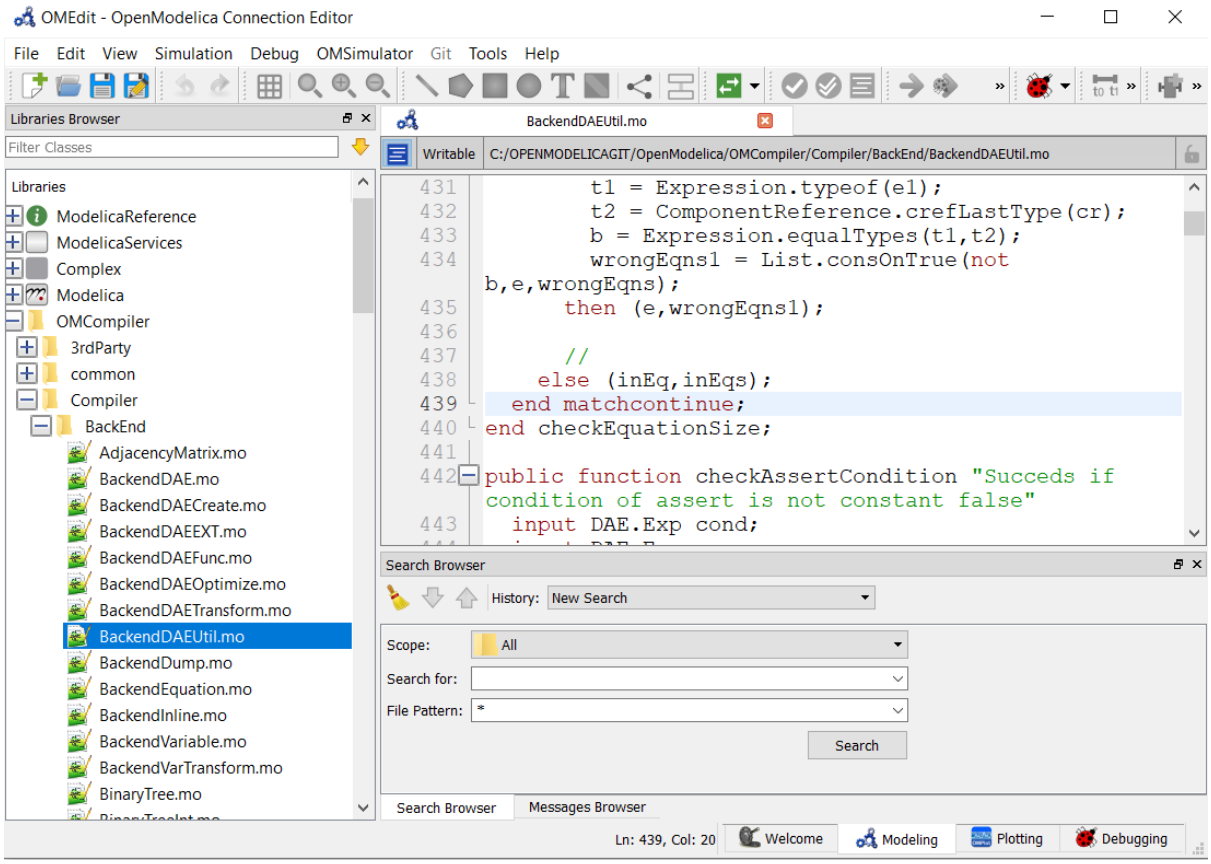


Figure 3.19: openfile in texteditor

3.33 Temporary Directory, Log Files and Working Directory

On Unix/Linux systems temporary directory is the path in the *TMPDIR* environment variable or */tmp* if *TMPDIR* is not defined appended with directory paths *OpenModelica<USERNAME>/OMEdit* so the complete path is usually */tmp/OpenModelica<USERNAME>/OMEdit*.

On Windows its the path in the *TEMP* or *TMP* environment variable appended with directory paths *OpenModelica/OMEdit* so the complete path is usually *%TEMP%/OpenModelica/OMEdit*.

All the log files are always generated in the temporary directory. Choose *Tools* > *Open Temporary Directory* to open the temporary directory.

By default the working directory has the same path as the temporary directory. You can change the working directory from *Tools* > *Options* > *General* see section *General Options*.

For each simulation a new directory with the model name is created in the working directory and then all the simulation intermediate and results files are generated in it.

3.34 High DPI Settings

When the text is too big / too small to read there are options to change the font size used in OMEdit, see *Text Editor Options*.

If you are using a high-resolution screen (1080p, 4k and more) and the app is blurry or the overall proportions of the different windows are off, it can help to change the DPI settings.

On Windows it is possible to change the scaling factor to adjust the size of text, apps and other times, but the default setting might not be appropriate for OMEdit e.g., on compact notebooks with high resolution screens.

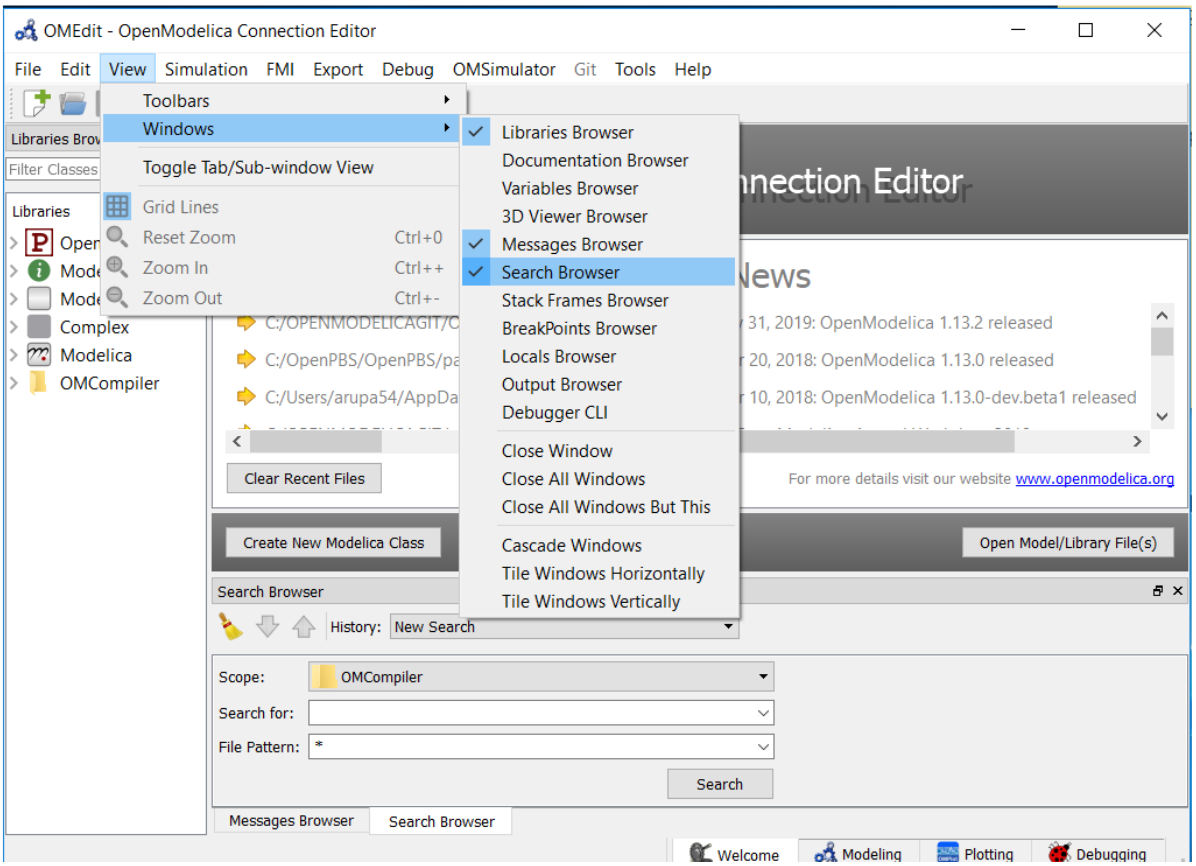


Figure 3.20: Enable omedit search browser

You can either change the scaling factor for the whole Windows system or only change the scaling used for OMEdit. This is done by changing the *Compatibility* settings for *High DPI settings for OMEdit.exe* with the following steps:

1. Press *Windows-Key* and type *OpenModelica Connection Editor* and right-click on the app and *Open file location*, Figure 3.24.
2. Right-click on *OpenModelica Connection Editor* and open *Properties*.
3. In the properties window go to tab *Compatibility* and open *Change high DPI settings*. In the *High DPI settings for OMEdit.exe* choose *Use the settings to fix scaling problems for this program instead of the one in Settings* and *Override high DPI scaling behavior.Scaling performed by:* and choose *System* from the drop-down menu, Figure 3.25.

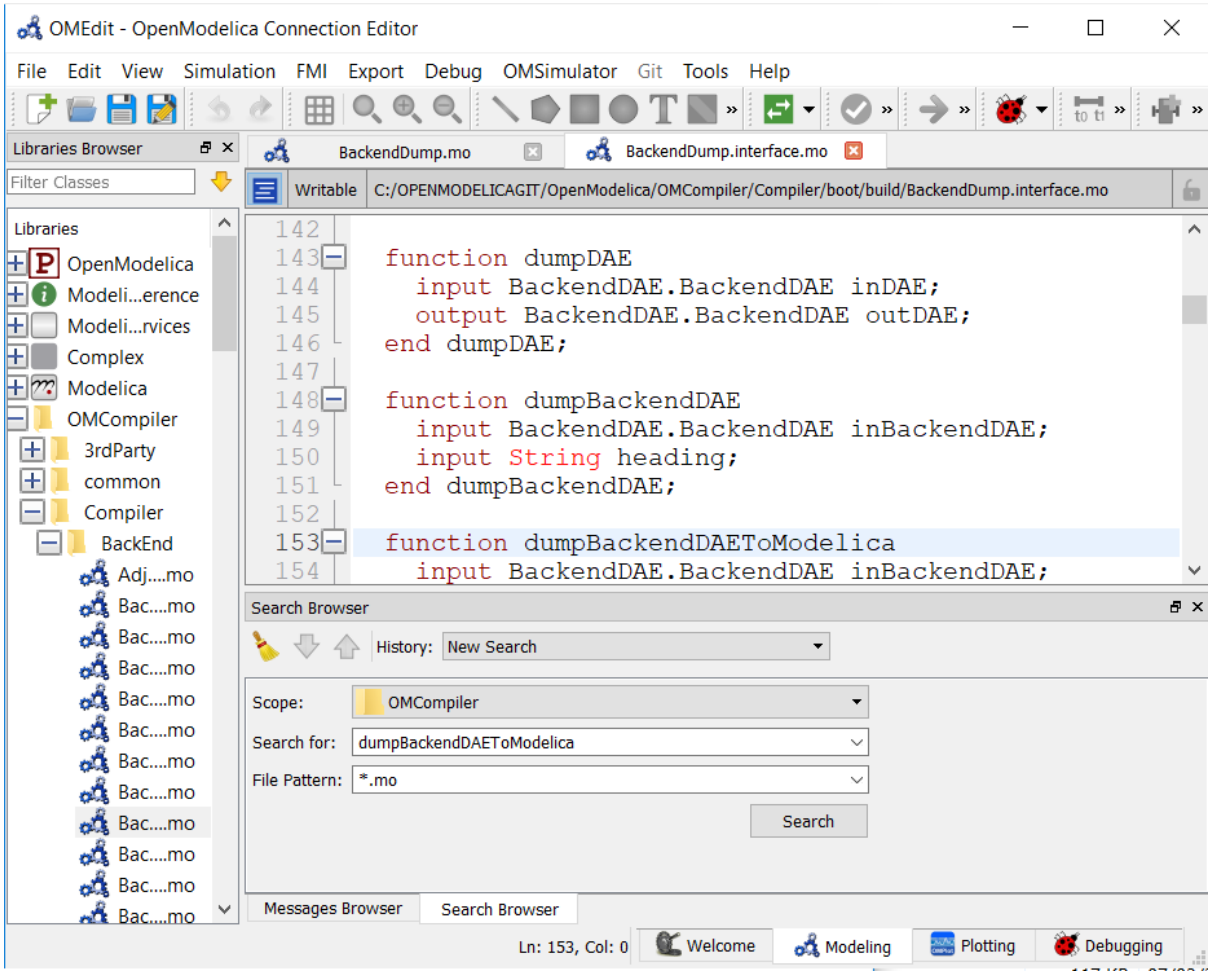


Figure 3.21: Start search in search browser

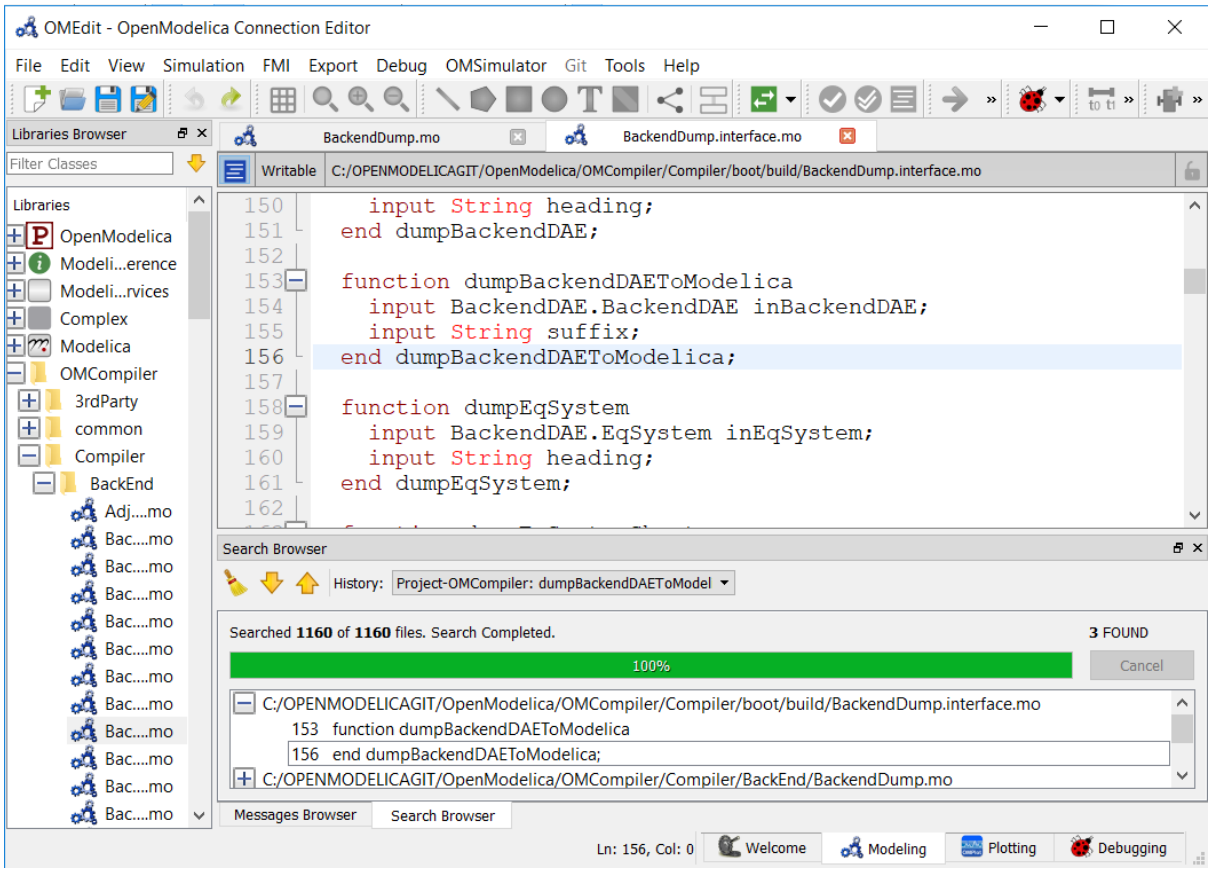


Figure 3.22: Search Results

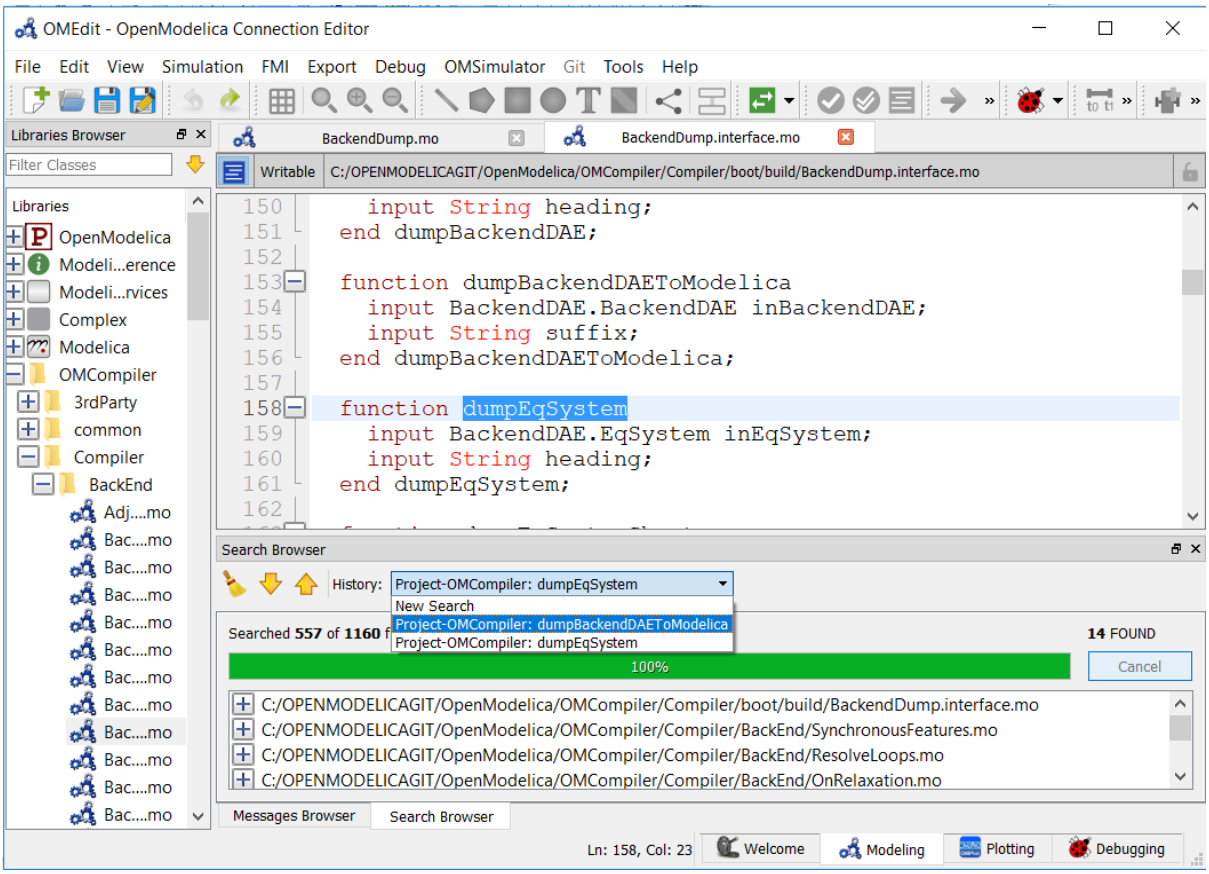


Figure 3.23: Search History

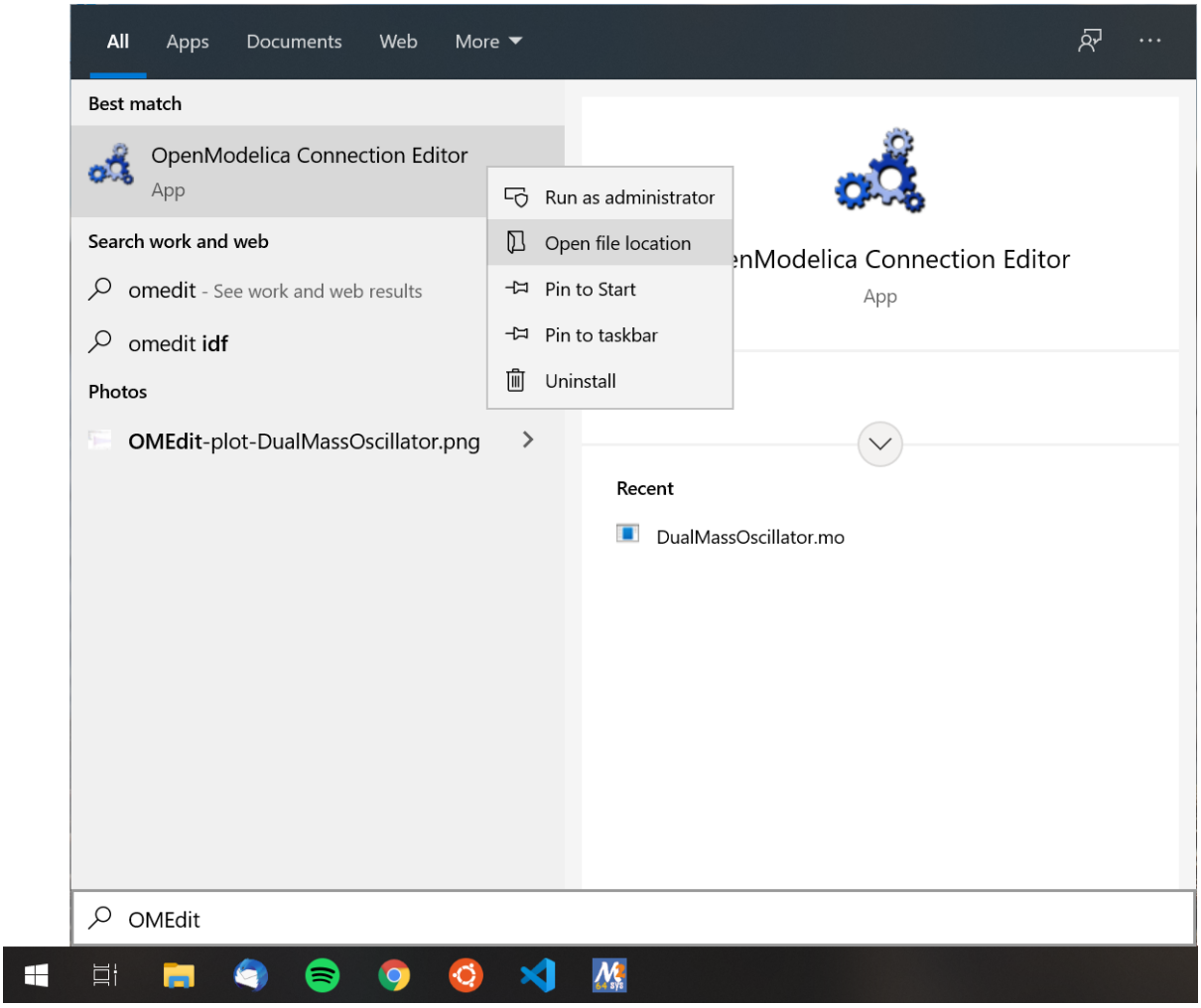


Figure 3.24: Open file location of OpenModelica Connection Editor

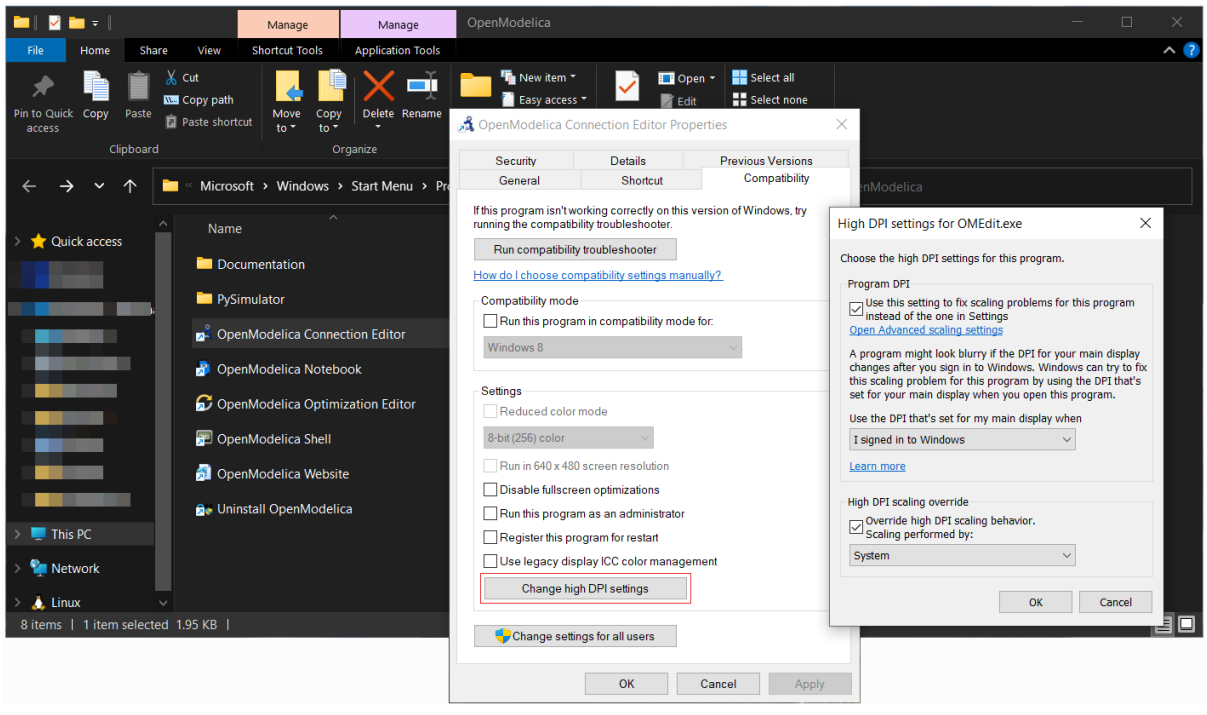


Figure 3.25: Change high DPI settings for OMEdit.exe

CHAPTER

FOUR

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. See also OMEdit *2D Plotting*.

4.1 Example

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

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

```
>>> simulate(HelloWorld, outputFormat="csv", startTime=0, stopTime=4,␣
↪numberOfIntervals=5)
record SimulationResult
    resultFile = "«DOCHOME»/HelloWorld_res.csv",
    simulationOptions = "startTime = 0.0, stopTime = 4.0, numberOfIntervals = 5,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'HelloWorld', options = '',␣
↪outputFormat = 'csv', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0015104390000000001,
    timeBackend = 0.003878331,
    timeSimCode = 9.618080000000001e-4,
    timeTemplates = 0.00198984,
    timeCompile = 0.15790330400000002,
    timeSimulation = 0.0058789210000000005,
    timeTotal = 0.17220248200000002
end SimulationResult;
```

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

Listing 4.1: HelloWorld_res.csv

```
"time","x","der(x)"
0,1,-1
0.8,0.4493289092712475,-0.4493289092712475
1.6,0.2018973974273906,-0.2018973974273906
```

(continues on next page)

(continued from previous page)

```
2.4,0.09071896372718975,-0.09071896372718975
3.2,0.04076293845066793,-0.04076293845066793
4,0.01831609502171534,-0.01831609502171534
4,0.01831609502171534,-0.01831609502171534
```

Use *plot(x)* to plot the diagram using OMPlot.

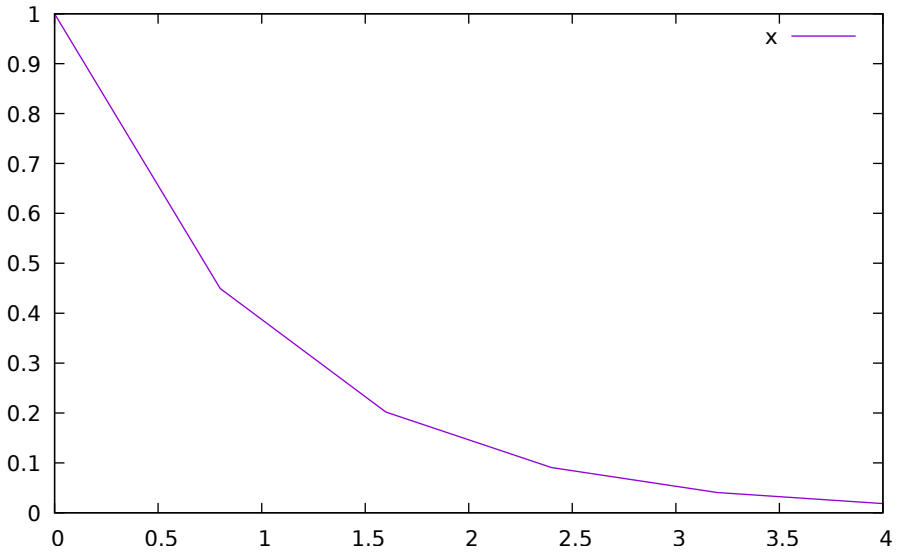


Figure 4.1: Simple 2D plot of the HelloWorld example.

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

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

4.2 Plot Command Interface

Plot command have a number of optional arguments to further customize the the resulting diagram.

```
>>> list(OpenModelica.Scripting.plot,interfaceOnly=true)
"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 = \"\" \"This text will be used as the diagram title.\";
  input String grid = \"simple\" \"Sets the grid for the plot i.e simple, detailed,␣
↪none.\";
  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.\";
```

(continues on next page)

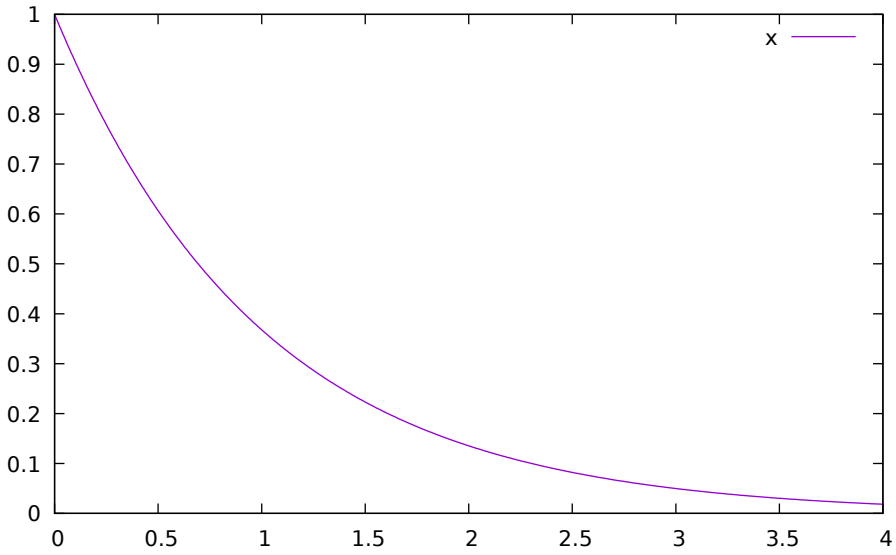


Figure 4.2: Simple 2D plot of the HelloWorld example with a larger number of output points.

(continued from previous page)

```
  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 left 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 left vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.\";
  input Real curveWidth = 1.0 \"Sets the width of the curve.\";
  input Integer curveStyle = 1 \"Sets the style of the curve. SolidLine=1, DashLine=2,
↪ DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.\";
  input String legendPosition = \"top\" \"Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.\";
  input String footer = \"\" \"This text will be used as the diagram footer.\";
  input Boolean autoScale = true \"Use auto scale while plotting.\";
  input Boolean forceOMPlot = false \"if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.\";
  input String yAxis = \"L\" \"Sets the variable to be plotted on the left (L) or␣
↪right (R) y-axis.\";
  input String yLabelRight = \"\" \"This text will be used as the right vertical␣
↪label in the diagram.\";
  input Real yRangeRight[2] = {0.0, 0.0} \"Determines the right vertical interval␣
↪that is visible in the diagram. {0,0} will select a suitable range.\";
  output Boolean success \"Returns true on success\";
end plot;"
```

CHAPTER

FIVE

OPENMODELICA COMPILER

The OpenModelica Compiler (OMC) consists of a frontend, backend, code generation and the runtimes.

1. **Lexical Analysis**

   Keywords, operators and identifiers are extracted from the model.

2. **Parsing**

   An abstract syntax tree represented in Meta-Modelica is created from the operators and identifiers.

3. **Semantic Analysis**

   The abstract syntax tree gets tested for semantic errors.

4. **Intermediate Representation**

   Translation of the abstract syntax tree to an intermediate representation called SCode in MetaModelica. This is further processed by the frontend producing DAE intermediate representation code.

5. **Symbolic Optimization Backend**

   The intermediate representation gets optimized and preprocessed.

6. **Code Generation**

   Executable code gets generated from the low level intermediate representation.

For more details see [FPA+20]. A full list of compiler flags can be found in *OpenModelica Compiler Flags*.

5.1 Frontend Modules

5.2 Backend Modules

1. **Pre-Optimization**
   - Partitioning
   - Alias removal
2. **Causalization**
   - Matching
   - Sorting
   - Index reduction
3. **Post-Optimization**
   - Tearing
   - Jacobian

5.2.1 Backend DAE Info

With compiler debug flag *backenddaeinfo* it is possible to get additional information from the Backend modules.

- Number of equations / variables
- Number of states
- Information about initialization and simulation system
  - Equation types
  - Equation system details (linear and non-linear)

The output of *backenddaeinfo* can be expanded by using additional compiler debug flags *stateselection* and *discreteinfo*.

Example

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.
↪mo")
true
>>> setCommandLineOptions("-d=backenddaeinfo,stateselection,discreteinfo")
true
>>> translateModel(BouncingBall)
true
"Notification: Model statistics after passing the front-end and creating the data␣
↪structures used by the back-end:
 * Number of equations: 6
 * Number of variables: 6
Notification: Model statistics after passing the back-end for initialization:
 * Number of independent subsystems: 3
 * Number of states: 0 ()
 * Number of discrete variables: 9 (v_new,$PRE.v_new,flying,$PRE.flying,impact,foo,
↪$whenCondition1,$whenCondition2,$whenCondition3)
 * Number of discrete states: 0 ()
 * Number of clocked states: 0 ()
 * Top-level inputs: 0
Notification: Strong component statistics for initialization (13):
 * Single equations (assignments): 13
 * Array equations: 0
 * Algorithm blocks: 0
 * Record equations: 0
 * When equations: 0
 * If-equations: 0
 * Equation systems (not torn): 0
 * Torn equation systems: 0
 * Mixed (continuous/discrete) equation systems: 0
Notification: Model statistics after passing the back-end for simulation:
 * Number of independent subsystems: 1
 * Number of states: 2 (v,h)
 * Number of discrete variables: 7 ($whenCondition3,$whenCondition2,$whenCondition1,
↪foo,v_new,impact,flying)
 * Number of discrete states: 2 (impact,v)
 * Number of clocked states: 0 ()
 * Top-level inputs: 0
Notification: Strong component statistics for simulation (9):
 * Single equations (assignments): 7
 * Array equations: 0
 * Algorithm blocks: 0
 * Record equations: 0
```

(continues on next page)

(continued from previous page)

```
* When equations: 2
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0"
```

5.3 Code generation

From the low level intermediate representation from the backend code will be generated. The default code generation target is C and offers the largest model coverage. An alternative is the C++ (*Cpp*) which can produce significant faster executables in some cases.

The target language can be changed with compiler flag *--simCodeTarget*.

Depending on the target the compiler will write code and compile everything into a single simulation executable.

5.4 Simulation Runtimes

The generated code is linked with the appropriate runtime.

5.4.1 C Runtime

In *Solving Modelica Models* the methods implemented in the C runtime are described. In *C Runtime Simulation Flags* the runtime flags are documented.

5.4.2 C++ Runtime

Solver methods and runtime flags are currently undocumented. Refer to the source code

5.4.3 References

CHAPTER

SIX

SOLVING MODELICA MODELS

6.1 Integration Methods

By default OpenModelica transforms a Modelica model into an ODE representation to perform a simulation by using numerical integration methods. This section contains additional information about the different integration methods in OpenModelica. They can be selected by the method parameter of the *simulate* command or the *-s simflag*.

The different methods are also called solver and can be distinguished by their characteristic:

- Method
- Explicit vs. implicit
- Suitability for stiff systems
- Usage of sparse methods to solver underlying equation systems
- Integration order
- Step size control: Fixed vs. adaptive

Solver	Method	Type	System	Sparsity	Order	Step Size
DASSL	BDF	imp.	stiff	sparse / dense	adaptive 1-5	adaptive
IDA	BDF	imp.	stiff	sparse / dense	adaptive 1-5	adaptive
CVODE	Adams- Moulton	imp.	non-stiff	dense	adaptive 1-12	adaptive
CVODE	BDF	imp.	stiff	dense	adaptive 1-5	adaptive
GBODE	RK	exp.	non-stiff	sparse / dense	1-14	adaptive
GBODE	RK	imp.	stiff	sparse / dense	1-12	adaptive
GBODE	RK	imp.	multi-rate	sparse / dense	1-14	adaptive
Euler	Euler	exp.	non-stiff	dense	1	fixed

A good introduction on this topic may be found in [CK06] and a more mathematical approach can be found in [HNorsettW93].

6.1.1 DASSL

DASSL is the default solver in OpenModelica, because of a severals reasons. It is an implicit, higher order, multi-step solver with a step-size control and with these properties it is quite stable for a wide range of models. Furthermore it has a mature source code, which was originally developed in the eighties an initial description may be found in [Pet82].

This solver is based on backward differentiation formula (BDF), which is a family of implicit methods for numerical integration. The used implementation is called DASPK2.0 (see[1]) and it is translated automatically to C by f2c (see[2]).

[1] DASPK Webpage
[2] Cdaskr source

Internal non-linear and linear equation systems are solved using dense methods. If the target model is known to have a sparse structure one of the sparse solvers might be a better alternative.

The following simulation flags can be used to adjust the behavior of the solver for specific simulation problems: *jacobian*, *noRootFinding*, *noRestart*, *initialStepSize*, *maxStepSize*, *maxIntegrationOrder*, *noEquidistantTimeGrid*.

6.1.2 IDA

The IDA solver is part of a software family called sundials: SUite of Nonlinear and DIfferential/ALgebraic equation Solvers [HBG+05]. The implementation is based on DASPK with an extended linear solver interface, which includes an interface to the high performance sparse non-lineaer solver KINSOL [HBG+05] and linear solver KLU [DN10].

The simulation flags of *DASSL* are also valid for the IDA solver and furthermore it has the following IDA specific flags: *idaLS*, *idaMaxNonLinIters*, *idaMaxConvFails*, *idaNonLinConvCoef*, *idaMaxErrorTestFails*.

6.1.3 CVODE

The CVODE solver is part of sundials: SUite of Nonlinear and DIfferential/ALgebraic equation Solvers [HBG+05]. CVODE solves initial value problems for ordinary differential equation (ODE) systems with variable-order, variable-step multistep methods.

In OpenModelica, CVODE uses a combination of Backward Differentiation Formulas (varying order 1 to 5) as linear multi-step method and a modified Newton iteration with fixed Jacobian as non-linear solver per default. This setting is advised for stiff problems which are very common for Modelica models. For non-stiff problems an combination of an Adams-Moulton formula (varying order 1 to 12) as linear multi-step method together with a fixed-point iteration as non-linear solver method can be chosen.

Both non-linear solver methods are internal functions of CVODE and use its internal direct dense linear solver CVDense. For the Jacobian of the ODE CVODE will use its internal dense difference quotient approximation.

CVODE has the following solver specific flags: *cvodeNonlinearSolverIteration*, *cvodeLinearMultistepMethod*.

6.1.4 GBODE

GBODE stands for Generic Bi-rate ordinary differential equation (ODE) solver and is a generic implementation for any Runge-Kutta (RK) scheme [HNorsettW00]. In GBODE there are already many different implicit and explicit RK methods (e.g. SDIRK, ESDIRK, Gauss, Radau, Lobatto, Fehlberg, DOPRI45, Merson) with different approximation order configurable and ready to use. New RK schemes can easily be added, if the corresponding Butcher tableau is available. By default the solver runs in single-rate mode using the embedded RK scheme ESDIRK4 [KC19] with variable-step-size control and efficient event handling.

The bi-rate mode can be utilized using the simulation flag *gbratio*. This flag determines the percentage of fast states with respect to all states. These states will then be automatically detected during integration based on the estimated approximation error and afterwards refined using an appropriate inner step-size control and interpolated values of the slow states.

The solver utilizes by default the sparsity pattern of the ODE Jacobian and solves the corresponding non-linear system in case of an implicit chosen RK scheme using sparse solver KINSOL.

GBODE is highly configurable and the following simulation flags can be used to adjust the behavior of the solver for specific simulation problems: *gbratio*, *gbm*, *gbctrl*, *gbnls*, *gbint*, *gberr*, *gbfm*, *gbfctrl*, *gbfnls*, *gbfint*, *gbferr*.

6.1.5 Basic Explicit Solvers

The basic explicit solvers Euler uses a fixed step-size based on the `numberOfIntervals`, the `startTime` and `stopTime` parameters in the *simulate* command: $\text{stepSize} \approx \frac{\text{stopTime} - \text{startTime}}{\text{numberOfIntervals}}$

- euler - Explicit Euler, fixed step size, order 1

6.1.6 Deprecated Solvers

The following solvers are deprecated and will be removed in a future version of OpenModelica:

- rungekutta - Classic Runge-Kutta method RK4, explicit, fixed step-size, oder 4

```
old: -s=rungekutta
new: -s=gbode -gbm=rungekutta -gbctrl=const
```

- symSolver - Symbolic inline solver (requires *--symSolver*) - fixed step-size, order 1
- symSolverSsc - Symbolic implicit inline Euler with step-size control (requires *--symSolver*) - step-size control, order 1-2
- qss - A QSS solver

6.2 DAE Mode Simulation

Beside the default ODE simulation, OpenModelica is able to simulate models in *DAE mode*. The *DAE mode* is enabled by the flag *--daeMode*. In general the whole equation system of a model is passed to the DAE integrator, including all algebraic loops. This reduces the amount of work that needs to be done in the post optimization phase of the OpenModelica backend. Thus models with large algebraic loops might compile faster in *DAE mode*.

Once a model is compiled in *DAE mode* the simulation can be only performed with *SUNDIALS/IDA* integrator and with enabled *-daeMode* simulation flag. Both are enabled automatically by default, when a simulation run is started.

6.2.1 Homotopy support in DAE mode

In *DAE mode* the *homotopy operator* is kept in the simulation residual (in ODE mode it is only used during initialization). This makes *DAE mode* robust against models whose *initial operating point is a degenerate point* of a `homotopy(actual, simplified)` characteristic — a typical example is a fan or pump that starts at zero speed and zero flow, where the derivative of the actual pressure characteristic vanishes and the initial DAE Jacobian is singular. Such models would otherwise fail at `time = 0` with `IDA_LSETUP_FAIL`.

When this happens, OpenModelica automatically performs a homotopy continuation: the homotopy parameter λ is ramped from 0 to 1 over the first part of the simulation interval (10% by default), so the simulation starts from the regularized *simplified* system and continuously transitions to the *actual* system once it has moved off the degenerate point. The continuation is activated only after a singular initial Jacobian is detected, so models that integrate normally are unaffected, and it removes the need for a model-level workaround such as giving the mover a non-zero `m_flow_start`.

6.3 Initialization

To simulate an ODE representation of an Modelica model with one of the methods shown in *Integration Methods* a valid initial state is needed. Equations from an initial equation or initial algorithm block define a desired initial system.

6.3.1 Choosing start values

Only non-linear iteration variables in non-linear strong components require start values. All other start values will have no influence on convergence of the initial system.

Use *-d=initialization* to show additional information from the initialization process. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call setCommandLineOptions("-d=initialization")

```
model piston
  Modelica.Mechanics.MultiBody.Parts.Fixed fixed1 annotation(
    Placement(visible = true, transformation(origin = {-80, 70}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 0)));
```

(continues on next page)

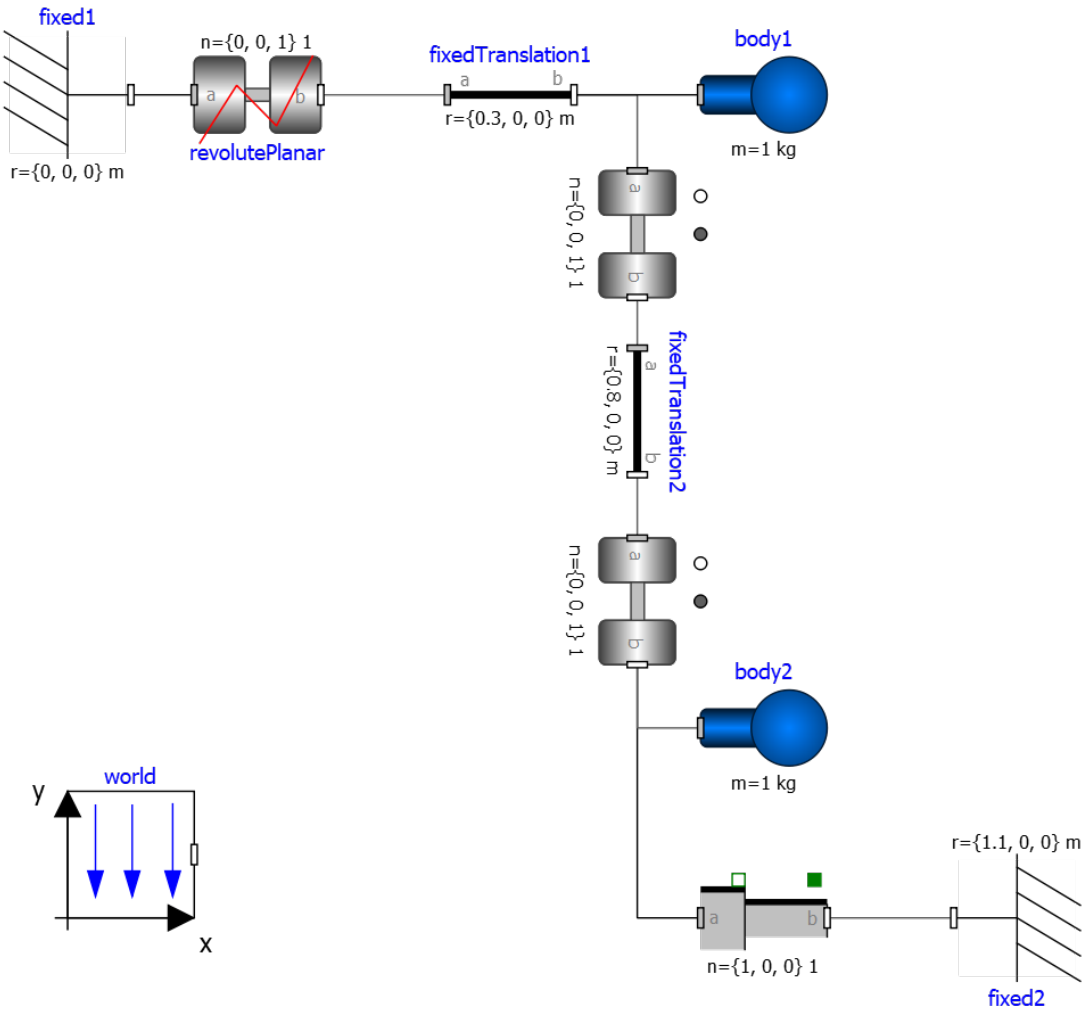


Figure 6.1: piston.mo

(continued from previous page)

```
  Modelica.Mechanics.MultiBody.Parts.Body body1(m = 1)  annotation(
    Placement(visible = true, transformation(origin = {30, 70}, extent = {{-10, -10},
↪{10, 10}}, rotation = 0)));
  Modelica.Mechanics.MultiBody.Parts.FixedTranslation fixedTranslation1(r = {0.3, 0,␣
↪0})  annotation(
    Placement(visible = true, transformation(origin = {-10, 70}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 0)));
  Modelica.Mechanics.MultiBody.Parts.FixedTranslation fixedTranslation2(r = {0.8, 0,␣
↪0})  annotation(
    Placement(visible = true, transformation(origin = {10, 20}, extent = {{-10, -10},
↪{10, 10}}, rotation = -90)));
  Modelica.Mechanics.MultiBody.Parts.Fixed fixed2(animation = false, r = {1.1, 0, 0})␣
↪ annotation(
    Placement(visible = true, transformation(origin = {70, -60}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 180)));
  Modelica.Mechanics.MultiBody.Parts.Body body2(m = 1)  annotation(
    Placement(visible = true, transformation(origin = {30, -30}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 0)));
  inner Modelica.Mechanics.MultiBody.World world annotation(
    Placement(visible = true, transformation(origin = {-70, -50}, extent = {{-10, -10}
↪, {10, 10}}, rotation = 0)));
  Modelica.Mechanics.MultiBody.Joints.Prismatic prismatic(animation = true) ␣
↪annotation(
    Placement(visible = true, transformation(origin = {30, -60}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 0)));
  Modelica.Mechanics.MultiBody.Joints.RevolutePlanarLoopConstraint revolutePlanar␣
↪annotation(
    Placement(visible = true, transformation(origin = {-50, 70}, extent = {{-10, -10},
↪ {10, 10}}, rotation = 0)));
  Modelica.Mechanics.MultiBody.Joints.Revolute revolute1(a(fixed = false),phi(fixed =␣
↪false), w(fixed = false))  annotation(
    Placement(visible = true, transformation(origin = {10, 48}, extent = {{-10, -10},
↪{10, 10}}, rotation = -90)));
  Modelica.Mechanics.MultiBody.Joints.Revolute revolute2 annotation(
    Placement(visible = true, transformation(origin = {10, -10}, extent = {{-10, -10},
↪ {10, 10}}, rotation = -90)));
equation
  connect(prismatic.frame_b, fixed2.frame_b) annotation(
    Line(points = {{40, -60}, {60, -60}, {60, -60}, {60, -60}}, color = {95, 95, 95}
↪));
  connect(fixed1.frame_b, revolutePlanar.frame_a) annotation(
    Line(points = {{-70, 70}, {-60, 70}, {-60, 70}, {-60, 70}}));
  connect(revolutePlanar.frame_b, fixedTranslation1.frame_a) annotation(
    Line(points = {{-40, 70}, {-20, 70}, {-20, 70}, {-20, 70}}, color = {95, 95, 95}
↪));
  connect(fixedTranslation1.frame_b, revolute1.frame_a) annotation(
    Line(points = {{0, 70}, {10, 70}, {10, 58}, {10, 58}}, color = {95, 95, 95}));
  connect(revolute1.frame_b, fixedTranslation2.frame_a) annotation(
    Line(points = {{10, 38}, {10, 38}, {10, 30}, {10, 30}}, color = {95, 95, 95}));
  connect(revolute2.frame_b, prismatic.frame_a) annotation(
    Line(points = {{10, -20}, {10, -20}, {10, -60}, {20, -60}, {20, -60}}));
  connect(revolute2.frame_b, body2.frame_a) annotation(
    Line(points = {{10, -20}, {10, -20}, {10, -30}, {20, -30}, {20, -30}}, color =
↪{95, 95, 95}));
  connect(revolute2.frame_a, fixedTranslation2.frame_b) annotation(
    Line(points = {{10, 0}, {10, 0}, {10, 10}, {10, 10}}, color = {95, 95, 95}));
```

(continues on next page)

(continued from previous page)

```
  connect(fixedTranslation1.frame_b, body1.frame_a) annotation(
    Line(points = {{0, 70}, {18, 70}, {18, 70}, {20, 70}}));
end piston;
```

```
>>> loadModel(Modelica);
>>> setCommandLineOptions("-d=initialization");
>>> buildModel(piston);
"Warning: Parameter body1.r_CM has no value, and is fixed during initialization␣
↪(fixed=true), using available start value (start={0, 0, 0}) as default value.
Warning: Parameter body2.r_CM has no value, and is fixed during initialization␣
↪(fixed=true), using available start value (start={0, 0, 0}) as default value.
Warning: Assuming fixed start value for the following 2 variables:
         $STATESET2.x:VARIABLE(start = /*Real*/($STATESET2.A[1]) * $START.revolute2.
↪phi + /*Real*/($STATESET2.A[2]) * $START.revolute1.phi fixed = true )  type: Real
         $STATESET1.x:VARIABLE(start = /*Real*/($STATESET1.A[1]) * $START.revolute1.w␣
↪+ /*Real*/($STATESET1.A[2]) * $START.revolute2.w fixed = true )  type: Real"
```

Note how OpenModelica will inform the user about relevant and irrelevant start values for this model and for which variables a fixed default start value is assumed. The model has four joints but only one degree of freedom, so one of the joints *revolutePlanar* or *prismatic* must be initialized.

So, initializing *phi* and *w* of *revolutePlanar* will give a sensible start system.

```
model pistonInitialize
  extends piston(revolute1.phi.fixed = true, revolute1.phi.start = -1.221730476396031,
↪ revolute1.w.fixed = true, revolute1.w.start = 5);
equation
end pistonInitialize;
```

```
>>> setCommandLineOptions("-d=initialization");
>>> simulate(pistonInitialize, stopTime=2.0);
"Warning: Parameter body1.r_CM has no value, and is fixed during initialization␣
↪(fixed=true), using available start value (start={0, 0, 0}) as default value.
Warning: Parameter body2.r_CM has no value, and is fixed during initialization␣
↪(fixed=true), using available start value (start={0, 0, 0}) as default value."
```

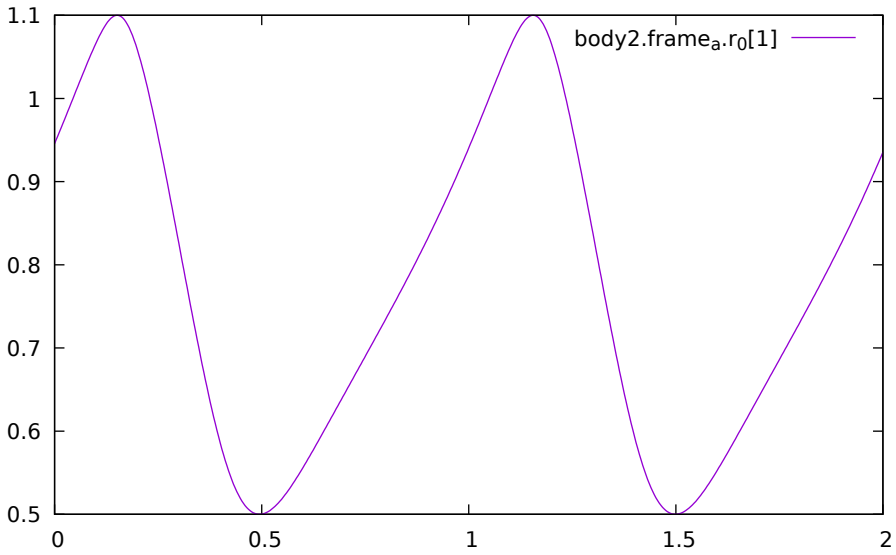


Figure 6.2: Vertical movement of mass body2.

6.3.2 Importing initial values from previous simulations

In many use cases it is useful to import initial values from previous simulations, possibly obtained with another Modelica tool, which are saved in a .mat file. There are two different options to do that.

Using previous simulation results as start values for the initial equations

The first option is to solve the initial equations specified by the Modelica model, using the previous simulation results as initial guesses for the iterative solvers, in case they are needed. This can be very helpful in case the initialization problem involves the solution of large nonlinear systems of equations by means of iterative algorithms, whose convergence is sensitive to the selected initial guess.

Importing a previously found solution allows the OpenModelica solver to pick very good initial guesses for the unknowns of the iterative solvers, thus achieving convergence with a few iterations. Since the initial equations are solved in the process, the values of all variables and derivatives, as well as of all parameters with *fixed = false* attribute, are re-computed and fully consistent with the selected initial conditions, even in case the previously saved simulation results refer to a slightly different model configuration. Note that parameters with *fixed = true* will also get their values from the imported .mat file, so if you want to change them you need to edit the .mat file accordingly.

This option is activated by selecting the simulation result file name in the OMEdit *Simulation Setup | Simulation Flag | Equation System Initialization File* input field, or by setting the additional simulation flag *-iif=resultfile.mat*. By activating the checkbox *Save simulation flags inside the model i.e., __OpenModelica_simulationFlags annotation*, a custom annotation *__OpenModelica_simulationFlags(iif="filename.mat")* is added to the model, so this setting is saved with the model and is reused when loading the model again later on. It is also possible to specify at which point in time of the saved simulation results the initial values should be picked, by means of the *Simulation Setup | Simulation Flags | Equation System Initialization Time* input field, or by setting the simulation flag *-iit=initialTimeValue*.

Using previous simulation results to directly initialize a simulation

The second option is to skip the solution of the initial equations entirely, directly starting the simulation with the imported start values. In this case, the initial equations of the model are ignored, and the initial values of all parameters and state variables are directly set to the values loaded from the .mat file. This option is useful to restart a simulation from the final state of a previous one, ignoring whatever initial conditions are declared in the Modelica model by either fixed = true attributes or initial equations.

Note that state variables and parameters will be directly initialized to the imported values, while algebraic variables will be recomputed with the regular simulation equations, on the basis of the imported initial state and parameter values. The values of algebraic variables in the imported file will only be used to initialize iteration variables, in case this computation involves nonlinear implicit equations and iterative solvers, otherwise they will be ignored.

To activate this second option, set *Simulation Setup | Simulation Flag | Initialization Method* to *none* in OMEdit, or set the simulation flag *-iim=none*. Also in this case, activating the checkbox *Save simulation flags inside model, i.e. __OpenModelica_simulationFlags annotation* saves this option in an *__OpenModelica_simulationFlags(iim=none)* annotation, so it is retained for future simulations of the same model.

Beware of missing variables in the simulation result file

When importing simulation results to initialize a new simulation, bear in mind that, by default, protected and hidden variables are *not saved* to the .mat file, so they will be missing when importing the simulation results for initialization. This is particularly dangerous when the imported values are used directly, as unsaved protected or hidden variables will not be initialized and will retain their default zero value, which is likely to cause numerical problems such as division by zero when the simulation is started.

To avoid this problem, make sure you also save protected and hidden variables in the simulation result file, by setting the corresponding checkboxes in the *Simulation Setup | Output* tab of OMEdit, or by setting the simulation flags *-emit_protected* and *-ignoreHideResult*.

Example of use

The following minimal working example demonstrates the use of the initial value import feature. You can create a new package *ImportInitialValues* in OMEdit, copy and paste its code from here, and then run the different models in it.

```
package ImportInitialValues "Test cases for importing initial values in OpenModelica"
  partial model Base "The mother of all models"
    Real v1, v2, x;
    parameter Real p1;
    parameter Real p2 = 2*p1;
    final Real p3 = 3*p1;
  end Base;

  model ResultFileGenerator "Dummy model for generating the initial.mat file"
    extends Base(p1 = 7, p2 = 10);
  equation
    v1 = 2.8;
    v2 = 10;
    der(x) = 0;
  initial equation
    x = 4;
  annotation(
    experiment(StopTime = 1),
    __OpenModelica_simulationFlags(r = "../initial.mat"));
  end ResultFileGenerator;

  model M "Relies on Modelica code only for initialization"
    extends Base(
      v1(start = 14),
      p1 = 1, p2 = 1);
  equation
    (v1 - 3)*(v1 + 10)*(v1 - 15) = 0;
    v2 = time;
    der(x) = -x;
  initial equation
    x = 6;
  end M;

  model M2 "Imports parameters and initial guesses only, solve initial equations"
    extends M;
  annotation(__OpenModelica_simulationFlags(iif = "../initial.mat"));
  end M2;

  model M3 "import parameters, initial guesses and initial states, skip initial␣
↪equations"
    extends M;
  annotation(__OpenModelica_simulationFlags(iim = "none", iif = "../initial.mat"));
  end M3;
end ImportInitialValues;
```

Running the *ResultFileGenerator* model creates an *initial.mat* file with some initial values in the root OMEdit working directory: $p1 = 7$, $p2 = 10$, $p3 = 21$, $v1 = 2.8$, $v2 = 10$, $x = 4$, $der(x) = 0$. Note that the default directory when running simulations in OMEdit is a sub-directory named as the full model pathname, located in the working directory, so it is necessary to go up one directory to access the root working directory.

When running model *M*, the simulation process only relies on the initial and guess values provided by the Modelica source code. Regarding the parameter values, $p1 = 1$, $p2 = 1$, $p3 = 3*p1 = 3$; regarding *v1*, the implicit cubic equation is solved iteratively using the start value 14 as an initial guess, thus converging to the nearest solution *v1*

= *15*. The other variable *v2* can be computed explicitly, so there is no need of any guess value for it. Finally, the initial value of the state variable is set to *x = 6* by the initial equations.

When running model *M2*, the values of the *initial.mat* file are imported to provide values for non-final parameters and guess values for the initial equations, which are solved starting from there. Hence, the imported parameter values p1 = 7 and p2 = 10 override the model's binding equations, that would set both to 1; on the other hand, the final parameter p3 is computed based on the final binding equation to *p3 = p1*3 = 21*. Regarding *v1*, the iterative solver converges to the solution closest to the imported start value of 2.8, i.e. *v1 = 3*, while *v2* is computed explicitly, so it doesn't depend on the imported start value, which is ignored. The initial value of the state *x = 6* is obtained by solving the initial equation, which is explicit and thus ignores the imported guess value *x = 4*.

Finally, when running model *M3*, parameters are handled like in the previous case, as well as the algebraic variables *v1* and *v2*. However, in this case the solution of the initial equations is skipped, so the state variable gets its initial value *x = 4* straight from the imported *initial.mat* file.

6.3.3 Homotopy Method

For complex start conditions OpenModelica can have trouble finding a solution for the initialization problem with the default Newton method.

Modelica offers the homotopy operator[3] to formulate *actual* and *simplified* expression for equations, with homotopy parameter λ going from 0 to 1:

$$actual \cdot \lambda + simplified \cdot (1 - \lambda).$$

OpenModelica has different solvers available for non-linear systems. Initializing with homotopy on the first try is default if a homotopy operator is used. It can be switched off with *noHomotopyOnFirstTry*. For a general overview see [SCO+11], [Sie12], for details on the implementation in OpenModelica see [OB13].

The homotopy methods distinguish between local and global methods meaning, if λ affects the entire initialization system or only local strong connected components. In addition the homotopy methods can use equidistant λ or and adaptive λ in [0,1].

Default order of methods tried to solve initialization system

If there is no homotopy in the model

- Solve without homotopy method.

If there is homotopy in the model or solving without homotopy failed

- Try global homotopy approach with equidistant λ.

The default homotopy method will do three global equidistant steps from 0 to 1 to solve the initialization system.

Several compiler and simulation flags influence initialization with homotopy: *--homotopyApproach*, *-homAdaptBend*, *-homBacktraceStrategy*, *-homHEps*, *-homMaxLambdaSteps*, *-homMaxNewtonSteps*, *-homMaxTries*, *-homNegStartDir*, *-homotopyOnFirstTry*, *-homTauDecFac*, *-homTauDecFacPredictor*, *-homTauIncFac*, *-homTauIncThreshold*, *-homTauMax*, *-homTauMin*, *-homTauStart*, *-ils*.

6.4 Tearing

The size of linear and nonlinear equation systems can be substantially reduced by means of the Tearing method. Consider a system of N equations. The Tearing method requires to pick $M < N$ variables x_t as *tearing* or *iteration* variables, so that assuming their values are known, $N - M$ *torn* equations can be solved explicitly for the remaining $N - M$ *torn* variables, by sorting them appropriately. Then, the remaining M equations are put in *residual* form $f(x_t) = 0$, where the residuals can ultimately be computed by explicit computations as a function of the tearing variables x_t only. The result is thus an equivalent implicit system of $M < N$ equations in the M tearing variables, with an explicit procedure to compute the residual function $f(x_t)$. The Jacobian of that function, which is required by the Newton method, can then be obtained by either symbolic or numerical differentiation techniques.

The Tearing method has three main advantages:

[3] Modelica Association, Modelica® - A Unified Object-Oriented Language for Systems Modeling Language Specification - Version 3.4, 2017 - Section 3.7.2.4

- the size of the Jacobian matrix to be factorized in order to solve it is greatly reduced;
- for nonlinear systems solved by iterative methods like Newton-Raphson, it is only necessary to give initial guess values to the much smaller set of variables x_t; the initial guess values are set to the start attributes of the tearing variables;
- the method allows to solve mixed systems containing Real and discrete (Boolean or Integer) variables and equations by means of standard nonlinear equation solvers, as long as the discrete variables are selected as torn variables and the resulting residual equations have a continuous dependency on the Real tearing variables.

OpenModelica implements some heuristic algorithms to automatically choose the set of tearing variables. The tearing algorithm can be selected with the compiler flags: *--tearingMethod*, *--tearingHeuristic*. Since the tearing algorithms can be very time-consuming for large systems, they are automatically disabled for systems above a certain size, see *--maxSizeLinearTearing*, *--maxSizeNonlinearTearing*.

As of Modelica 3.6, there is no standardized way to influence the choice of tearing variables. OpenModelica provides a custom *__OpenModelica_tearingSelect* annotation that can be added to variable declarations to influence the choice of tearing variables:

```
Real x annotation(__OpenModelica_tearingSelect = TearingSelect.always);
Real y annotation(__OpenModelica_tearingSelect = TearingSelect.prefer);
Real z annotation(__OpenModelica_tearingSelect = TearingSelect.default);
Real v annotation(__OpenModelica_tearingSelect = TearingSelect.avoid);
Real w annotation(__OpenModelica_tearingSelect = TearingSelect.never);
```

This feature is currently experimental. There is discussion going on within the MAP-Lang group of the Modelica Association to standardize features for the selection of tearing variables and residual equations.

6.5 Algebraic Solvers

If the ODE system contains equations that need to be solved together, so called algebraic loops, OpenModelica can use a variety of different linear and non-linear methods to solve the equation system during simulation.

For the C runtime the linear solver can be set with *-ls* and the non-linear solver with *-nls*. There are dense and sparse solver available.

Linear solvers

- *default* : Lapack with totalpivot as fallback [ABB+99]
- *lapack* : Non-Sparse LU factorization using [ABB+99]
- *lis* : Iterative linear solver [Nis10]
- *klu* : Sparse LU factorization [Nat05]
- *umfpack* : Sparse unsymmetric multifrontal LU factorization [Dav04]
- *totalpivot* : Total pivoting LU factorization for underdetermined systems

Non-linear solvers

- *hybrid* : Modified Powell hybrid method from MINPACK [DJS96]
- *kinsol* : Combination of Newton-Krylov, Picard and fixed-point solver [T+98]
- *newton* : Newton-Raphson method [CK06]
- *mixed* : Homotopy with hybrid as fallback [Kel78] [BBOR15]
- *homotopy* : Damped Newton solver with fixed-point solver and Newton homotopy solver as fallbacks

In addition, there are further optional settings for the algebraic solvers available. A few of them are listed in the following:

General: *-nlsLS*

Newton: *-newton -newtonFTol -newtonMaxStepFactor -newtonXTol*

Sparse solver: *-nlssMinSize -nlssMaxDensity*

Enable logging: *-lv=LOG_LS -lv=LOG_LS_V -lv=LOG_NLS -lv=LOG_NLS_V*

6.5.1 References

CHAPTER

SEVEN

AUTOMATIC PARALLELIZATION OF SIMULATIONS

OpenModelica can automatically parallelize the simulation of a model over the available CPU cores. The feature is enabled by building the model with the `--parmodauto` compiler flag:

```
setCommandLineOptions("--parmodauto");
loadModel(Modelica);
buildModel(Modelica.Fluid.Examples.BranchingDynamicPipes, fileNamePrefix="m");
```

With `--parmodauto` the compiler emits, alongside the usual simulation code, a description of the model's equation **task graph** (which equation depends on which). At run time the executable reads that description, groups the tasks into **clusters**, builds a task graph over the clusters and executes it in parallel with oneTBB.

How the tasks are grouped into clusters — the *clustering* — determines how well the work balances across the cores. The runtime ships several clustering strategies, and it can also **export** the task graph and clustering to JSON and **import** a clustering produced by an external tool. This makes it possible to develop and tune clustering/scheduling outside of the C++ runtime, for example with graph-visualization tools or with an optimizer, and then feed the result back into the simulation.

7.1 Run-time flags

The following simulation-executable flags control the clustering and the JSON export/import. They are passed to the generated executable (e.g. via `simulate(M, simflags="...")` or on the command line).

```
-parmodScheduler=flow|level          select the scheduler (default: flow)
-parmodClustering=NAME               select the clustering strategy
                                     (default, fixed_width_min_height, none)
-parmodClustersPerLevel=INT          max clusters per level for the default clustering

-parmodExportTaskGraph=FILE          export the final task graph + clustering as JSON
-parmodDumpStages=PREFIX             export one JSON snapshot before and after every
                                     clustering optimization (PREFIX.00.initial.json,
                                     PREFIX.01.<opt>.json, ...)
-parmodImportClustering=FILE         use the clustering from FILE instead of␣
↪computing one
```

Importing a clustering aborts the simulation (with an error) if the file references an unknown or duplicated equation, or if the requested clustering would form a cycle in the contracted task graph.

7.2 JSON format

Everything is keyed by the **stable equation index** (`eq`), which is independent of the internal graph ordering:

```
{
  "name": "ModelName",
  "num_threads": 8,
```

(continues on next page)

(continued from previous page)

```
  "tasks":        [ {"eq": 42, "level": 3, "cost": 2.0, "out_degree": 2} ],
  "dependencies": [ [11, 42] ],
  "clusters":     [ {"eqs": [42, 43], "lane": 1} ],
  "stage": 1, "stage_name": "cluster_merge_common"
}
```

- `level` is the dependency depth (level 1 runs first), `cost` is the runtime's per-task cost estimate, and `out_degree` is the task's number of successors.
- `lane` is the hardware lane (core) `0 .. num_threads-1` assigned by lane-based clustering, or `-1` when the active clustering does not assign lanes.
- `stage` / `stage_name` appear only in `-parmodDumpStages` snapshots.
- For **import**, only `{ "clusters": [ {"eqs": [...]}, ... ] }` is required; equations not listed become singleton clusters.

7.3 External tools

OpenModelica ships a small toolchain (installed under `share/omc/scripts/`) that consumes and produces this JSON. Each tool reads any export or `-parmodDumpStages` snapshot.

`parmod_graph_to_graphml.py` *(Python, standard library only)*

Convert an export to GraphML for general graph tools (yEd, Gephi, NetworkX):

```
python3 parmod_graph_to_graphml.py taskgraph.json                # tasks,␣
↪colored by cluster
python3 parmod_graph_to_graphml.py --mode clusters taskgraph.json  # contracted␣
↪cluster graph
python3 parmod_graph_to_graphml.py --yed -o g.graphml taskgraph.json   # yEd,␣
↪pre-laid-out
```

Views (`--mode`): `tasks` (default), `clusters`, `lanes`, `both`.

`parmod_graph_plot.jl` *(Julia, GraphPlot.jl)*

Render an export to SVG (default) or PNG, laid out left-to-right by level and colored by cluster:

```
julia parmod_graph_plot.jl taskgraph.json
julia parmod_graph_plot.jl --mode lanes --png taskgraph.json
```

Dependencies: `JSON`, `Graphs`, `GraphPlot`, `Colors`, `Compose` (and `Cairo`, `Fontconfig` for `--png`). Best for small/medium graphs; for very large graphs prefer the GraphML route.

`parmod_optimize_clustering.jl` *(Julia, MetaheuristicsAlgorithms.jl)*

Optimize the clustering with a metaheuristic and write it back in the import format. The objective is the graph **height**: levels execute sequentially while the lanes within a level execute in parallel, so a level costs as much as its busiest lane and the height is the sum of those over all levels. Minimizing it balances each level's work across the `K` lanes (the *fixed-width / minimum height* goal). Clusters are keyed by `(level, lane)` so the result is always acyclic and safe to import.

```
julia parmod_optimize_clustering.jl taskgraph.json                  # GWO, K␣
↪= num_threads
julia parmod_optimize_clustering.jl --algorithm AEO --cores 8 --iters 500␣
↪taskgraph.json
julia parmod_optimize_clustering.jl --run --exe ./m taskgraph.json      #␣
↪optimize, then simulate
```

Options include `-a`/`--algorithm` (`GWO`, `WOA`, `AEO`, `SSA`, … — default `GWO`), `--pop`, `--iters`, `-k`/`--cores`, `--seed`, `-o`, and `--run`/`--exe`. It prints the serial and lower-bound reference heights, the op-

timized height, the speedup, and the improvement over the imported clustering. Install the dependency once with:

```
julia -e 'using Pkg; Pkg.add("JSON"); \
          Pkg.add(url="https://github.com/AbdelazimHussien/
↪MetaheuristicsAlgorithms.jl")'
```

parmod_optimize_demo.sh ***(Bash, end-to-end)***

Run the whole loop on a model and dump a GraphML/SVG of the clustering at every stage — both the optimizations performed inside the executable and the metaheuristic optimization performed in Julia — then report the differences:

```
MODELFILE=ParModelicaDemo.mo CORES=4 STOPTIME=1.0 \
  ./parmod_optimize_demo.sh ParModelicaDemo ./parmod_demo_out
./parmod_optimize_demo.sh --clean ./parmod_demo_out   # remove everything␣
↪regenerable
```

It builds the model with `--parmodauto`, exports the task graph and the per-stage clustering, renders them, runs `parmod_optimize_clustering.jl`, re-imports the optimized clustering, and finally compares the graph height (executable vs. optimized clustering), the simulation result (default vs. optimized clustering — these must match, since only the scheduling changes), and the wall-clock time of the two runs. `ParModelicaDemo.mo` is a small bundled model with a wide, shallow task graph that makes the clustering easy to see.

7.4 The optimization workflow

Putting it together, the clustering can be developed entirely outside the runtime and then used for the real simulation:

```
Modelica --omc --parmodauto-->                    executable
executable --parmodExportTaskGraph-->             taskgraph.json
taskgraph.json --parmod_optimize_clustering.jl--> optimized.json
optimized.json --parmodImportClustering-->        executable runs in parallel
```

Because the optimized clustering only changes how the equations are scheduled, the simulation results are identical to the default run; only the parallel execution differs.

CHAPTER

EIGHT

DEBUGGING

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

8.1 The Equation-based Debugger

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

8.1.1 Enable Tracing Symbolic Transformations

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

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

8.1.2 Load a Model to Debug

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

8.1.3 Simulate and Start the Debugger

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

8.1.4 Use the Transformation Debugger for Browsing

The debugger 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.

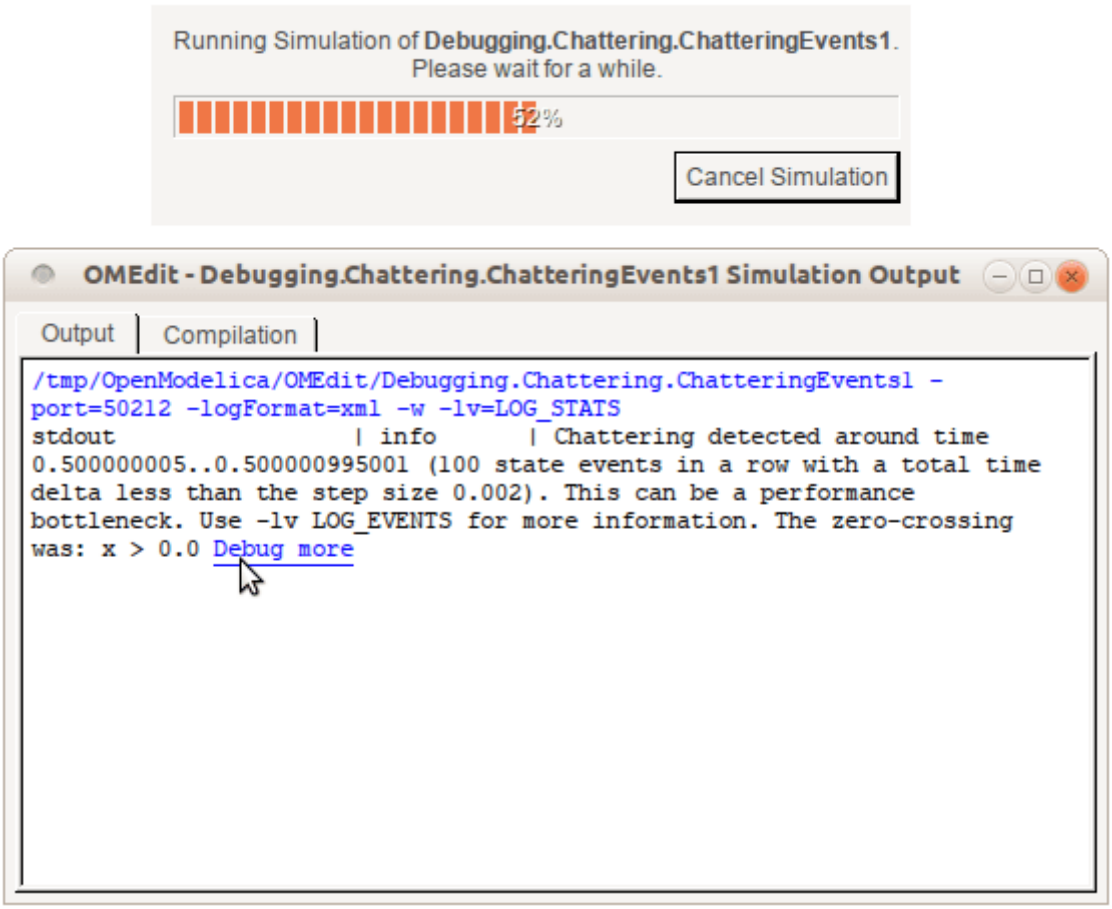


Figure 8.1: Simulating the model.

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

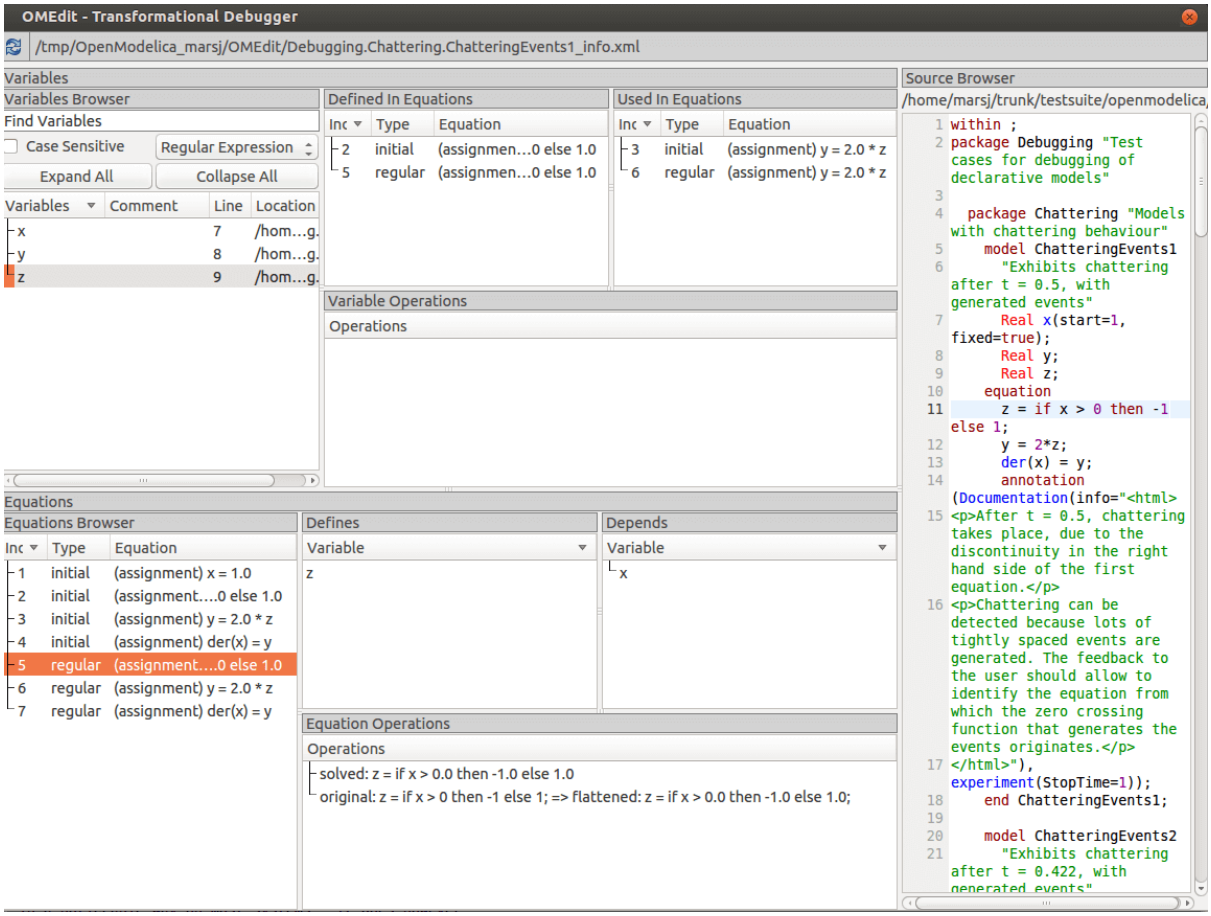


Figure 8.2: Transfomations Browser.

8.2 The Algorithmic Debugger

This section gives a short description how to get started using the algorithmic debugger in OMEdit. See section *Simulation Options* for further details of debugger options.

8.2.1 Adding Breakpoints

There are two ways to add the breakpoints,

- Click directly on the line number in Text View, a red circle is created indicating a breakpoint as shown in Figure 8.3.
- Open the Algorithmic Debugger window and add a breakpoint using the right click menu of Breakpoints Browser window.

8.2.2 Start the Algorithmic Debugger

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

There are four ways to start the debugger,

- Open the Simulation Setup and click on Launch Algorithmic Debugger before pressing Simulate.

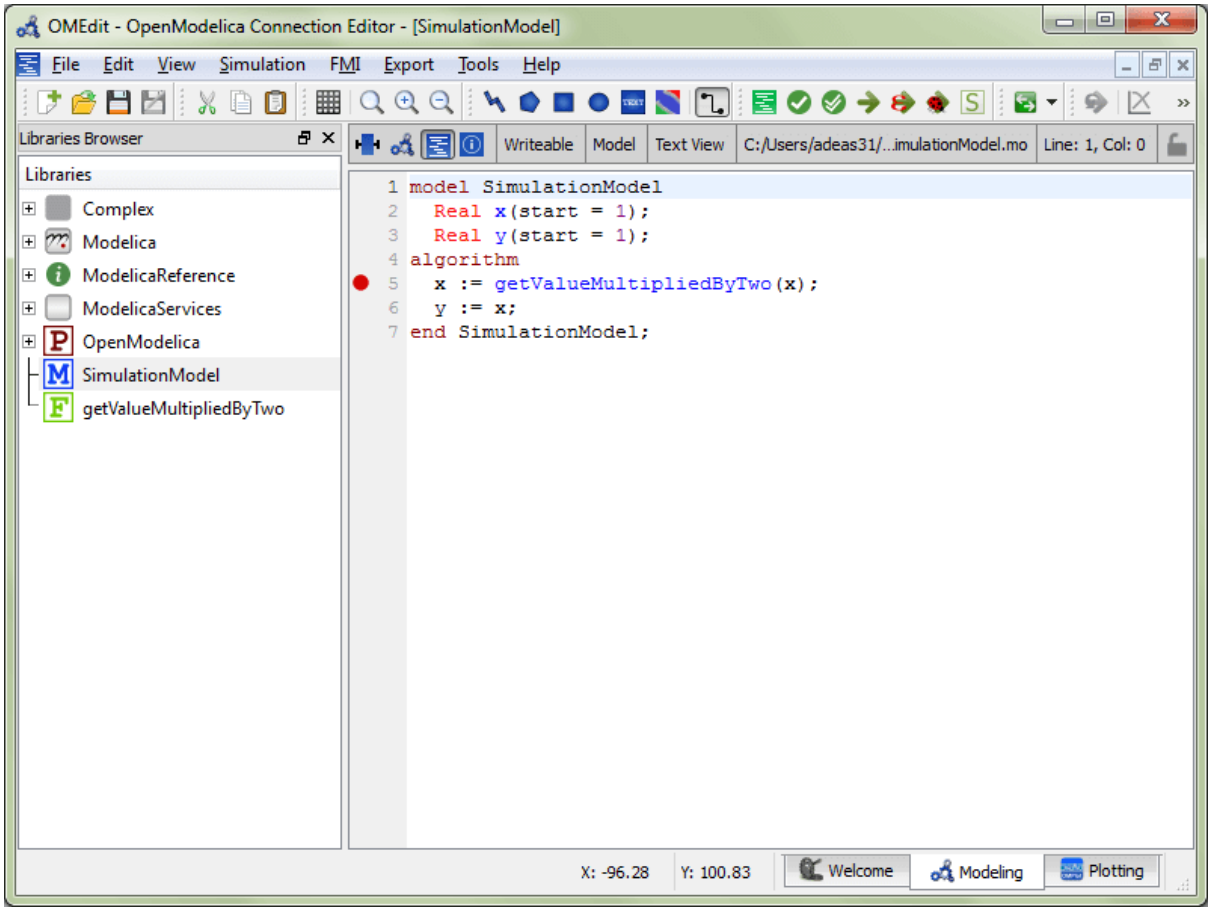


Figure 8.3: Adding breakpoint in Text View.

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

8.2.3 Debug Configurations

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

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

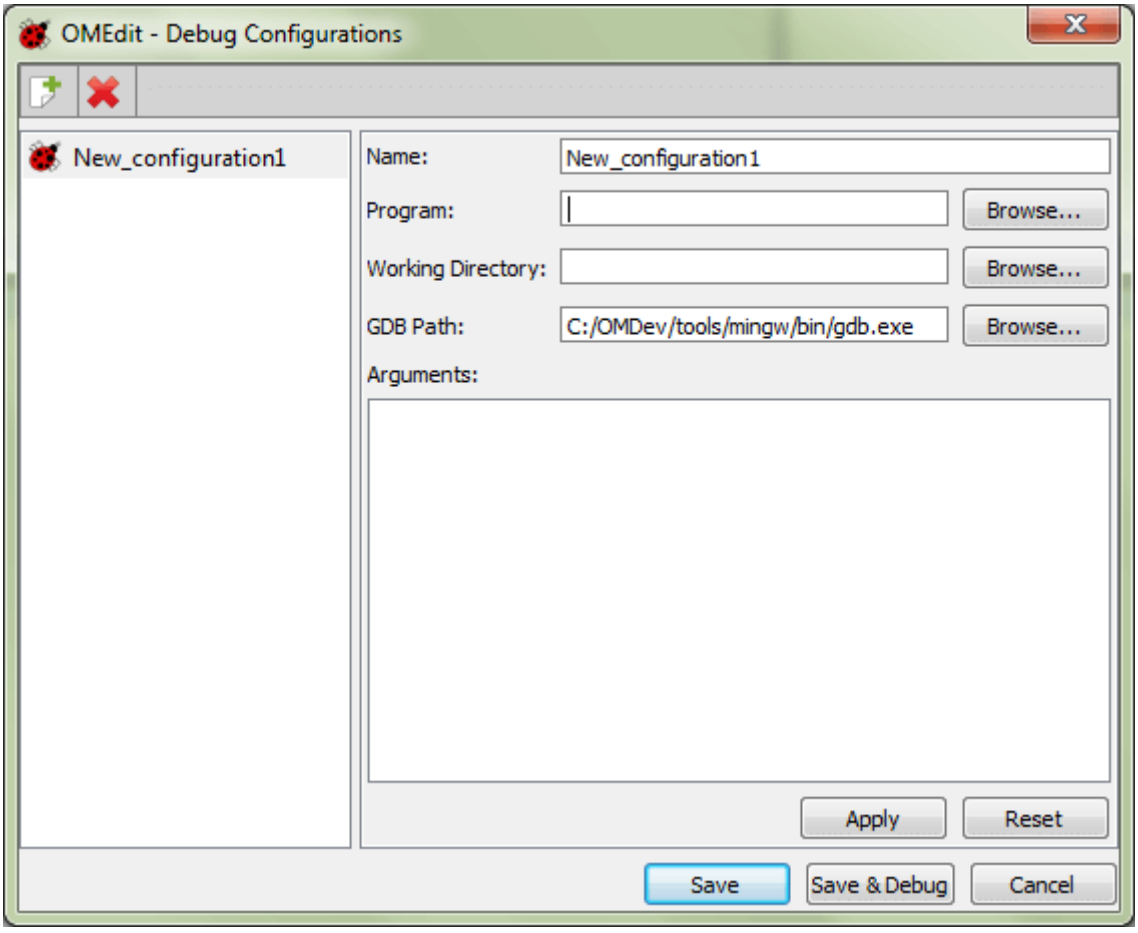


Figure 8.4: Debug Configurations.

8.2.4 Attach to Running Process

If you already have a running simulation executable with debugging symbols outside of OMEdit then you can use the Debug->Attach to Running Process option to attach the debugger with it. Figure 8.5 shows the Attach to Running Process dialog. The dialog shows the list of processes running on the machine. The user selects the program that he/she wish to debug. OMEdit debugger attaches to the process.

8.2.5 Using the Algorithmic Debugger Window

Figure 8.6 shows the Algorithmic Debugger window. The window contains the following browsers,

- *Stack Frames Browser* - shows the list of frames. It contains the program context buttons like resume, interrupt, exit, step over, step in, step return. It also contains a threads drop down which allows switching between different threads.

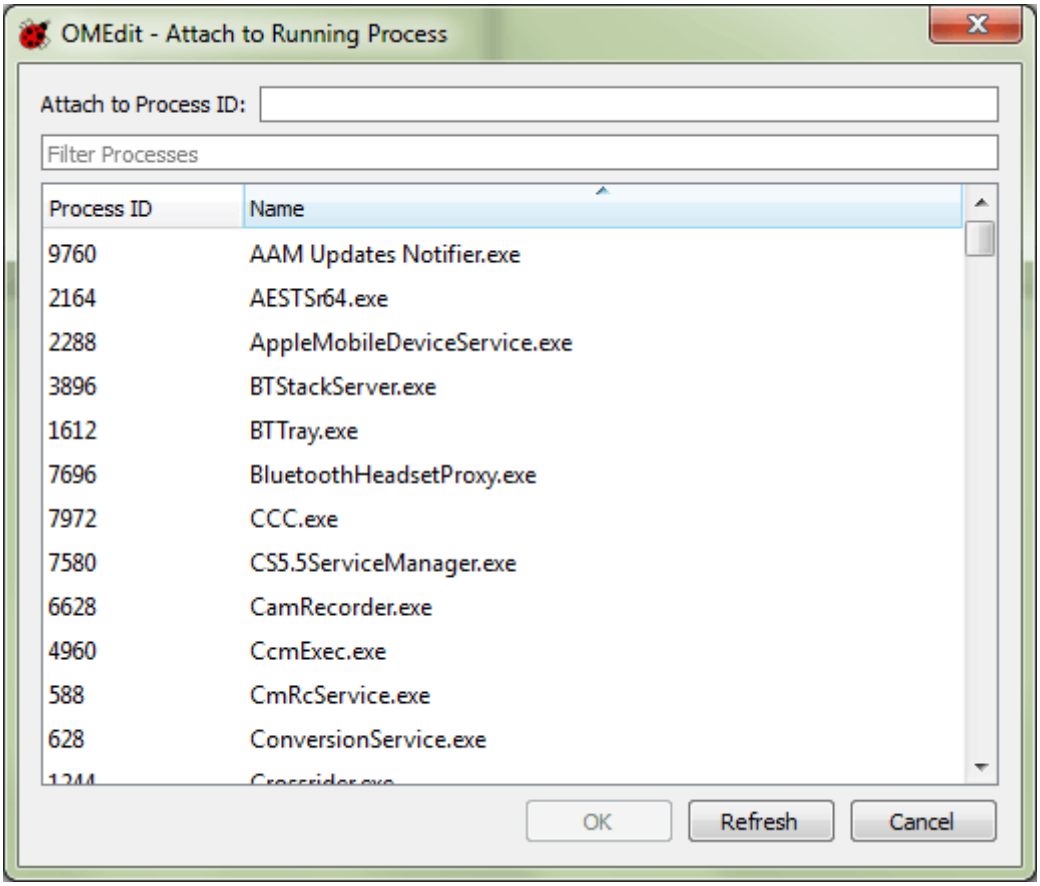


Figure 8.5: Attach to Running Process.

- *BreakPoints Browser* - shows the list of breakpoints. Allows adding/editing/removing breakpoints.
- *Locals Browser* - Shows the list of local variables with values. Select the variable and the value will be shown in the bottom right window. This is just for convenience because some variables might have long values.
- *Debugger CLI* - shows the commands sent to gdb and their responses. This is for advanced users who want to have more control of the debugger. It allows sending commands to gdb.
- *Output Browser* - shows the output of the debugged executable.

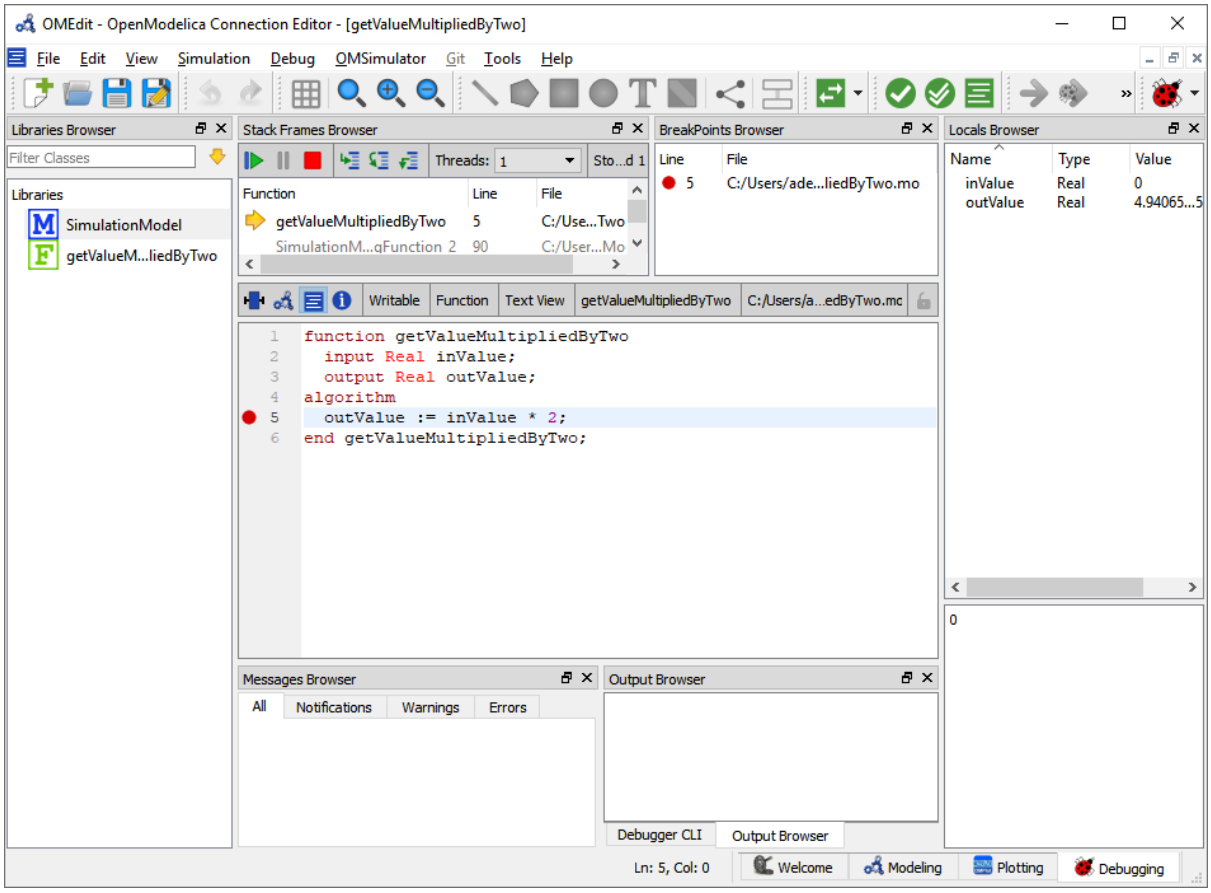


Figure 8.6: Algorithmic Debugger.

CHAPTER

NINE

FLATTENING MODELS TO BASEMODELICA

9.1 BaseModelica

BaseModelica is an intermediate format to describe hybrid (continuous and discrete) systems with emphasis on defining the dynamic behavior of systems, rather than their structure. It is meant to become part of the Modelica standard, as a subset of the Modelica language that does not include object-oriented features such as lookup, instantiation, inheritance, connections, but rather gives a flat representation of a Modelica model which only contains variable declarations, function declarations, record declarations, equations, and initial equations.

The main aim of BaseModelica is to provide a much lower barrier of entry to the Modelica world, since writing a BaseModelica compiler or interpreter will be a much easier task than writing a full-fledged Modelica compiler.

BaseModelica is currently described by the MCP 0031 draft , and will eventually be incorporated in a future version of the Modelica Language Specification.

9.2 Converting Modelica models in BaseModelica with OpenModelica

The OpenModelica compiler front-end can flatten virtually 100% of Modelica models that are fully compliant with the Modelica Language Specification, converting them into a BaseModelica output. This provides a unique opportunity for organizations that want to enter the Modelica ecosystem, as they can delegate the heavy-lifting of flattening a Modelica model to the OpenModelica compiler (OMC), developing tools that only need to be able to parse and compile (or interpret) BaseModelica input.

Assume you have a package `MyPackage` contained in a file `MyPackage.mo` and you want to get the BaseModelica flattened code of model `MyPackage.Examples.MyModel` in the MyModel.mo file. From the command line, this is accomplished by typing

```
omc --BaseModelica -i=MyPackage.Examples.MyModel MyPackage.mo > MyModel.mo
```

If the package `MyPackage` is installed with the Package Manager, you can type

```
omc --BaseModelica -i=MyPackage.Examples.MyModel MyPackage > MyModel.mo
```

If you want to use OMEdit for that, you can load `MyPackage`, go to *Tools* | *Options* | *Simulation*, add `--BaseModelica` to the *Additional Translation Flags* input field, open `MyModel` and click on the Instantiate Model button, to get the BaseModelica flattened model in a separate window. Don't forget to remove `--BaseModelica` from the simulation options when you are done, otherwise regular simulations will be broken.

9.3 Array-preserving BaseModelica output

The OMC front-end can flatten models without scalarizing them, i.e., keeping arrays of variables together as first-class citizens and keeping array equations together via for loops. This feature is essential to manage models with large arrays efficiently.

From the command line, you can get array-preserving BaseModelica flat output by adding some extra debug flags to the previous command line, e.g.,

```
omc --BaseModelica -d=nonfScalarize,arrayConnect,combineSubscripts,
evaluateAllParameters,vectorizeBindings -i=MyPackage.Examples.MyModel MyPackage >
MyModel.mo
```

or by adding them to the Additional Translation Flags option in OMEdit.

Last, but not least, if you have a model with a large number of instances of the same class with the same modifier structure, the OMC front-end can automatically collect them into a single array, which can then be flattened efficiently without scalarization. To get that, replace the debug flags of the previous command line with

```
-d=nonfScalarize,mergeComponents,combineSubscripts,evaluateAllParameters,
vectorizeBindings
```

In this case, you also get a `MyModel_merged_table.json` file in the working directory, which lists the correspondences between the original scalar component names and the elements of the automatically created arrays.

CHAPTER

TEN

PORTING MODELICA LIBRARIES TO OPENMODELICA

One of the goals of OpenModelica is to provide a full, no-compromise implementation of the latest version of the Modelica Language Specification, released by the non-profit Modelica Association. This means that a main requirement for a Modelica library to work in OpenModelica is to be fully compliant to the Language Specification.

Libraries and models developed with other Modelica tools may contain some code which is not valid according to the current language specification, but still accepted by that tool, e.g. to support legacy code of their customers. In order to use those libraries and models in OpenModelica, one needs to make sure that such code is replaced by a valid one. Note that getting rid of invalid Modelica code does not make the library *only* usable in OpenModelica; to the contrary, doing that is the best guarantee that the library will be usable *both* with the original tool used for development *and* with OpenModelica, as well as with any other present or future Modelica tool that follows the standard strictly.

The first recommendation is to use any flag or option of the tool that was originally used to develop the library, that allows to check for strict compliance to the language specification. For example, Dymola features a translation option 'Pedantic mode for checking Modelica semantics' that issues an error if non-standard constructs are used.

For your convenience, here you can find a list of commonly reported issues.

10.1 Mapping of the library on the file system

Packages can be mapped onto individual *.mo* files or onto hierarchical directory structures on the file system, according to the rules set forth in Section 13.4. of the language specification. The file encoding must be UTF-8; the use of a BOM at the beginning of the file is deprecated and preferably avoided. If there are non-ASCII characters in the comments or in the documentation of your library, make sure that the file is encoded as UTF-8.

If a directory-based representation is chosen, each *.mo* file must start with a *within* clause, and each directory should contain a *package.order* file that lists all the classes and constants defined as separate files in that directory.

When using revision control systems such as Git, if the library is stored in a directory structure, it is recommended to include the top-level directory (that must have the same name as the top-level package) in the repository itself, to avoid problems in case the repository is cloned locally on a directory that doesn't have the right name.

The top-level directory name, or the single *.mo* file containing the entire package, should be named exactly as the package (e.g. *Modelica*), possibly followed by a space and by the version number (e.g. *Modelica 3.2.3*).

10.2 Modifiers for arrays

According to the rules set forth in Section 7.2.5 of the language specification, when instantiating arrays of components, modifier values should be arrays of the same size of the component array, unless the *each* prefix is introduced, in which case the scalar modifier values is applied to all the elements of the array. Thus, if *MyComponent* has a Real parameter *p*, these are all valid declarations

```
parameter Real q = {0, 1, 2};
MyComponent ma[3](p = {10, 20, 30});
MyComponent mb[3](p = q);
MyComponent mb[3](each p = 10);
```

while these are not

```
parameter Real r = 4;
MyComponent ma[3](p = r);
MyComponent mb[3](p = 20);
```

In most cases, the problem is solved by simply adding the *each* keyword where appropriate.

10.3 Access to conditional components

Up to Modelica 3.6, according to Section 4.4.5 of the language specification, "A component declared with a condition-attribute can only be modified and/or used in connections". This required to use the following, slightly convoluted patterns when dealing, e.g., with conditional input connectors, making use of internal auxiliary variables or connectors:

```
model M
  parameter Boolean activateIn1 = true;
  parameter Boolean activateIn2 = true;
  Modelica.Blocks.Interfaces.RealInput u1_in if activateIn1;
  Modelica.Blocks.Interfaces.RealInput u2_in = u2 if activateIn2 "binding equation␣
↪only holds if activateIn2 = true";
  Real u2 "internal variable corresponding to u2_in";
  Real y;
protected
  Modelica.Blocks.Interfaces.RealInput u1 "internal connector corresponding to u1_in";
equation
  y = u1 + u2;
  connect(u1_in, u1) "automatically disabled if u1_in is deactivated";
  if not activateIn1 then
    u1 = 0 "default value for protected connector value when u1_in is disabled";
  end if;
  if not activateIn2 then
    u2 = 0 "default value for u2 when u2_in is disabled";
  end if;
end M;
```

where conditional components 'u1_in' and 'u2_in' were only used in connect equations. Other Modelica tools accepted code that was more straightforward and easier to understand, but actually not compliant with this restriction, causing compatibility issues, e.g.:

```
model M
  parameter Boolean activateIn1 = true;
  parameter Boolean activateIn2 = true;
  Modelica.Blocks.Interfaces.RealInput u1_in if activateIn1;
  Modelica.Blocks.Interfaces.RealInput u2_in if activateIn2;
  Real u1;
  Real u2;
  Real y;
equation
  y = u1 + u2;
  if activateIn1 then
    u1 = u1_in;
  else
    u1 = 0 "default value for protected connector value when u1_in is disabled";
  end if;
  if activateIn2 then
    u2 = u2_in;
```

(continues on next page)

(continued from previous page)

```
  else
    u2 = 0 "default value for protected connector value when u2_in is disabled";
  end if;
end M;
```

Although this restriction makes identifying structurally inconsistent models easier, it requires to write code that can be pretty obscure when handling use cases that would be very straightforward to handle otherwise. Hence, this restriction will be removed in Modelica 3.7. In particular, the current draft of Modelica 3.7 explicitly states that if the Boolean expression activating the component is true, there are no restrictions on the use of such component. Since version 1.26.0, OpenModelica complies with these new relaxed rules.

10.4 Access to classes defined in partial packages

Consider the following example package

```
package TestPartialPackage
  partial package PartialPackage
    function f
      input Real x;
      output Real y;
    algorithm
      y := 2*x;
    end f;
  end PartialPackage;

  package RegularPackage
    extends PartialPackage;
    model A
      Real x = time;
    end A;
  end RegularPackage;

  model M1
    package P = PartialPackage;
    Real x = P.f(time);
  end M1;

  model M2
    extends M1(redeclare package P = RegularPackage);
  end M2;

  model M3
    encapsulated package LocalPackage
      import TestPartialPackage.PartialPackage;
      extends PartialPackage;
    end LocalPackage;
    package P = LocalPackage;
    Real x = P.f(time);
  end M3;
end TestPartialPackage;
```

Model *M1* references a class (a function, in this case) from a partial package. This is perfectly fine if one wants to write a generic model, which is then specialized by redeclaring the package to a non-partial one, as in *M2*. However, *M1* cannot be compiled for simulation, since, according to Section 5.3.2 of the language specification, the classes that are looked inside during lookup shall not be partial in a simulation model.

This problem can be fixed by accessing that class (the function *f*, in this case) from a non-final package that extends

the partial one, either by redeclaring the partial package to a non-partial one, as in *M2*, or by locally defining a non-partial package that extends from the partial one, as in *M3*. The latter option is of course viable only if the class being accessed is in itself not a partial or somehow incomplete one.

This issue is often encountered in models using *Modelica.Media*, that sometimes use some class definitions (e.g. unit types) from partial packages such as *Modelica.Media.Interfaces.PartialMedium*. The fix in most cases is just to use the same definition from the actual replaceable *Medium* package defined in the model, which will eventually be redeclared to a non-partial one in the simulation model.

10.5 Equality operator in algorithms

The following code is illegal, because it uses the equality '=' operator, which is reserved for equations, instead of the assignment operator ':=' inside an algorithm:

```
function f
  input Real x;
  input Real y = 0;
  output Real z;
algorithm
  z = x + y;
end f;
```

so, the OpenModelica parser does not accept it. The correct code is:

```
function f
  input Real x;
  input Real y = 0;
  output Real z;
algorithm
  z := x + y;
end f;
```

Some tools automatically and silently apply the correction to the code, please save it in its correct form to make it usable with OpenModelica.

Also note that binding *equations* with '=' sign are instead required for default values of function inputs.

10.6 Public non-input non-output variables in functions

According to Section 12.2 of the language specification, only input and output formal parameters are allowed in the function's public variable section. Hence, the following function declaration is not valid:

```
function f
  input Real x;
  output Real y;
  Real z;
algorithm
  z := 2;
  y := x+z;
end f;
```

and should be fixed by putting the variable *z* in the protected section:

```
function f
  input Real x;
  output Real y;
protected
  Real z;
```

(continues on next page)

(continued from previous page)

```
algorithm
  z := 2;
  y := x+z;
end f;
```

10.7 Subscripting of expressions

Some libraries use expression subscripting, e.g.

```
model M
  Real x[3];
  Real y[3];
  Real z;
equation
  z = (x.*y)[2];
  ...
end M;
```

This construct is already accepted by some Modelica tools, but is not yet included in the current Modelica Specification 3.6, so it is not supported in OpenModelica up to version 1.22.0. It has now been included in the draft for the 3.7 language specification, so it will be implemented in the future also by OpenModelica.

10.8 Incomplete specification of initial conditions

The simulation of Modelica models of dynamical systems requires the tool to determine a consistent initial solution for the simulation to start. To do so, the system equations are augmented by adding one initial condition for each continuous state variable (after index reduction) and one initial condition for each discrete state variable. Then, the augmented system is solved upon initialization.

These initial conditions can be formulated by adding a *start* = <*expression*> and a *fixed* = *true* attribute to those variables, e.g.

```
parameter Real x_start = 10;
parameter Real v_start = 2.5;
Real x(start = x_start, fixed = true);
discrete Real v(start = v_start, fixed = true);
Integer i(start = 2, fixed = true);
```

or by adding initial equations, e.g.:

```
  parameter Real x_start = 10;
  parameter Real v_start = 2.5;
  Real x;
  discrete Real v;
  Integer i;
  Real y(start = 3.5);
initial equation
  x = x_start;
  v = v_start;
  i = 2;
  der(y) = 0;
```

Note that in the latter case, the start attribute on *y* is not used directly to set the initial value of that variable, but only potentially used as initial guess for the solution of the initialization problem, that may require using an iterative nonlinear solver. Also note that sets of initial equations are often added to the models taken from reusable component libraries by selecting certain component parameters, such as *initOpt* or similar.

If the number of initial conditions matches the number of continuous and discrete states, then the initialization problem is well-defined. Although this is per se not a guarantee that all tools will be able to solve it and find the same solution, this is for sure a prerequisite for across-tool portability.

Conversely, if the number of initial conditions is less than the number of states, the tool has to add some initial equations, using some heuristics to change the fixed attribute of some variables from false to true. Consider for example the following model:

```
model M
  Real x;
  Real y(start = 1);
  Real z(start = 2);
equation
  der(x) = y + z;
  y = 2*x;
  z = 10*x + 1;
end M;
```

This model has one state variable x, no variables with *fixed* = *true* attributes and no initial equation, so there is one missing initial condition. One tool could choose to add the *fixed* = *true* attribute to the state variable x, fixing it to the default value of zero of its *start* attribute. Or, it could decide to give more priority to variables that have an explicitly modified *start* attribute, hence fix the initial value of y to 1, or the initial value of z to 2. Three completely different simulations would ensue.

The Modelica Language Specification, Section 8.6 does not prescribe or recommend any specific choice criterion in this case. Hence, different tools, or even different versions of the same tool, could add different initial conditions, leading to completely different simulations. In order to avoid any ambiguity and achieve good portability, it is thus recommended to make sure that the initial conditions of all simulation model are well-specified.

A model with not enough initial conditions causes the OMC to issue the following translation warning: "The initial conditions are not fully specified". By activating the Tools | Options | Simulation | Show additional information from the initialization process option, or the *-d=initialization* compiler flag, one can get an explicit list of the additional equations that OpenModelica automatically adds to get a fully specified initialization problem, which may be helpful to figure out which initial conditions are missing. In this case, we recommend to amend the source code of the model by adding suitable extra initial conditions, until that warning message no longer appears.

10.9 Modelica_LinearSystems2 Library

The Modelica_LinearSystem2 library was originally developed in Dymola with a plan of eventually making it part of the Modelica Standard Library (thus the underscore in the library name). The library is based on several functions, e.g. *readStringMatrix()*, *simulateModel()*, *linearizeModel()* that are built-in Dymola functions but are not part of the Modelica Standard Library.

In principle, these functions could be standardized and become part of the ModelicaServices library, which collects standardized interfaces to tool-specific functionality; then, OpenModelica could easily implement them based on its internal functionality. However, until this effort is undertaken, the Modelica_LinearSystem2 library cannot be considered as a full-fledged Modelica library, but only a Dymola-specific one.

If you are interested in using this library in OpenModelica and are willing to contribute to get it supported, please contact the development team, e.g. by opening an ticket on the issue tracker.

CHAPTER

ELEVEN

BACKWARDS COMPATIBILITY OF OPENMODELICA RELEASED VERSIONS

The development of OpenModelica is guided by two basic principles:

1. Follow the Modelica Language Specification (MLS) as strictly as possible.
2. Preserve backwards compatibility as much as possible.

The Open Source Modelica Consortium strongly believes in open standards, as enablers of strong and healthy eco-systems involving software tool developers and software tool users. Strict compliance with the MLS is the key factor to enable portability of models among different Modelica tools, which gives a huge added value to the Modelica community, compared to other communities of users of software using proprietary modelling languages. Compliance to the standard not only gives you the freedom to switch from one tool to another, avoiding vendor lock-in and protecting the value of your investment in modelling in the long term, but also allows you to use different tools simultaneously on the same Modelica code for different purposes, e.g., simulation, generation of FMUs, sensitivity analysis, parameter optimization, optimal control, etc. Hence, the development of OpenModelica development strives to implement the MLS standard as strictly as possible.

The Open Source Modelica Consortium also recognizes the value of the code developed by the users of OpenModelica. There is hardly anything more annoying than not being able to use legacy code with up-to-date versions of a software tool. Hence, the Consortium strives to keep newer versions of OpenModelica fully backwards compatible with older ones and is committed to releasing patch versions of the latest x.y.0 release of the software, in case regressions from previously released versions are reported on the OpenModelica issue tracker.

Given this commitment, we strongly recommend OpenModelica users to always use the latest released version of the software, to benefit from bug fixes, performance improvements and new added features. If you find any backwards-compatibility issue with new released versions, we strongly encourage you to report them on the issue tracker; chances are that it can get fixed in the nightly build in a short time, if possible.

Given the first stated principle, compliance to the MLS, there is one exception to the rule of keeping backwards compatibility: if we find that OpenModelica is not following the MSL for some reason, we try to fix it as soon as possible. This means that a model or library developed with older versions of OpenModelica may not work with more recently released version of the software *because the Modelica code was invalid according to the MLS*, but the older version of OpenModelica accepted it anyway. You can check ticket #10386 for one such example.

In these cases, one may be tempted to stick indefinitely to the latest version of OpenModelica that handled the invalid Modelica code successfully. Although you are of course free to do so, we do not recommend this policy because you are going to miss all the improvements to the OpenModelica software in the future. More importantly, if you then discover bugs that prevent you from using your Modelica code in new situations, we can't help you in any way, because you are locked to an old version for which we cannot provide maintenance support.

The ideal solution to handle these cases is to update the Modelica source code of the models to make it fully compliant with the MLS. This ensures maximum portability and long-term support of your Modelica code.

In case this is not possible for some reason, e.g. lack of time and resources, or the fact that the legacy code belongs to Modelica libraries you did not develop yourself, we provide a way to cope with non-standard Modelica code in newer version of OpenModelica: the --allowNonStandardModelica compiler flag allows to disable some Modelica compatibility checks and continue using your legacy code with newer versions of the compiler. This flag can be set in OMEdit in the *Tools* | *Options* | *Simulation* | *Additional Translation Flags*.

CHAPTER

TWELVE

GENERATING GRAPH REPRESENTATIONS FOR MODELS

The system of equations after symbolic transformation is represented by a graph. OpenModelica can generate graph representations which can be displayed in the graph tool *yed* (http://www.yworks.com/products/yed). The graph generation is activated with the debug flag

+d=graphml

Two different graphml- files are generated in the working directory. *TaskGraph_model.graphml*, showing the strongly-connected components of the model and *BipartiteGraph_CompleteDAE_model.graphml* showing all variables and equations. When loading the graphs with *yEd*, all nodes are in one place. Please use the various layout algorithms to get a better overview.

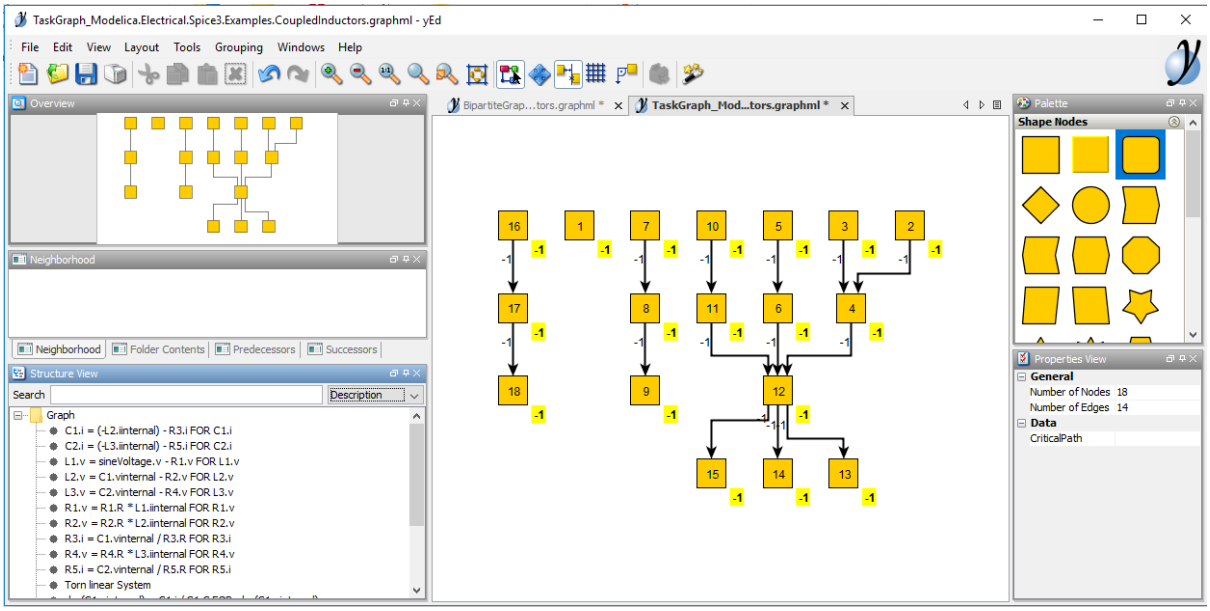


Figure 12.1: A task-graph representation of a model in yEd

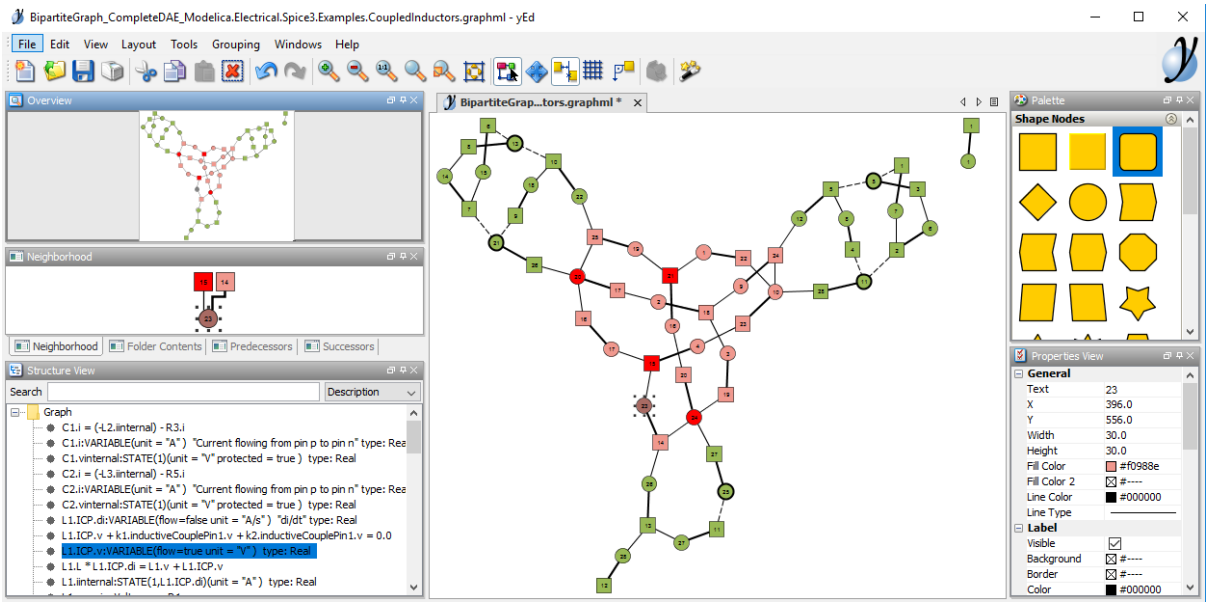


Figure 12.2: A biparite graph representation of a model in yEd

CHAPTER

THIRTEEN

FUNCTIONAL MOCK-UP INTERFACE - FMI

The Functional Mock-up Interface (FMI) Standard for model exchange and co-simulation allows export, exchange and import of pre-compiled models between different tools. The FMI standard is Modelica independent, so import and export works both between different Modelica or non-Modelica tools.

See also *OMSimulator documentation*.

13.1 FMI Standards

Table 13.1: FMI standard versions

Version	Support in OpenModelica
1.0.1	Deprecated
2.0.5	Supported
3.0.2	Experimental

13.1.1 Layered Standards

Layered standards extend FMI 3.0 with optional, separately versioned specifications.

Table 13.2: FMI 3.0 layered standards

Layered standard	Support in OpenModelica
FMI-LS-BUS	Not planned
FMI-LS-XCP	Not planned
FMI-LS-STRUCT	Planned
FMI-LS-REF	Planned
FMI-LS-DAE	Planned (demonstrator in progress). Export and import follow the `1.0.0-alpha.1` draft as of its commit `78313f4` (2026-09-02); the revision is also reported as a notification when an FMU is exported.

13.1.2 Supported Capability Flags

The optional capabilities advertised in the `modelDescription.xml` of an exported FMU, per FMI version and interface type (Model Exchange, Co-Simulation and, for FMI 3.0, Scheduled Execution).

Table 13.3: Exported capability flags

Capability	2.0 ME	2.0 CS	3.0 ME	3.0 CS	3.0 SE
Get and set FMU state	yes	yes	yes	yes	yes
Serialize FMU state	no	no	yes	yes	yes
Directional derivatives	cond	exp	cond	cond	cond
Adjoint derivatives	—	—	wip	wip	wip
Per-element dependencies	—	—	no	no	no
Variable communication step size	—	yes	—	yes	—
Interpolate inputs	—	yes	—	—	—
Max output derivative order	—	1	—	1	—
Event mode	—	—	—	yes	—
Intermediate update	—	—	—	no	—

Legend:

- `yes` / `no` — the capability flag is exported as `true` / `false`.
- `cond` — exported as `true` only when a symbolic directional-derivative Jacobian is available, i.e. not disabled via *-d=disableDirectionalDerivatives*.
- `exp` — only enabled with the experimental flag `-d=fmuExperimental` (`false` by default). In FMI 3.0 these state features are always enabled.
- `wip` — in active development; currently exported as `false`.
- `-` — not applicable to this interface type or FMI version.

13.2 FMI Export

To export a FMU use the OpenModelica command *buildModelFMU()* from the command line interface, OMShell, OMNotebook or MDT. The FMU export command is also integrated in OMEdit. Select *File* > *Export* > *FMU*. Or alternatively, right click a model to obtain the export command. The FMU package is generated in the current working directory of OMC or the directory set in *OMEdit* > *Options* > *FMI* > *Move FMU*. You can use the *cd()* command to see the current location. The location of the generated FMU is printed in the Messages Browser of OMEdit or on the command line.

You can set which version of FMI to export through OMEdit settings, see section *FMI Options*.

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

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.
↪mo")
true
>>> buildModelFMU(BouncingBall)
"«DOCHOME»/BouncingBall.fmu"
```

Note

Notification: Model statistics after passing the front-end and creating the data structures used by the back-end:

* Number of equations: 6

* Number of variables: 6

Notification: Model statistics after passing the back-end for initialization:

* Number of independent subsystems: 3

* Number of states: 0 ()

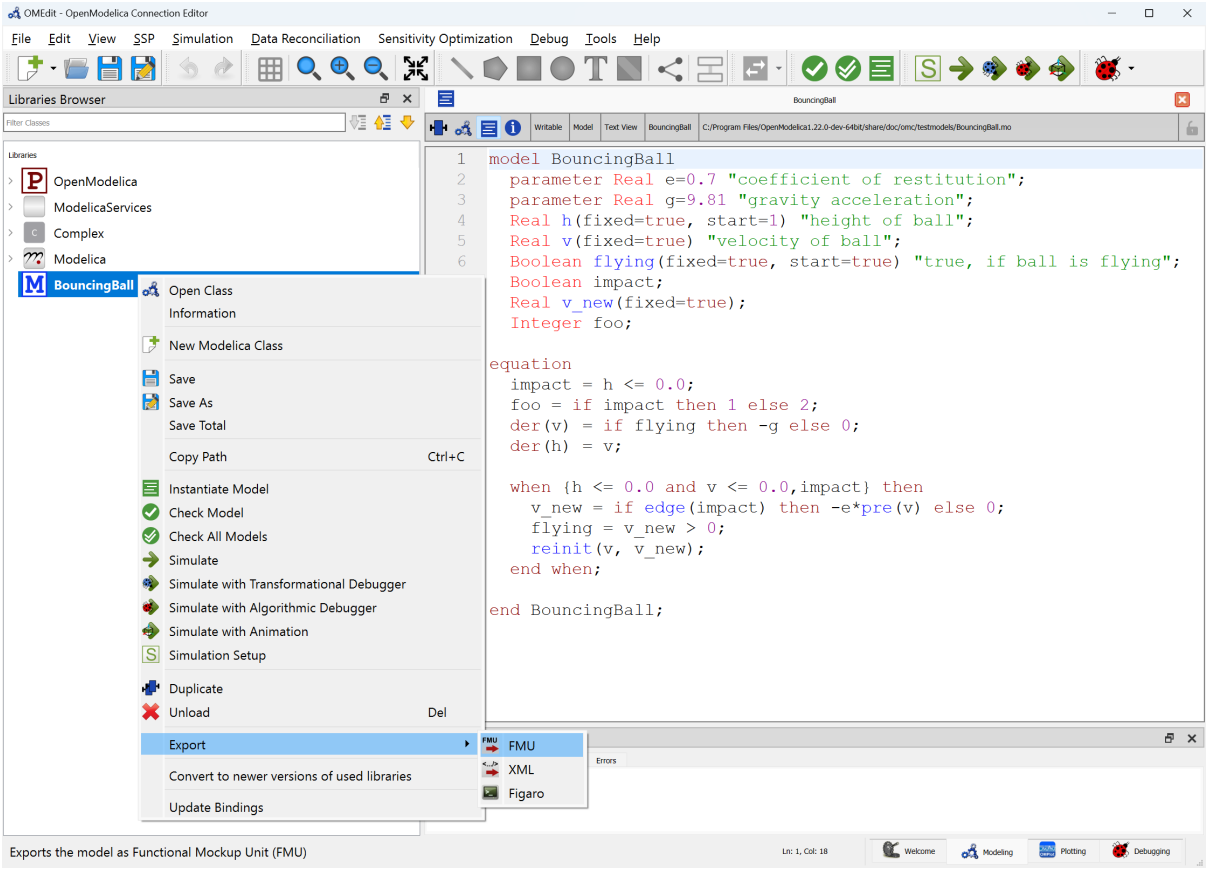


Figure 13.1: FMI Export.

```
* Number of discrete variables: 9 (v_new,$PRE.v_new,flying,$PRE.flying,impact,foo,$whenCondition1,$whenCondition2,$whenC
* Number of discrete states: 0 ()
* Number of clocked states: 0 ()
* Top-level inputs: 0
Notification: Strong component statistics for initialization (13):
* Single equations (assignments): 13
* Array equations: 0
* Algorithm blocks: 0
* Record equations: 0
* When equations: 0
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0
Notification: Model statistics after passing the back-end for simulation:
* Number of independent subsystems: 1
* Number of states: 2 (v,h)
* Number of discrete variables: 7 ($whenCondition3,$whenCondition2,$whenCondition1,foo,v_new,impact,flying)
```

```
* Number of discrete states: 2 (impact,v)
* Number of clocked states: 0 ()
* Top-level inputs: 0
Notification: Strong component statistics for simulation (9):
* Single equations (assignments): 7
* Array equations: 0
* Algorithm blocks: 0
* Record equations: 0
* When equations: 2
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0
Notification: Building FMU for platform 'static' (1/1).
Notification: Finished FMU for platform 'static' (1/1).
```

After the command execution is complete you will see that a file BouncingBall.fmu has been created. Its contents varies depending on the target platform. On the machine generating this documentation the contents in Listing 13.1 are generated (along with the C source code).

Listing 13.1: BouncingBall FMU contents

```
binaries/
binaries/linux64/
binaries/linux64/BouncingBall.so
modelDescription.xml
```

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

By default an FMU that can be used for both Model Exchange and Co-Simulation is generated. We support FMI 1.0 (deprecated) and FMI 2.0 for Model Exchange and Co-Simulation FMUs, with experimental support for FMI 3.0 (see the version table above).

For the Co-Simulation FMU two integrator methods are available:

- Forward Euler [default]
- SUNDIALS CVODE (see[1])

Forward Euler uses root finding, which does an Euler step of `communicationStepSize` in `fmi2DoStep`. Events are checked for before and after the call to `fmi2GetDerivatives`.

If CVODE is chosen as integrator the FMU should also include runtime dependencies (*--fmuRuntimeDepends=modelica*) to copy all used dynamic libraries into the generated FMU to make it exchangeable.

To export a Co-Simulation FMU with CVODE for the bouncing ball example use the following commands:

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

(continues on next page)

[1] Sundials Webpage

(continued from previous page)

```
>>> setCommandLineOptions("--fmiFlags=s:cvode")
true
>>> buildModelFMU(BouncingBall, version = "2.0", fmuType="cs")
"«DOCHOME»/BouncingBall.fmu"
```

Note

Notification: Model statistics after passing the front-end and creating the data structures used by the back-end:

* Number of equations: 6

* Number of variables: 6

Notification: Model statistics after passing the back-end for initialization:

* Number of independent subsystems: 3

* Number of states: 0 ()

* Number of discrete variables: 9 (v_new,$PRE.v_new,flying,$PRE.flying,impact,foo,$whenCondition1,$whenCondition2,$whenC

* Number of discrete states: 0 ()

* Number of clocked states: 0 ()

* Top-level inputs: 0

Notification: Strong component statistics for initialization (13):

* Single equations (assignments): 13

* Array equations: 0

* Algorithm blocks: 0

* Record equations: 0

* When equations: 0

* If-equations: 0

* Equation systems (not torn): 0

* Torn equation systems: 0

* Mixed (continuous/discrete) equation systems: 0

Notification: Model statistics after passing the back-end for simulation:

* Number of independent subsystems: 1

* Number of states: 2 (v,h)

* Number of discrete variables: 7 ($whenCondition3,$whenCondition2,$whenCondition1,foo,v_new,impact,flying)

* Number of discrete states: 2 (impact,v)

* Number of clocked states: 0 ()

* Top-level inputs: 0

Notification: Strong component statistics for simulation (9):

* Single equations (assignments): 7

* Array equations: 0

* Algorithm blocks: 0

* Record equations: 0

```
* When equations: 2
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0
Notification: Building FMU for platform 'static' (1/1).
Notification: Finished FMU for platform 'static' (1/1).
```

The FMU BouncingBall.fmu will have a new file BouncingBall_flags.json in its resources directory. By manually changing its content users can change the solver method without recompiling the FMU.

The BouncingBall_flags.json for this example is displayed in Listing 13.2.

Listing 13.2: BouncingBall FMI flags

```
{
  "s" : "cvode"
}
```

13.2.1 Compilation Process

OpenModelica can export FMUs that are compiled with CMake (default) or Makefiles. CMake version v3.21 or newer is recommended, minimum CMake version is v3.5.

The Makefile FMU export will be removed in a future version of OpenModelica. Set compiler flag *--fmuCMakeBuild=false* to use the Makefiles export.

The FMU contains a CMakeLists.txt file in the sources directory that can be used to re-compile the FMU for a different host and is also used to cross compile for different platforms.

The CMake compilation accepts the following settings:

- `BUILD_SHARED_LIBS`: Boolean value to switch between dynamic and statically linked binaries.
  - `ON` (default): Compile DLL/Shared Object binary object.
  - `OFF`: Compile static binary object.
- `FMI_INTERFACE_HEADER_FILES_DIRECTORY`: String value specifying path to FMI header files containing `fmi2Functions.h`, `fmi2FunctionTypes.h` and `fmi2TypesPlatforms.h`.
  - Defaults to a location inside the OpenModelica installation directory, which was used to create the FMU. They need to be version 2.0.4 from the FMI Standard.
- `RUNTIME_DEPENDENCIES_LEVEL`: String value to specify runtime dependencies set.
  - `none`: Adds no runtime dependencies to FMU. The FMU can't be used on a system if it doesn't provided all needed dependencies.
  - `modelica` (default): Add Modelica runtime dependencies to FMU, e.g. a external C library used from a Modelica function. Needs CMake version v3.21 or newer.
  - `all`: Add system and Modelica runtime dependencies. Needs CMake version v3.21 or newer.

  CMake install TARGETS RUNTIME_DEPENDENCIES is not supported when cross compiling. When cross compiling the FMU CMake project therefore falls back to installing the external libraries it linked against directly, which covers the Modelica libraries a model depends on but not the system libraries of the target platform.
- `NEED_CVODE`: Boolean value to integrate CVODE integrator into CoSimulation FMU.

  - `ON`: Link to SUNDIALS CVODE. If CVODE is not in a default location `CVODE_DIRECTORY` needs to be set. Its also recommended to use `RUNTIME_DEPENDENCIES_LEVEL=modelica` or higher to add SUNDIALS runtime dependencies into the FMU.
  - `OFF` (default): Don't link to SUNDIALS CVODE.
- `CVODE_DIRECTORY`: String value with location of libraries `sundials_cvode` and `sundials_nvecserial` with SUNDIALS version 5.4.0.
  - Defaults to a location inside the OpenModelica installation directory, which was used to create the FMU.

Then use CMake to configure, build and install the FMU. To repack the FMU after installation use custom target `create_fmu`.

For example to re-compile the FMU with cmake and runtime dependencies use:

```
$ unzip BouncingBall.fmu -d BouncingBall_FMU
$ cd BouncingBall_FMU/sources
$ cmake -S . -B build_cmake \
  -D RUNTIME_DEPENDENCIES_LEVEL=modelica \
  -D CMAKE_C_COMPILER=clang -D CMAKE_CXX_COMPILER=clang++
$ cmake --build build_cmake --target create_fmu --parallel
```

13.2.2 Platforms

The `platforms` setting specifies for what target system the FMU is compiled:

- Empty: Create a Source-Code-only FMU.
- `native`: Create a FMU compiled for the exporting system.
- `<cpu>-<vendor>-<os>` host triple: OpenModelica searches for programs in PATH matching pattern `<cpu>-<vendor>-<os>cc` to compile. E.g. `x86_64-linux-gnu` for a 64 bit Linux OS or `i686-w64-mingw32` for a 32 bit Windows OS using MINGW.
- `<cpu>-<vendor>-<os> docker run ghcr.io/openmodelica/crossbuild:v1.27.0` Host triple with Docker image provided by OpenModelica: OpenModelica will use Docker image ghcr.io/openmodelica/crossbuild:v1.27.0 to cross compile. The image provides compiler toolchain files to cross compile with CMake for the following host triples:
  - `i686-linux-gnu`
  - `x86_64-linux-gnu`
  - `aarch64-linux-gnu`
  - `arm-linux-gnueabi`
  - `arm-linux-gnueabihf`
  - `i686-w64-mingw32`
  - `x86_64-w64-mingw32`

  OpenModelica will add a matching `CMAKE_TOOLCHAIN_FILE` to the compilation process.

  If your model depends on external C libraries the library has to be available for the target platform, e.g. in `Resources/Library/win64` when compiling for `x86_64-w64-mingw32`. OpenModelica searches the `Resources/Library` sub directory matching the target platform and copies the library into the FMU next to the model binary. If no binary for the target platform is shipped with the Modelica library, providing a pre-compiled static library can be necessary.
- `<cpu>-<vendor>-<os> docker run <image>` Host triple with Docker image: OpenModelica will use the specified Docker image to cross compile for given host triple. Because privilege escalation is very easy to achieve with Docker OMEdit adds `--pull=never` to the Docker calls for the `multiarch/crossbuild` images. Only use this option if you understand the security risks associated with Docker images from

unknown sources. E.g. `x86_64-linux-gnu docker run --pull=never multiarch/crossbuild` to cross compile for a 64 bit Linux OS. Because system libraries can be different for different versions of the same operating system, it is advised to use *--fmuRuntimeDepends=all*.

13.2.3 Cross Compilation

Cross compilation can be done by using platform `<cpu>-<vendor>-<os> docker run ghcr.io/openmodelica/crossbuild:v1.27.0` or done manually. Both can be difficult at times.

To cross compile with CMake provide a toolchain file specifying the target system and where to find the compiler toolchain for the target system.

For example the Docker image ghcr.io/openmodelica/crossbuild:v1.27.0 provided by OpenModelica is based on Linux (Ubuntu 24.04 at the time of writing) and has toolchains installed to cross compile together with matching toolchain files.

Manual Cross Compilation

Let's have a look at example model `BouncingBall`:

First generate C sources for the FMU:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.
↪mo")
True
>>> buildModelFMU(BouncingBall, platforms = {""})

"Notification: Model statistics after passing the front-end and creating the data␣
↪structures used by the back-end:
 * Number of equations: 6
 * Number of variables: 6
Notification: Model statistics after passing the back-end for initialization:
 * Number of independent subsystems: 3
 * Number of states: 0 ()
 * Number of discrete variables: 9 (v_new,$PRE.v_new,flying,$PRE.flying,impact,foo,
↪$whenCondition1,$whenCondition2,$whenCondition3)
 * Number of discrete states: 0 ()
 * Number of clocked states: 0 ()
 * Top-level inputs: 0
Notification: Strong component statistics for initialization (13):
 * Single equations (assignments): 13
 * Array equations: 0
 * Algorithm blocks: 0
 * Record equations: 0
 * When equations: 0
 * If-equations: 0
 * Equation systems (not torn): 0
 * Torn equation systems: 0
 * Mixed (continuous/discrete) equation systems: 0
Notification: Model statistics after passing the back-end for simulation:
 * Number of independent subsystems: 1
 * Number of states: 2 (v,h)
 * Number of discrete variables: 7 ($whenCondition3,$whenCondition2,$whenCondition1,
↪foo,v_new,impact,flying)
 * Number of discrete states: 2 (impact,v)
 * Number of clocked states: 0 ()
 * Top-level inputs: 0
Notification: Strong component statistics for simulation (9):
 * Single equations (assignments): 7
 * Array equations: 0
```

(continues on next page)

(continued from previous page)

```
 * Algorithm blocks: 0
 * Record equations: 0
 * When equations: 2
 * If-equations: 0
 * Equation systems (not torn): 0
 * Torn equation systems: 0
 * Mixed (continuous/discrete) equation systems: 0"
```

OpenModelica won't zip the FMU in this case. If you already have an existing FMU unzip it into some directory `<Model>.fmutmp`. Optionally delete existing binaries like `<Model>.fmutmp/binaries/linux64/`.

Then cross compile the sources with a suitable toolchain file.

```
# Optional: Work inside interactive Docker container
docker run --rm -it \
  -v $PWD:/fmu \
  -v $OPENMODELICAHOME/include/omc/FMI2:/fmiInclude \
  -w/fmu \
  ghcr.io/openmodelica/crossbuild:v1.27.0 bash

cd <Model>.fmutmp/sources
cmake -S . -B build \
  -DCMAKE_TOOLCHAIN_FILE=/opt/cmake/toolchain/x86_64-w64-mingw32.cmake \
  -DRUNTIME_DEPENDENCIES_LEVEL=none \
  -DFMI_INTERFACE_HEADER_FILES_DIRECTORY=/fmiInclude \
  -G "Unix Makefiles"
cmake --build build --parallel --target install
```

Now the FMU should contain `binaries/win64/BouncingBall.dll`. Compile additional binaries in the same way and when done zip the FMU by running

```
cmake --build build --parallel --target create_fmu
```

Now there should be a FMU with `win64` binaries `BouncingBall-fmu`.

13.3 FMI Import - SSP

If you want to simulate a single, stand-alone FMU, or possibly a connection of several FMUs, the recommended tool to do that is OMSimulator, see the *OMSimulator documentation* and *Graphical Modelling* for further information.

13.3.1 FMI Import - Non-Standard Modelica Model

FMI Import allows to use an FMU, generated according to the FMI for Model Exchange 2.0 standard, as a component in a Modelica model. This can be useful if the FMU describes the behavior of a component or sub-system in a structured Modelica model, which is not easily turned into a pure FMI-based model that can be handled by OMSimulator.

FMI is a computational description of a dynamic model, while a Modelica model is a declarative description; this means that not all conceivable FMUs can be successfully imported as Modelica models. Also, the current implementation of FMU import in OpenModelica is still somewhat experimental and not guaranteed to work in all cases. However, if the FMU-ME you want to import was exported from a Modelica model and only represents continuous time dynamic behavior, it should work without problems when imported as a Modelica block.

Please also note that the current implementation of FMI Import in OpenModelica is based on a built-in wrapper that uses a *reinit()* statement in an algorithm section. This is not allowed by the Modelica Language Specification, so it is necessary to set the compiler to accept this non-standard construct by setting the *--allowNonStandardModelica=reinitInAlgorithms* compiler flag. In OMEdit, you can set this option by activating the *Enable FMU Import* checkbox in the *Tools* | *Options* | *Simulation* | *Translation Flags* tab. This will generate

a warning during compilation, as there is no guarantee that the imported model using this feature can be ported to other Modelica tools; if you want to use a model that contains imported FMUs in another Modelica tool, you should rely on the other tool's import feature to generate the Modelica blocks corresponding to the FMUs.

After setting the *--allowNonStandardModelica* flag, to import the FMU package use the OpenModelica command importFMU,

```
>>> list(OpenModelica.Scripting.importFMU, interfaceOnly=true);
```

The command could be used from command line interface, OMShell, OMNotebook or MDT. The importFMU command is also integrated with OMEdit through the *File* > *Import* > *FMU* dialog: the FMU package is extracted in the directory specified by workdir, or in the current directory of omc if not specified, see *Tools* > *Open Working Directory*.

The imported FMU is then loaded in the Libraries Browser and can be used as any other regular Modelica block.

CHAPTER

FOURTEEN

OMSIMULATOR

Version: v1.17.0.postdev.alpha0-5508-g4c2a253

14.1 Introduction

The OMSimulator project is a FMI-based co-simulation tool. It supports large-scale simulation and virtual prototyping using models from multiple sources utilizing the FMI standard. It is integrated into OpenModelica but also available stand-alone, i.e., without dependencies to Modelica specific models or technology. OMSimulator provides an industrial-strength open-source FMI-based modelling and simulation tool. Input/output ports of FMUs can be connected, ports can be grouped to buses, FMUs can be parameterized and composed, and composite models can be exported according to the SSP (System Structure and Parameterization) standard. Efficient FMI based simulation is provided for both model exchange and co-simulation. An external API is available for use from other tools and scripting languages such as *Python* and *Lua*.

14.2 OMSimulator

OMSimulator is a command line wrapper for the OMSimulatorLib library.

14.2.1 OMSimulator Flags

A brief description of all command line flags will be displayed using `OMSimulator --help`:

```
usage: OMSimulator [-h] [--validate] [--version]
                   [--addParametersToCSV | --no-addParametersToCSV]
                   [--algLoopSolver {fixedpoint,kinsol}] [--clearAllOptions]
                   [--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS]
                   [--CVODEMaxNLSFailures CVODEMAXNLSFAILURES]
                   [--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS]
                   [--CVODEMaxSteps CVODEMAXSTEPS]
                   [--deleteTempFiles | --no-deleteTempFiles]
                   [--directionalDerivatives | --no-directionalDerivatives]
                   [--dumpAlgLoops | --no-dumpAlgLoops]
                   [--emitEvents | --no-emitEvents]
                   [--ignoreInitialUnknowns | --no-ignoreInitialUnknowns]
                   [--initialStepSize INITIALSTEPSIZE]
                   [--inputExtrapolation | --no-inputExtrapolation]
                   [--intervals INTERVALS] [--logFile LOGFILE]
                   [--logLevel LOGLEVEL] [--master MASTER]
                   [--maxEventIteration MAXEVENTITERATION]
                   [--maxLoopIteration MAXLOOPITERATION]
                   [--minimumStepSize MINIMUMSTEPSIZE] [--mode {cs,me}]
                   [--numProcs NUMPROCS] [--progressBar | --no-progressBar]
                   [--realTime | --no-realTime] [--resultFile RESULTFILE]
                   [--skipCSVHeader | --no-skipCSVHeader]
```

(continues on next page)

(continued from previous page)

```
                    [--solver {euler,cvode}] [--solverStats | --no-solverStats]
                    [--startTime STARTTIME] [--stepSize STEPSIZE]
                    [--stopTime STOPTIME] [--stripRoot | --no-stripRoot]
                    [--suppressPath | --no-suppressPath] [--tempDir TEMPDIR]
                    [--timeout TIMEOUT] [--tolerance TOLERANCE]
                    [--wallTime | --no-wallTime] [--workingDir WORKINGDIR]
                    [--zeroNominal | --no-zeroNominal]
                    model

Command line entry point for OMSimulator. Allows running a model file
directly, e.g.: OMSimulator test.ssp OMSimulator model.fmu without having to
write a driver script. All simulation settings (start/stop time, result file,
tolerance, ...) are taken from the model itself; use
--resultFile/--startTime/--stopTime/--tolerance/ --stepSize to override them.
For FMUs that export both model exchange and co-simulation, use --mode to pick
which one to run. Pass --stripRoot to drop the "model.root" prefix from
exported signal names. Files are validated against their schema (FMI for .fmu,
SSP for .ssp) before simulation; pass --validate to only validate the file and
skip simulation e.g.: OMSimulator --validate test.ssp OMSimulator --validate
model.fmu

positional arguments:
  model                 Path to a .ssp or .fmu file to simulate

options:
  -h, --help            show this help message and exit
  --validate            Only validate the file against its schema; do not
                        simulate
  --version             show program's version number and exit
  --addParametersToCSV, --no-addParametersToCSV
                        Export parameters to a .csv file (default: false)
  --algLoopSolver {fixedpoint,kinsol}
                        Specifies the loop solver method used for algebraic
                        loops spanning multiple components (default: kinsol)
  --clearAllOptions     Reset all flags to their default values
  --CVODEMaxErrTestFails CVODEMAXERRTESTFAILS
                        Maximum number of error test failures for CVODE
                        (default: 100)
  --CVODEMaxNLSFailures CVODEMAXNLSFAILURES
                        Maximum number of nonlinear convergence failures for
                        CVODE (default: 100)
  --CVODEMaxNLSIterations CVODEMAXNLSITERATIONS
                        Maximum number of nonlinear solver iterations for
                        CVODE (default: 5)
  --CVODEMaxSteps CVODEMAXSTEPS
                        Maximum number of steps for CVODE (default: 1000)
  --deleteTempFiles, --no-deleteTempFiles
                        Delete temporary files as soon as they are no longer
                        needed (default: true)
  --directionalDerivatives, --no-directionalDerivatives
                        Use directional derivatives to calculate the Jacobian
                        for algebraic loops (default: true)
  --dumpAlgLoops, --no-dumpAlgLoops
                        Dump information for algebraic loops (default: false)
  --emitEvents, --no-emitEvents
                        Emit events during simulation (default: true)
```

(continues on next page)

(continued from previous page)

```
--ignoreInitialUnknowns, --no-ignoreInitialUnknowns
                      Ignore initial unknowns from the modelDescription.xml
                      (default: false)
--initialStepSize INITIALSTEPSIZE
                      Specify the initial step size (default: 1e-06)
--inputExtrapolation, --no-inputExtrapolation
                      Enable input extrapolation using derivative
                      information (default: false)
--intervals INTERVALS
                      Specify the number of communication points (arg > 1)
                      (default: 500)
--logFile LOGFILE     Specify the log file (default: stdout)
--logLevel LOGLEVEL   Set the log level (0: default, 1: debug, 2:
                      debug+trace) (default: 0)
--master MASTER       Specify the master algorithm (ma) (default: ma)
--maxEventIteration MAXEVENTITERATION
                      Specify the maximum number of iterations for handling
                      a single event (default: 100)
--maxLoopIteration MAXLOOPITERATION
                      Specify the maximum number of iterations for solving
                      algebraic loops between system-level components.
                      Internal algebraic loops of components are not
                      affected. (default: 10)
--minimumStepSize MINIMUMSTEPSIZE
                      Specify the minimum step size (default: 1e-12)
--mode {cs,me}        Force 'cs' (co-simulation) or 'me' (model exchange)
                      for FMUs that export both kinds (.fmu only) (default:
                      co-simulation)
--numProcs NUMPROCS   Specify the maximum number of processors to use
                      (0=auto, 1=default) (default: 1)
--progressBar, --no-progressBar
                      Show a progress bar for the simulation progress in the
                      terminal (default: false)
--realTime, --no-realTime
                      Enable experimental feature for (soft) real-time co-
                      simulation (default: false)
--resultFile RESULTFILE
                      Specify the name of the output result file (default:
                      the model name plus '_res.mat')
--skipCSVHeader, --no-skipCSVHeader
                      Skip the CSV delimiter row in the header of .csv
                      result files (default: true)
--solver {euler,cvode}
                      Set the ODE solver for model-exchange FMUs (.fmu,
                      mode=me only) (default: cvode)
--solverStats, --no-solverStats
                      Add solver stats to the result file, e.g., step size;
                      not supported for all solvers (default: false)
--startTime STARTTIME
                      Specify the start time (default: from the model)
--stepSize STEPSIZE   Specify the (maximum) step size (.fmu only) (default:
                      from the model)
--stopTime STOPTIME   Specify the stop time (default: from the model)
--stripRoot, --no-stripRoot
                      Remove the root system prefix from all exported
                      signals (default: false)
```

(continues on next page)

(continued from previous page)

```
  --suppressPath, --no-suppressPath
                        Suppress path information in info messages; especially
                        useful for testing (default: true)
  --tempDir TEMPDIR     Specify the temporary directory (default: the working
                        directory)
  --timeout TIMEOUT     Specify the maximum allowed time in seconds for
                        running a simulation (default: 0 (disabled))
  --tolerance TOLERANCE
                        Specify the relative tolerance (default: from the
                        model)
  --wallTime, --no-wallTime
                        Add wall time information to the result file (default:
                        false)
  --workingDir WORKINGDIR
                        Specify the working directory (default: the current
                        directory)
  --zeroNominal, --no-zeroNominal
                        Accept FMUs with invalid nominal values and replace
                        the invalid nominal values with 1.0 (default: false)
```

To use flag `logLevel` with option debug (`--logLevel=1`) or debug+trace (`--logLevel=2`) one needs to build OMSimulator with debug configuration enabled. Refer to the OMSimulator README on GitHub for further instructions.

14.2.2 Examples

```
OMSimulator --timeout 180 example.lua
```

14.3 OMSimulatorLib

This library is the core of OMSimulator and provides a C interface that can easily be utilized to handle co-simulation scenarios.

14.4 C-API

14.4.1 RunFile

Simulates a single FMU or SSP model.

```
oms_status_enu_t oms_RunFile(const char* filename);
```

14.4.2 activateVariant

This API provides support to activate a multi-variant modelling from an ssp file [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file. By default when importing a ssp file the default variant will be "SystemStructure.ssd". The users can be able to switch between other variants by using this API and make changes to that particular variant and simulate them.

```
oms_status_enu_t oms_activateVariant(const char* crefA, const char* crefB);
```

An example of activating the number of available variants in a ssp file

oms_newModel("model") oms_addSystem("model.root", "system_wc") oms_addSubModel("model.root.A", "A.fmu") oms_duplicateVariant("model", "varA") // varA

will be the current variant oms_duplicateVariant("varA", "varB") // varB will be the current variant oms_activateVariant("varB", "varA") // Reactivate the variant varB to varA oms_activateVariant("varA", "model") // Reactivate the variant varA to model

14.4.3 addBus

Adds a bus to a given component.

```
oms_status_enu_t oms_addBus(const char* cref);
```

14.4.4 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
oms_status_enu_t oms_addConnection(const char* crefA, const char* crefB, bool␣
↪suppressUnitConversion);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary. The third argument suppressUnitConversion is optional and the default value is *false* which allows automatic unit conversion between connections, if set to *true* then automatic unit conversion will be disabled.

14.4.5 addConnector

Adds a connector to a given component.

```
oms_status_enu_t oms_addConnector(const char* cref, oms_causality_enu_t causality,␣
↪oms_signal_type_enu_t type);
```

14.4.6 addConnectorToBus

Adds a connector to a bus.

```
oms_status_enu_t oms_addConnectorToBus(const char* busCref, const char*␣
↪connectorCref);
```

14.4.7 addResources

Adds an external resources to an existing SSP. The external resources should be a ".ssv" or ".ssm" file

```
oms_status_enu_t oms_addResources(const char* cref_, const char* path)
```

14.4.8 addSignalsToResults

Add all variables that match the given regex to the result file.

```
oms_status_enu_t oms_addSignalsToResults(const char* cref, const char* regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.4.9 addSubModel

Adds a component to a system.

```
oms_status_enu_t oms_addSubModel(const char* cref, const char* fmuPath);
```

14.4.10 addSystem

Adds a (sub-)system to a model or system.

```
oms_status_enu_t oms_addSystem(const char* cref, oms_system_enu_t type);
```

14.4.11 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
int oms_compareSimulationResults(const char* filenameA, const char* filenameB, const␣
↪char* var, double relTol, double absTol);
```

The following table describes the input values:

Input	Type	Description
filenameA	String	Name of first result file to compare.
filenameB	String	Name of second result file to compare.
var	String	Name of signal to compare.
relTol	Number	Relative tolerance.
absTol	Number	Absolute tolerance.

The following table describes the return values:

Type	Description
Integer	1 if the signal is considered as equal, 0 otherwise

14.4.12 copySystem

Copies a system.

```
oms_status_enu_t oms_copySystem(const char* source, const char* target);
```

14.4.13 delete

Deletes a connector, component, system, or model object.

```
oms_status_enu_t oms_delete(const char* cref);
```

14.4.14 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
oms_status_enu_t oms_deleteConnection(const char* crefA, const char* crefB);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

14.4.15 deleteConnectorFromBus

Deletes a connector from a given bus.

```
oms_status_enu_t oms_deleteConnectorFromBus(const char* busCref, const char*␣
↪connectorCref);
```

14.4.16 deleteResources

Deletes the reference and resource file in a SSP. Deletion of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) deleting only the reference file in ".ssd".

2) deleting both reference and resource files in ".ssp".

To delete only the reference file in ssd, the user should provide the full qualified cref of the ".ssv" file associated with a system or subsystem or component (e.g) "model.root:root1.ssv".

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the ".ssv" file (e.g) "model:root1.ssv".

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file then the ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
oms_status_enu_t oms_deleteResources(const char* cref);
```

14.4.17 doStep

Simulates a macro step of the given composite model. The step size will be determined by the master algorithm and is limited by the definied minimal and maximal step sizes.

```
oms_status_enu_t oms_doStep(const char* cref);
```

14.4.18 duplicateVariant

This API provides support to develop a multi-variant modelling in OMSimulator [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd]. When duplicating a variant, the new variant becomes the current variant and all the changes made by the users are applied to the new variants only, and all the ssv and ssm resources associated with the new variant will be given new name based on the variant name provided by the user. This allows the bundling of multiple variants of a system structure definition referencing a similar set of packaged resources as a single SSP. However there must still be one SSD file named SystemStructure.ssd at the root of the ZIP archive which will be considered as default variant.

```
oms_status_enu_t oms_duplicateVariant(const char* crefA, const char* crefB);
```

An example of creating a multi-variant modelling is presented below

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_setReal("model.root.A.param1", "10")
oms_duplicateVariant("model", "varB")
oms_addSubModel("varB.root.B" ,"B.fmu")
oms_setReal("varB.root.A.param2", "20")
oms_export("varB", "variant.ssp")
```

The variant.ssp file will have the following structure

```
Variant.ssp
    SystemStructure.ssd
    varB.ssd
    resources\
        A.fmu
        B.fmu
```

14.4.19 export

Exports a composite model to a SPP file.

```
oms_status_enu_t oms_export(const char* cref, const char* filename);
```

14.4.20 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
oms_status_enu_t oms_exportDependencyGraphs(const char* cref, const char*␣
↪initialization, const char* event, const char* simulation);
```

14.4.21 exportSSMTemplate

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

```
oms_status_enu_t oms_exportSSMTemplate(const char* cref, const char* filename)
```

14.4.22 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.

```
oms_status_enu_t oms_exportSSVTemplate(const char* cref, const char* filename)
```

14.4.23 exportSnapshot

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_exportSnapshot(const char* cref, char** contents);
```

14.4.24 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

```
void oms_freeMemory(void* obj);
```

14.4.25 getBoolean

Get boolean value of given signal.

```
oms_status_enu_t oms_getBoolean(const char* cref, bool* value);
```

14.4.26 getBus

Gets the bus object.

```
oms_status_enu_t oms_getBus(const char* cref, oms_busconnector_t** busConnector);
```

14.4.27 getComponentType

Gets the type of the given component.

```
oms_status_enu_t oms_getComponentType(const char* cref, oms_component_enu_t* type);
```

14.4.28 getConnections

Get list of all connections from a given component.

```
oms_status_enu_t oms_getConnections(const char* cref, oms_connection_t***␣
↪connections);
```

14.4.29 getConnector

Gets the connector object of the given connector cref.

```
oms_status_enu_t oms_getConnector(const char* cref, oms_connector_t** connector);
```

14.4.30 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

```
oms_status_enu_t oms_getDirectionalDerivative(const char* cref, double* value);
```

14.4.31 getElement

Get element information of a given component reference.

```
oms_status_enu_t oms_getElement(const char* cref, oms_element_t** element);
```

14.4.32 getElements

Get list of all sub-components of a given component reference.

```
oms_status_enu_t oms_getElements(const char* cref, oms_element_t*** elements);
```

14.4.33 getFMUInfo

Returns FMU specific information.

```
oms_status_enu_t oms_getFMUInfo(const char* cref, const oms_fmu_info_t** fmuInfo);
```

14.4.34 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
oms_status_enu_t oms_getFixedStepSize(const char* cref, double* stepSize);
```

14.4.35 getInteger

Get integer value of given signal.

```
oms_status_enu_t oms_getInteger(const char* cref, int* value);
```

14.4.36 getModelState

Gets the model state of the given model cref.

```
oms_status_enu_t oms_getModelState(const char* cref, oms_modelState_enu_t*␣
↪modelState);
```

14.4.37 getReal

Get real value.

```
oms_status_enu_t oms_getReal(const char* cref, double* value);
```

14.4.38 getResultFile

Gets the result filename and buffer size of the given model cref.

```
oms_status_enu_t oms_getResultFile(const char* cref, char** filename, int*␣
↪bufferSize);
```

14.4.39 getSolver

Gets the selected solver method of the given system.

```
oms_status_enu_t oms_getSolver(const char* cref, oms_solver_enu_t* solver);
```

14.4.40 getStartTime

Get the start time from the model.

```
oms_status_enu_t oms_getStartTime(const char* cref, double* startTime);
```

14.4.41 getStopTime

Get the stop time from the model.

```
oms_status_enu_t oms_getStopTime(const char* cref, double* stopTime);
```

14.4.42 getString

Get string value.

Memory is allocated for *value*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_getString(const char* cref, char** value);
```

14.4.43 getSubModelPath

Returns the path of a given component.

```
oms_status_enu_t oms_getSubModelPath(const char* cref, char** path);
```

14.4.44 getSystemType

Gets the type of the given system.

```
oms_status_enu_t oms_getSystemType(const char* cref, oms_system_enu_t* type);
```

14.4.45 getTime

Get the current simulation time from the model.

```
oms_status_enu_t oms_getTime(const char* cref, double* time);
```

14.4.46 getTolerance

Gets the tolerance of a given system or component.

```
oms_status_enu_t oms_getTolerance(const char* cref, double* relativeTolerance);
```

14.4.47 getVariableStepSize

Gets the step size parameters.

```
oms_status_enu_t oms_getVariableStepSize(const char* cref, double* initialStepSize,␣
↪double* minimumStepSize, double* maximumStepSize);
```

14.4.48 getVersion

Returns the library's version string.

```
const char* oms_getVersion();
```

14.4.49 importFile

Imports a composite model from a SSP file.

```
oms_status_enu_t oms_importFile(const char* filename, char** cref);
```

14.4.50 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
oms_status_enu_t oms_importSnapshot(const char* cref, const char* snapshot, char**␣
↪newCref);
```

14.4.51 initialize

Initializes a composite model.

```
oms_status_enu_t oms_initialize(const char* cref);
```

14.4.52 instantiate

Instantiates a given composite model.

```
oms_status_enu_t oms_instantiate(const char* cref);
```

14.4.53 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_list(const char* cref, char** contents);
```

14.4.54 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_listUnconnectedConnectors(const char* cref, char** contents);
```

14.4.55 listVariants

This API shows the number of variants available [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file.

```
oms_status_enu_t oms_listVariants(const char* cref);
```

An example for finding the number of available variants in a ssp file

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_duplicateVariant("model", "varA")
oms_duplicateVariant("varA", "varB")

oms_listVariants("varB")
```

The API will list the available variants like below

```
<oms:Variants>
  <oms:variant name="model" />
  <oms:variant name="varB" />
  <oms:variant name="varA" />
</oms:Variants>
```

14.4.56 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
oms_status_enu_t oms_loadSnapshot(const char* cref, const char* snapshot, char**␣
↪newCref);
```

14.4.57 newModel

Creates a new and yet empty composite model.

```
oms_status_enu_t oms_newModel(const char* cref);
```

14.4.58 newResources

Adds a new empty resources to the SSP. The resource file is a ".ssv" file where the parameter values set by the users using "oms_setReal()", "oms_setInteger()" and "oms_setReal()" are writtern to the file. Currently only ".ssv" files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
oms_status_enu_t oms_newResources(const char* cref)
```

14.4.59 referenceResources

Switches the references of ".ssv" and ".ssm" in a SSP file. Referencing of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) Referencing only the ".ssv" file.

2) Referencing both the ".ssv" along with the ".ssm" file.

This API should be used in combination with "oms_deleteResources".To switch with a new reference, the old reference must be deleted first using "oms_deleteResources" and then reference with new resources.

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file, then the reference of ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as the ssm file is binded to a ssv file. Hence it is not possible to switch the reference of ".ssm" file alone. So inorder to switch the reference of ".ssm" file, the users need to bind the reference of ".ssm" file along with the ".ssv".

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) ":root.ssv", and the ".ssm" file is optional and is provided by the user as the second argument to the API.

```
oms_status_enu_t oms_referenceResources(const char* cref, const char* ssmFile);
```

14.4.60 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
oms_status_enu_t oms_removeSignalsFromResults(const char* cref, const char* regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.4.61 rename

Renames a model, system, or component.

```
oms_status_enu_t oms_rename(const char* cref, const char* newCref);
```

14.4.62 replaceSubModel

Replaces an existing fmu component, with a new component provided by the user, When replacing the fmu checks are made in all ssp concepts like in ssd, ssv and ssm, so that connections and parameter settings are not lost. It is possible that the namings of inputs and parameters match, but the start values might have been changed, in such cases new start values will be applied in ssd, ssv and ssm. In case if the Types of inputs and outputs and parameters differed, then the variables are updated according to the new changes and the connections will be removed with warning messages to user. In case when replacing a fmu, if the fmu contains parameter mapping associated with the ssv file, then only the ssm file entries are updated and the start values in the ssv files will not be changed.

```
oms_status_enu_t oms_replaceSubModel(const char* cref, const char* fmuPath);
```

It is possible to import an partially developed fmu (i.e contains only modeldescription.xml without any binaries) in OMSimulator, and later can be replaced with a fully develped fmu. An example to use the API, oms_addSubModel("model.root.A", "../resources/replaceA.fmu") oms_export("model", "test.ssp") oms_import("test.ssp") oms_replaceSubModel("model.root.A", "../resources/replaceA_extended.fmu")

14.4.63 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
oms_status_enu_t oms_reset(const char* cref);
```

14.4.64 setActivationRatio

Experimental feature for setting the activation ratio of FMUs for experimenting with multi-rate master algorithms.

```
oms_status_enu_t experimental_setActivationRatio(const char* cref, int k);
```

14.4.65 setBoolean

Sets the value of a given boolean signal.

```
oms_status_enu_t oms_setBoolean(const char* cref, bool value);
```

14.4.66 setBusGeometry

```
oms_status_enu_t oms_setBusGeometry(const char* bus, const ssd_connector_geometry_t*␣
↪geometry);
```

14.4.67 setCommandLineOption

Sets special flags.

```
oms_status_enu_t oms_setCommandLineOption(const char* cmd);
```

Available flags:

```
usage: OMSimulator [-h] [--validate] [--version]
                   [--addParametersToCSV | --no-addParametersToCSV]
                   [--algLoopSolver {fixedpoint,kinsol}] [--clearAllOptions]
                   [--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS]
                   [--CVODEMaxNLSFailures CVODEMAXNLSFAILURES]
                   [--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS]
                   [--CVODEMaxSteps CVODEMAXSTEPS]
                   [--deleteTempFiles | --no-deleteTempFiles]
                   [--directionalDerivatives | --no-directionalDerivatives]
                   [--dumpAlgLoops | --no-dumpAlgLoops]
                   [--emitEvents | --no-emitEvents]
                   [--ignoreInitialUnknowns | --no-ignoreInitialUnknowns]
                   [--initialStepSize INITIALSTEPSIZE]
                   [--inputExtrapolation | --no-inputExtrapolation]
                   [--intervals INTERVALS] [--logFile LOGFILE]
                   [--logLevel LOGLEVEL] [--master MASTER]
```

(continues on next page)

(continued from previous page)

```
                    [--maxEventIteration MAXEVENTITERATION]
                    [--maxLoopIteration MAXLOOPITERATION]
                    [--minimumStepSize MINIMUMSTEPSIZE] [--mode {cs,me}]
                    [--numProcs NUMPROCS] [--progressBar | --no-progressBar]
                    [--realTime | --no-realTime] [--resultFile RESULTFILE]
                    [--skipCSVHeader | --no-skipCSVHeader]
                    [--solver {euler,cvode}] [--solverStats | --no-solverStats]
                    [--startTime STARTTIME] [--stepSize STEPSIZE]
                    [--stopTime STOPTIME] [--stripRoot | --no-stripRoot]
                    [--suppressPath | --no-suppressPath] [--tempDir TEMPDIR]
                    [--timeout TIMEOUT] [--tolerance TOLERANCE]
                    [--wallTime | --no-wallTime] [--workingDir WORKINGDIR]
                    [--zeroNominal | --no-zeroNominal]
                    model

Command line entry point for OMSimulator. Allows running a model file
directly, e.g.: OMSimulator test.ssp OMSimulator model.fmu without having to
write a driver script. All simulation settings (start/stop time, result file,
tolerance, ...) are taken from the model itself; use
--resultFile/--startTime/--stopTime/--tolerance/ --stepSize to override them.
For FMUs that export both model exchange and co-simulation, use --mode to pick
which one to run. Pass --stripRoot to drop the "model.root" prefix from
exported signal names. Files are validated against their schema (FMI for .fmu,
SSP for .ssp) before simulation; pass --validate to only validate the file and
skip simulation e.g.: OMSimulator --validate test.ssp OMSimulator --validate
model.fmu

positional arguments:
  model                 Path to a .ssp or .fmu file to simulate

options:
  -h, --help            show this help message and exit
  --validate            Only validate the file against its schema; do not
                        simulate
  --version             show program's version number and exit
  --addParametersToCSV, --no-addParametersToCSV
                        Export parameters to a .csv file (default: false)
  --algLoopSolver {fixedpoint,kinsol}
                        Specifies the loop solver method used for algebraic
                        loops spanning multiple components (default: kinsol)
  --clearAllOptions     Reset all flags to their default values
  --CVODEMaxErrTestFails CVODEMAXERRTESTFAILS
                        Maximum number of error test failures for CVODE
                        (default: 100)
  --CVODEMaxNLSFailures CVODEMAXNLSFAILURES
                        Maximum number of nonlinear convergence failures for
                        CVODE (default: 100)
  --CVODEMaxNLSIterations CVODEMAXNLSITERATIONS
                        Maximum number of nonlinear solver iterations for
                        CVODE (default: 5)
  --CVODEMaxSteps CVODEMAXSTEPS
                        Maximum number of steps for CVODE (default: 1000)
  --deleteTempFiles, --no-deleteTempFiles
                        Delete temporary files as soon as they are no longer
                        needed (default: true)
  --directionalDerivatives, --no-directionalDerivatives
```

(continues on next page)

(continued from previous page)

```
                      Use directional derivatives to calculate the Jacobian
                      for algebraic loops (default: true)
--dumpAlgLoops, --no-dumpAlgLoops
                      Dump information for algebraic loops (default: false)
--emitEvents, --no-emitEvents
                      Emit events during simulation (default: true)
--ignoreInitialUnknowns, --no-ignoreInitialUnknowns
                      Ignore initial unknowns from the modelDescription.xml
                      (default: false)
--initialStepSize INITIALSTEPSIZE
                      Specify the initial step size (default: 1e-06)
--inputExtrapolation, --no-inputExtrapolation
                      Enable input extrapolation using derivative
                      information (default: false)
--intervals INTERVALS
                      Specify the number of communication points (arg > 1)
                      (default: 500)
--logFile LOGFILE     Specify the log file (default: stdout)
--logLevel LOGLEVEL   Set the log level (0: default, 1: debug, 2:
                      debug+trace) (default: 0)
--master MASTER       Specify the master algorithm (ma) (default: ma)
--maxEventIteration MAXEVENTITERATION
                      Specify the maximum number of iterations for handling
                      a single event (default: 100)
--maxLoopIteration MAXLOOPITERATION
                      Specify the maximum number of iterations for solving
                      algebraic loops between system-level components.
                      Internal algebraic loops of components are not
                      affected. (default: 10)
--minimumStepSize MINIMUMSTEPSIZE
                      Specify the minimum step size (default: 1e-12)
--mode {cs,me}        Force 'cs' (co-simulation) or 'me' (model exchange)
                      for FMUs that export both kinds (.fmu only) (default:
                      co-simulation)
--numProcs NUMPROCS   Specify the maximum number of processors to use
                      (0=auto, 1=default) (default: 1)
--progressBar, --no-progressBar
                      Show a progress bar for the simulation progress in the
                      terminal (default: false)
--realTime, --no-realTime
                      Enable experimental feature for (soft) real-time co-
                      simulation (default: false)
--resultFile RESULTFILE
                      Specify the name of the output result file (default:
                      the model name plus '_res.mat')
--skipCSVHeader, --no-skipCSVHeader
                      Skip the CSV delimiter row in the header of .csv
                      result files (default: true)
--solver {euler,cvode}
                      Set the ODE solver for model-exchange FMUs (.fmu,
                      mode=me only) (default: cvode)
--solverStats, --no-solverStats
                      Add solver stats to the result file, e.g., step size;
                      not supported for all solvers (default: false)
--startTime STARTTIME
                      Specify the start time (default: from the model)
```

(continues on next page)

(continued from previous page)

```
  --stepSize STEPSIZE   Specify the (maximum) step size (.fmu only) (default:
                        from the model)
  --stopTime STOPTIME   Specify the stop time (default: from the model)
  --stripRoot, --no-stripRoot
                        Remove the root system prefix from all exported
                        signals (default: false)
  --suppressPath, --no-suppressPath
                        Suppress path information in info messages; especially
                        useful for testing (default: true)
  --tempDir TEMPDIR     Specify the temporary directory (default: the working
                        directory)
  --timeout TIMEOUT     Specify the maximum allowed time in seconds for
                        running a simulation (default: 0 (disabled))
  --tolerance TOLERANCE
                        Specify the relative tolerance (default: from the
                        model)
  --wallTime, --no-wallTime
                        Add wall time information to the result file (default:
                        false)
  --workingDir WORKINGDIR
                        Specify the working directory (default: the current
                        directory)
  --zeroNominal, --no-zeroNominal
                        Accept FMUs with invalid nominal values and replace
                        the invalid nominal values with 1.0 (default: false)
```

14.4.68 setConnectionGeometry

```
oms_status_enu_t oms_setConnectionGeometry(const char* crefA, const char* crefB,␣
↪const ssd_connection_geometry_t* geometry);
```

14.4.69 setConnectorGeometry

Set geometry information to a given connector.

```
oms_status_enu_t oms_setConnectorGeometry(const char* cref, const ssd_connector_
↪geometry_t* geometry);
```

14.4.70 setElementGeometry

Set geometry information to a given component.

```
oms_status_enu_t oms_setElementGeometry(const char* cref, const ssd_element_geometry_
↪t* geometry);
```

14.4.71 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
oms_status_enu_t oms_setFixedStepSize(const char* cref, double stepSize);
```

14.4.72 setInteger

Sets the value of a given integer signal.

```
oms_status_enu_t oms_setInteger(const char* cref, int value);
```

14.4.73 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
oms_status_enu_t oms_setLogFile(const char* filename);
```

14.4.74 setLoggingCallback

Sets a callback function for the logging system.

```
void oms_setLoggingCallback(void (*cb)(oms_message_type_enu_t type, const char*␣
↪message));
```

14.4.75 setLoggingInterval

Set the logging interval of the simulation.

```
oms_status_enu_t oms_setLoggingInterval(const char* cref, double loggingInterval);
```

14.4.76 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
void oms_setLoggingLevel(int logLevel);
```

14.4.77 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
void oms_setMaxLogFileSize(const unsigned long size);
```

14.4.78 setReal

Sets the value of a given real signal.

```
oms_status_enu_t oms_setReal(const char* cref, double value);
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

14.4.79 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
oms_status_enu_t oms_setRealInputDerivative(const char* cref, double value);
```

14.4.80 setResultFile

Set the result file of the simulation.

```
oms_status_enu_t oms_setResultFile(const char* cref, const char* filename, int␣
↪bufferSize);
```

The creation of a result file is omitted if the filename is an empty string.

14.4.81 setSolver

Sets the solver method for the given system.

```
oms_status_enu_t oms_setSolver(const char* cref, oms_solver_enu_t solver);
```

14.4.82 setStartTime

Set the start time of the simulation.

```
oms_status_enu_t oms_setStartTime(const char* cref, double startTime);
```

14.4.83 setStopTime

Set the stop time of the simulation.

```
oms_status_enu_t oms_setStopTime(const char* cref, double stopTime);
```

14.4.84 setString

Sets the value of a given string signal.

```
oms_status_enu_t oms_setString(const char* cref, const char* value);
```

14.4.85 setTempDirectory

Set new temp directory.

```
oms_status_enu_t oms_setTempDirectory(const char* newTempDir);
```

14.4.86 setTolerance

Sets the tolerance for a given model or system.

```
oms_status_enu_t oms_setTolerance(const char* cref, double relativeTolerance);
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", relativeTolerance);
oms_setTolerance("model.root", relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

14.4.87 setUnit

Sets the unit of a given signal.

```
oms_status_enu_t oms_setUnit(const char* cref, const char* value);
```

14.4.88 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
oms_status_enu_t oms_getVariableStepSize(const char* cref, double* initialStepSize,␣
↪double* minimumStepSize, double* maximumStepSize);
```

14.4.89 setWorkingDirectory

Set a new working directory.

```
oms_status_enu_t oms_setWorkingDirectory(const char* newWorkingDir);
```

14.4.90 simulate

Simulates a composite model.

```
oms_status_enu_t oms_simulate(const char* cref);
```

14.4.91 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
oms_status_enu_t experimental_simulate_realtime(const char* ident);
```

14.4.92 stepUntil

Simulates a composite model until a given time value.

```
oms_status_enu_t oms_stepUntil(const char* cref, double stopTime);
```

14.4.93 terminate

Terminates a given composite model.

```
oms_status_enu_t oms_terminate(const char* cref);
```

14.5 OMSimulatorLua

This is a shared library that provides a Lua interface for the OMSimulatorLib library.

14.5.1 Examples

```
oms_setTempDirectory("./temp/")
oms_newModel("model")
oms_addSystem("model.root", oms_system_sc)

-- instantiate FMUs
oms_addSubModel("model.root.system1", "FMUs/System1.fmu")
oms_addSubModel("model.root.system2", "FMUs/System2.fmu")

-- add connections
oms_addConnection("model.root.system1.y", "model.root.system2.u")
oms_addConnection("model.root.system2.y", "model.root.system1.u")

-- simulation settings
oms_setResultFile("model", "results.mat")
oms_setStopTime("model", 0.1)
oms_setFixedStepSize("model.root", 1e-4)

oms_instantiate("model")
oms_setReal("model.root.system1.x_start", 2.5)

oms_initialize("model")
oms_simulate("model")
oms_terminate("model")
oms_delete("model")
```

Lua Scripting Commands

14.5.2 activateVariant

This API provides support to activate a multi-variant modelling from an ssp file [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file. By default when importing a ssp file the default variant will be "SystemStructure.ssd". The users can be able to switch between other variants by using this API and make changes to that particular variant and simulate them.

```
status = oms_activateVariant(crefA, crefB)
```

An example of activating the number of available variants in a ssp file

oms_newModel("model") oms_addSystem("model.root", "system_wc") oms_addSubModel("model.root.A", "A.fmu") oms_duplicateVariant("model", "varA") // varA will be the current variant oms_duplicateVariant("varA", "varB") // varB will be the current variant oms_activateVariant("varB", "varA") // Reactivate the variant varB to varA oms_activateVariant("varA", "model") // Reactivate the variant varA to model

14.5.3 addBus

Adds a bus to a given component.

```
status = oms_addBus(cref)
```

14.5.4 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
status = oms_addConnection(crefA, crefB, suppressUnitConversion)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary. The third argument suppressUnitConversion is optional and the default value is *false* which allows automatic unit conversion between connections, if set to *true* then automatic unit conversion will be disabled.

14.5.5 addConnector

Adds a connector to a given component.

```
status = oms_addConnector(cref, causality, type)

The second argument "causality", should be any of the following,

oms_causality_input
oms_causality_output
oms_causality_parameter
oms_causality_bidir
oms_causality_undefined

The third argument "type", should be any of the following,

oms_signal_type_real
oms_signal_type_integer
oms_signal_type_boolean
oms_signal_type_string
oms_signal_type_enum
oms_signal_type_bus
```

14.5.6 addConnectorToBus

Adds a connector to a bus.

```
status = oms_addConnectorToBus(busCref, connectorCref)
```

14.5.7 addResources

Adds an external resources to an existing SSP. The external resources should be a ".ssv" or ".ssm" file

```
status = oms_addResources(cref, path)

-- Example
oms_importFile("addExternalResources1.ssp")
-- add list of external resources from filesystem to ssp
oms_addResources("addExternalResources", "../../resources/externalRoot.ssv")
oms_addResources("addExternalResources:externalSystem.ssv", "../../resources/
↪externalSystem1.ssv")
oms_addResources("addExternalResources", "../../resources/externalGain.ssv")
-- export the ssp with new resources
oms_export("addExternalResources", "addExternalResources1.ssp")
```

14.5.8 addSignalsToResults

Add all variables that match the given regex to the result file.

```
status = oms_addSignalsToResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.5.9 addSubModel

Adds a component to a system.

```
status = oms_addSubModel(cref, fmuPath)
```

14.5.10 addSystem

Adds a (sub-)system to a model or system.

```
status = oms_addSystem(cref, type)
```

14.5.11 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
oms_compareSimulationResults(filenameA, filenameB, var, relTol, absTol)
```

The following table describes the input values:

Input	Type	Description
filenameA	String	Name of first result file to compare.
filenameB	String	Name of second result file to compare.
var	String	Name of signal to compare.
relTol	Number	Relative tolerance.
absTol	Number	Absolute tolerance.

The following table describes the return values:

Type	Description
Integer	1 if the signal is considered as equal, 0 otherwise

14.5.12 copySystem

Copies a system.

```
status = oms_copySystem(source, target)
```

14.5.13 delete

Deletes a connector, component, system, or model object.

```
status = oms_delete(cref)
```

14.5.14 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
status = oms_deleteConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

14.5.15 deleteConnectorFromBus

Deletes a connector from a given bus.

```
status = oms_deleteConnectorFromBus(busCref, connectorCref)
```

14.5.16 deleteResources

Deletes the reference and resource file in a SSP. Deletion of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) deleting only the reference file in ".ssd".

2) deleting both reference and resource files in ".ssp".

To delete only the reference file in ssd, the user should provide the full qualified cref of the ".ssv" file associated with a system or subsystem or component (e.g) "model.root:root1.ssv".

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the ".ssv" file (e.g) "model:root1.ssv".

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file then the ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
status = oms_deleteResources(cref)

-- Example
oms_importFile("deleteResources1.ssp")
-- delete only the references in ".ssd" file
oms_deleteResources("deleteResources.root:root.ssv")
-- delete both references and resources
oms_deleteResources("deleteResources:root.ssv")
oms_export("deleteResources1.ssp")
```

14.5.17 duplicateVariant

This API provides support to develop a multi-variant modelling in OMSimulator [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd]. When duplicating a variant, the new variant becomes the current variant and all the changes made by the users are applied to the new variants only, and all the ssv and ssm resources associated with the new variant will be given new name based on the variant name provided by the user. This allows the bundling of multiple variants of a system structure definition referencing a similar set of packaged resources as a single SSP. However there must still be one SSD file named SystemStructure.ssd at the root of the ZIP archive which will be considered as default variant.

```
status = oms_duplicateVariant(crefA, crefB)
```

An example of creating a multi-variant modelling is presented below

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_setReal("model.root.A.param1", "10")
oms_duplicateVariant("model", "varB")
oms_addSubModel("varB.root.B" ,"B.fmu")
oms_setReal("varB.root.A.param2", "20")
oms_export("varB", "variant.ssp")
```

The variant.ssp file will have the following structure

```
Variant.ssp
    SystemStructure.ssd
    varB.ssd
    resources\
        A.fmu
        B.fmu
```

14.5.18 export

Exports a composite model to a SPP file.

```
status = oms_export(cref, filename)
```

14.5.19 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
status = oms_exportDependencyGraphs(cref, initialization, event, simulation)
```

14.5.20 exportSSMTemplate

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

```
status = oms_exportSSMTemplate(cref, filename)
```

14.5.21 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.

```
status = oms_exportSSVTemplate(cref, filename)
```

14.5.22 exportSnapshot

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms_exportSnapshot(cref)
```

14.5.23 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

This function is neither needed nor available from the Lua interface.

14.5.24 getBoolean

Get boolean value of given signal.

```
value, status = oms_getBoolean(cref)
```

14.5.25 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

```
value, status = oms_getDirectionalDerivative(cref)
```

14.5.26 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
stepSize, status = oms_setFixedStepSize(cref)
```

14.5.27 getInteger

Get integer value of given signal.

```
value, status = oms_getInteger(cref)
```

14.5.28 getModelState

Gets the model state of the given model cref.

```
modelState, status = oms_getModelState(cref)
```

14.5.29 getReal

Get real value.

```
value, status = oms_getReal(cref)
```

14.5.30 getSolver

Gets the selected solver method of the given system.

```
solver, status = oms_getSolver(cref)
```

14.5.31 getStartTime

Get the start time from the model.

```
startTime, status = oms_getStartTime(cref)
```

14.5.32 getStopTime

Get the stop time from the model.

```
stopTime, status = oms_getStopTime(cref)
```

14.5.33 getString

Get string value.

Memory is allocated for *value*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
value, status = oms_getString(cref)
```

14.5.34 getSystemType

Gets the type of the given system.

```
type, status = oms_getSystemType(cref)
```

14.5.35 getTime

Get the current simulation time from the model.

```
time, status = oms_getTime(cref)
```

14.5.36 getTolerance

Gets the tolerance of a given system or component.

```
relativeTolerance, status = oms_getTolerance(cref)
```

14.5.37 getVariableStepSize

Gets the step size parameters.

```
initialStepSize, minimumStepSize, maximumStepSize, status = oms_
↪getVariableStepSize(cref)
```

14.5.38 getVersion

Returns the library's version string.

```
version = oms_getVersion()
```

14.5.39 importFile

Imports a composite model from a SSP file.

```
cref, status = oms_importFile(filename)
```

14.5.40 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
newCref, status = oms_importSnapshot(cref, snapshot)
```

14.5.41 initialize

Initializes a composite model.

```
status = oms_initialize(cref)
```

14.5.42 instantiate

Instantiates a given composite model.

```
status = oms_instantiate(cref)
```

14.5.43 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms_list(cref)
```

14.5.44 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms_listUnconnectedConnectors(cref)
```

14.5.45 listVariants

This API shows the number of variants available [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file.

```
status = oms_listVariants(cref)
```

An example for finding the number of available variants in a ssp file

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_duplicateVariant("model", "varA")
oms_duplicateVariant("varA", "varB")

oms_listVariants("varB")
```

The API will list the available variants like below

```
<oms:Variants>
  <oms:variant name="model" />
  <oms:variant name="varB" />
  <oms:variant name="varA" />
</oms:Variants>
```

14.5.46 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
newCref, status = oms_loadSnapshot(cref, snapshot)
```

14.5.47 newModel

Creates a new and yet empty composite model.

```
status = oms_newModel(cref)
```

14.5.48 newResources

Adds a new empty resources to the SSP. The resource file is a ".ssv" file where the parameter values set by the users using "oms_setReal()", "oms_setInteger()" and "oms_setReal()" are writtern to the file. Currently only ".ssv" files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
status = oms_newResources(cref)

-- Example
oms_newModel("newResources")

oms_addSystem("newResources.root", oms_system_wc)
oms_addConnector("newResources.root.Input1", oms_causality_input, oms_signal_type_
↪real)
oms_addConnector("newResources.root.Input2", oms_causality_input, oms_signal_type_
↪real)

-- add Top level new resources, the filename is provided using the colon suffix
↪":root.ssv"
oms_newResources("newResources.root:root.ssv")
oms_setReal("newResources.root.Input1", 10)
-- export the ssp with new resources
oms_export("newResources", "newResources.ssp")
```

14.5.49 referenceResources

Switches the references of ".ssv" and ".ssm" in a SSP file. Referencing of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) Referencing only the ".ssv" file.

2) Referencing both the ".ssv" along with the ".ssm" file.

This API should be used in combination with "oms_deleteResources".To switch with a new reference, the old reference must be deleted first using "oms_deleteResources" and then reference with new resources.

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file, then the reference of ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as the ssm file is binded to a ssv file. Hence it is not possible to switch the reference of ".ssm" file alone. So inorder to switch the reference of ".ssm" file, the users need to bind the reference of ".ssm" file along with the ".ssv".

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) ":root.ssv", and the ".ssm" file is optional and is provided by the user as the second argument to the API.

```
status = oms_referenceResources(cref, ssmFile)

-- Example
oms_importFile("referenceResources1.ssp")
-- delete only the references in ".ssd" file
oms_deleteResources("referenceResources1.root:root.ssv")
-- usage-1 switch with new references, only ssv file
oms_referenceResources("referenceResources1.root:Config1.ssv")
-- usage-2 switch with new references, both ssv and ssm file
```

(continues on next page)

(continued from previous page)

```
oms_referenceResources("referenceResources1.root:Config1.ssv", "Config1.ssm")
oms_export("referenceResources1.ssp")
```

14.5.50 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
status = oms_removeSignalsFromResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.5.51 rename

Renames a model, system, or component.

```
status = oms_rename(cref, newCref)
```

14.5.52 replaceSubModel

Replaces an existing fmu component, with a new component provided by the user, When replacing the fmu checks are made in all ssp concepts like in ssd, ssv and ssm, so that connections and parameter settings are not lost. It is possible that the namings of inputs and parameters match, but the start values might have been changed, in such cases new start values will be applied in ssd, ssv and ssm. In case if the Types of inputs and outputs and parameters differed, then the variables are updated according to the new changes and the connections will be removed with warning messages to user. In case when replacing a fmu, if the fmu contains parameter mapping associated with the ssv file, then only the ssm file entries are updated and the start values in the ssv files will not be changed.

```
status = oms_replaceSubModel(cref, fmuPath)
```

> It is possible to import an partially developed fmu (i.e contains only modeldescription.xml without any binaries) in OMSimulator, and later can be replaced with a fully develped fmu. An example to use the API, oms_addSubModel("model.root.A", "../resources/replaceA.fmu") oms_export("model", "test.ssp") oms_import("test.ssp") oms_replaceSubModel("model.root.A", "../resources/replaceA_extended.fmu")

14.5.53 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
status = oms_reset(cref)
```

14.5.54 setActivationRatio

Experimental feature for setting the activation ratio of FMUs for experimenting with multi-rate master algorithms.

```
status = experimental_setActivationRatio(cref, k)
```

14.5.55 setBoolean

Sets the value of a given boolean signal.

```
status = oms_setBoolean(cref, value)
```

14.5.56 setCommandLineOption

Sets special flags.

```
status = oms_setCommandLineOption(cmd)
```

Available flags:

```
usage: OMSimulator [-h] [--validate] [--version]
                   [--addParametersToCSV | --no-addParametersToCSV]
                   [--algLoopSolver {fixedpoint,kinsol}] [--clearAllOptions]
                   [--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS]
                   [--CVODEMaxNLSFailures CVODEMAXNLSFAILURES]
                   [--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS]
                   [--CVODEMaxSteps CVODEMAXSTEPS]
                   [--deleteTempFiles | --no-deleteTempFiles]
                   [--directionalDerivatives | --no-directionalDerivatives]
                   [--dumpAlgLoops | --no-dumpAlgLoops]
                   [--emitEvents | --no-emitEvents]
                   [--ignoreInitialUnknowns | --no-ignoreInitialUnknowns]
                   [--initialStepSize INITIALSTEPSIZE]
                   [--inputExtrapolation | --no-inputExtrapolation]
                   [--intervals INTERVALS] [--logFile LOGFILE]
                   [--logLevel LOGLEVEL] [--master MASTER]
                   [--maxEventIteration MAXEVENTITERATION]
                   [--maxLoopIteration MAXLOOPITERATION]
                   [--minimumStepSize MINIMUMSTEPSIZE] [--mode {cs,me}]
                   [--numProcs NUMPROCS] [--progressBar | --no-progressBar]
                   [--realTime | --no-realTime] [--resultFile RESULTFILE]
                   [--skipCSVHeader | --no-skipCSVHeader]
                   [--solver {euler,cvode}] [--solverStats | --no-solverStats]
                   [--startTime STARTTIME] [--stepSize STEPSIZE]
                   [--stopTime STOPTIME] [--stripRoot | --no-stripRoot]
                   [--suppressPath | --no-suppressPath] [--tempDir TEMPDIR]
                   [--timeout TIMEOUT] [--tolerance TOLERANCE]
                   [--wallTime | --no-wallTime] [--workingDir WORKINGDIR]
                   [--zeroNominal | --no-zeroNominal]
                   model

Command line entry point for OMSimulator. Allows running a model file
directly, e.g.: OMSimulator test.ssp OMSimulator model.fmu without having to
write a driver script. All simulation settings (start/stop time, result file,
tolerance, ...) are taken from the model itself; use
--resultFile/--startTime/--stopTime/--tolerance/ --stepSize to override them.
For FMUs that export both model exchange and co-simulation, use --mode to pick
which one to run. Pass --stripRoot to drop the "model.root" prefix from
exported signal names. Files are validated against their schema (FMI for .fmu,
SSP for .ssp) before simulation; pass --validate to only validate the file and
skip simulation e.g.: OMSimulator --validate test.ssp OMSimulator --validate
model.fmu

positional arguments:
  model                 Path to a .ssp or .fmu file to simulate

options:
  -h, --help            show this help message and exit
  --validate            Only validate the file against its schema; do not
                        simulate
```

(continues on next page)

(continued from previous page)

```
--version             show program's version number and exit
--addParametersToCSV, --no-addParametersToCSV
                      Export parameters to a .csv file (default: false)
--algLoopSolver {fixedpoint,kinsol}
                      Specifies the loop solver method used for algebraic
                      loops spanning multiple components (default: kinsol)
--clearAllOptions     Reset all flags to their default values
--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS
                      Maximum number of error test failures for CVODE
                      (default: 100)
--CVODEMaxNLSFailures CVODEMAXNLSFAILURES
                      Maximum number of nonlinear convergence failures for
                      CVODE (default: 100)
--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS
                      Maximum number of nonlinear solver iterations for
                      CVODE (default: 5)
--CVODEMaxSteps CVODEMAXSTEPS
                      Maximum number of steps for CVODE (default: 1000)
--deleteTempFiles, --no-deleteTempFiles
                      Delete temporary files as soon as they are no longer
                      needed (default: true)
--directionalDerivatives, --no-directionalDerivatives
                      Use directional derivatives to calculate the Jacobian
                      for algebraic loops (default: true)
--dumpAlgLoops, --no-dumpAlgLoops
                      Dump information for algebraic loops (default: false)
--emitEvents, --no-emitEvents
                      Emit events during simulation (default: true)
--ignoreInitialUnknowns, --no-ignoreInitialUnknowns
                      Ignore initial unknowns from the modelDescription.xml
                      (default: false)
--initialStepSize INITIALSTEPSIZE
                      Specify the initial step size (default: 1e-06)
--inputExtrapolation, --no-inputExtrapolation
                      Enable input extrapolation using derivative
                      information (default: false)
--intervals INTERVALS
                      Specify the number of communication points (arg > 1)
                      (default: 500)
--logFile LOGFILE     Specify the log file (default: stdout)
--logLevel LOGLEVEL   Set the log level (0: default, 1: debug, 2:
                      debug+trace) (default: 0)
--master MASTER       Specify the master algorithm (ma) (default: ma)
--maxEventIteration MAXEVENTITERATION
                      Specify the maximum number of iterations for handling
                      a single event (default: 100)
--maxLoopIteration MAXLOOPITERATION
                      Specify the maximum number of iterations for solving
                      algebraic loops between system-level components.
                      Internal algebraic loops of components are not
                      affected. (default: 10)
--minimumStepSize MINIMUMSTEPSIZE
                      Specify the minimum step size (default: 1e-12)
--mode {cs,me}        Force 'cs' (co-simulation) or 'me' (model exchange)
                      for FMUs that export both kinds (.fmu only) (default:
                      co-simulation)
```

(continues on next page)

(continued from previous page)

```
  --numProcs NUMPROCS   Specify the maximum number of processors to use
                        (0=auto, 1=default) (default: 1)
  --progressBar, --no-progressBar
                        Show a progress bar for the simulation progress in the
                        terminal (default: false)
  --realTime, --no-realTime
                        Enable experimental feature for (soft) real-time co-
                        simulation (default: false)
  --resultFile RESULTFILE
                        Specify the name of the output result file (default:
                        the model name plus '_res.mat')
  --skipCSVHeader, --no-skipCSVHeader
                        Skip the CSV delimiter row in the header of .csv
                        result files (default: true)
  --solver {euler,cvode}
                        Set the ODE solver for model-exchange FMUs (.fmu,
                        mode=me only) (default: cvode)
  --solverStats, --no-solverStats
                        Add solver stats to the result file, e.g., step size;
                        not supported for all solvers (default: false)
  --startTime STARTTIME
                        Specify the start time (default: from the model)
  --stepSize STEPSIZE   Specify the (maximum) step size (.fmu only) (default:
                        from the model)
  --stopTime STOPTIME   Specify the stop time (default: from the model)
  --stripRoot, --no-stripRoot
                        Remove the root system prefix from all exported
                        signals (default: false)
  --suppressPath, --no-suppressPath
                        Suppress path information in info messages; especially
                        useful for testing (default: true)
  --tempDir TEMPDIR     Specify the temporary directory (default: the working
                        directory)
  --timeout TIMEOUT     Specify the maximum allowed time in seconds for
                        running a simulation (default: 0 (disabled))
  --tolerance TOLERANCE
                        Specify the relative tolerance (default: from the
                        model)
  --wallTime, --no-wallTime
                        Add wall time information to the result file (default:
                        false)
  --workingDir WORKINGDIR
                        Specify the working directory (default: the current
                        directory)
  --zeroNominal, --no-zeroNominal
                        Accept FMUs with invalid nominal values and replace
                        the invalid nominal values with 1.0 (default: false)
```

14.5.57 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
status = oms_setFixedStepSize(cref, stepSize)
```

14.5.58 setInteger

Sets the value of a given integer signal.

```
status = oms_setInteger(cref, value)
```

14.5.59 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
status = oms_setLogFile(filename)
```

14.5.60 setLoggingInterval

Set the logging interval of the simulation.

```
status = oms_setLoggingInterval(cref, loggingInterval)
```

14.5.61 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
oms_setLoggingLevel(logLevel)
```

14.5.62 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
oms_setMaxLogFileSize(size)
```

14.5.63 setReal

Sets the value of a given real signal.

```
status = oms_setReal(cref, value)
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

14.5.64 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
status = oms_setRealInputDerivative(cref, value)
```

14.5.65 setResultFile

Set the result file of the simulation.

```
status = oms_setResultFile(cref, filename)
status = oms_setResultFile(cref, filename, bufferSize)
```

The creation of a result file is omitted if the filename is an empty string.

14.5.66 setSolver

Sets the solver method for the given system.

```
status = oms_setSolver(cref, solver)
```

solver	Type	Description
oms_solver_sc_explicit_euler	sc-system	Explicit euler with fixed step size
oms_solver_sc_cvode	sc-system	CVODE with adaptive stepsize
oms_solver_wc_ma	wc-system	default master algorithm with fixed step size
oms_solver_wc_mav	wc-system	master algorithm with adaptive stepsize
oms_solver_wc_mav2	wc-system	master algorithm with adaptive stepsize (double-step)

14.5.67 setStartTime

Set the start time of the simulation.

```
status = oms_setStartTime(cref, startTime)
```

14.5.68 setStopTime

Set the stop time of the simulation.

```
status = oms_setStopTime(cref, stopTime)
```

14.5.69 setString

Sets the value of a given string signal.

```
status = oms_setString(cref, value)
```

14.5.70 setTempDirectory

Set new temp directory.

```
status = oms_setTempDirectory(newTempDir)
```

14.5.71 setTolerance

Sets the tolerance for a given model or system.

```
status = oms_setTolerance(const char* cref, double relativeTolerance)
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", relativeTolerance);
oms_setTolerance("model.root", relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

14.5.72 setUnit

Sets the unit of a given signal.

```
status = oms_setUnit(cref, value)
```

14.5.73 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
status = oms_getVariableStepSize(cref, initialStepSize, minimumStepSize,␣
↪maximumStepSize)
```

14.5.74 setWorkingDirectory

Set a new working directory.

```
status = oms_setWorkingDirectory(newWorkingDir)
```

14.5.75 simulate

Simulates a composite model.

```
status = oms_simulate(cref)
```

14.5.76 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
status = experimental_simulate_realtime(ident)
```

14.5.77 stepUntil

Simulates a composite model until a given time value.

```
status = oms_stepUntil(cref, stopTime)
```

14.5.78 terminate

Terminates a given composite model.

```
status = oms_terminate(cref)
```

14.6 OMSimulatorPython

This is a shared library that provides a Python interface for the OMSimulatorLib library.

Installation using `pip` is recommended:

```
> pip3 install OMSimulator --upgrade
```

14.6.1 Examples

```
from OMSimulator import OMSimulator

oms = OMSimulator()
oms.setTempDirectory("./temp/")
oms.newModel("model")
oms.addSystem("model.root", oms.system_sc)

# instantiate FMUs
oms.addSubModel("model.root.system1", "FMUs/System1.fmu")
oms.addSubModel("model.root.system2", "FMUs/System2.fmu")

# add connections
oms.addConnection("model.root.system1.y", "model.root.system2.u")
oms.addConnection("model.root.system2.y", "model.root.system1.u")

# simulation settings
oms.setResultFile("model", "results.mat")
oms.setStopTime("model", 0.1)
oms.setFixedStepSize("model.root", 1e-4)

oms.instantiate("model")
oms.setReal("model.root.system1.x_start", 2.5)

oms.initialize("model")
oms.simulate("model")
oms.terminate("model")
oms.delete("model")
```

The python package also provides a more object oriented API. The following example is equivalent to the previous one:

```
import OMSimulator as oms

oms.setTempDirectory('./temp/')
model = oms.newModel("model")
root = model.addSystem('root', oms.Types.System.SC)

# instantiate FMUs
root.addSubModel('system1', 'FMUs/System1.fmu')
root.addSubModel('system2', 'FMUs/System2.fmu')

# add connections
root.addConnection('system1.y', 'system2.u')
root.addConnection('system2.y', 'system1.u')

# simulation settings
model.resultFile = 'results.mat'
model.stopTime = 0.1
```

(continues on next page)

(continued from previous page)

```
model.fixedStepSize = 1e-4

model.instantiate()
model.setReal('root.system1.x_start', 2.5)
#or system.setReal('system1.x_start', 2.5)

model.initialize()
model.simulate()
model.terminate()
model.delete()
```

14.6.2 Python Scripting Commands

14.6.3 activateVariant

This API provides support to activate a multi-variant modelling from an ssp file [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file. By default when importing a ssp file the default variant will be "SystemStructure.ssd". The users can be able to switch between other variants by using this API and make changes to that particular variant and simulate them.

```
status = oms.activateVariant(crefA, crefB)
```

An example of activating the number of available variants in a ssp file

oms_newModel("model") oms_addSystem("model.root", "system_wc") oms_addSubModel("model.root.A", "A.fmu") oms_duplicateVariant("model", "varA") // varA will be the current variant oms_duplicateVariant("varA", "varB") // varB will be the current variant oms_activateVariant("varB", "varA") // Reactivate the variant varB to varA oms_activateVariant("varA", "model") // Reactivate the variant varA to model

14.6.4 addBus

Adds a bus to a given component.

```
status = oms.addBus(cref)
```

14.6.5 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
status = oms.addConnection(crefA, crefB, suppressUnitConversion)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary. The third argument suppressUnitConversion is optional and the default value is *false* which allows automatic unit conversion between connections, if set to *true* then automatic unit conversion will be disabled.

14.6.6 addConnector

Adds a connector to a given component.

```
status = oms.addConnector(cref, causality, type)

The second argument "causality", should be any of the following,

oms.input
oms.output
```

(continues on next page)

(continued from previous page)

```
oms.parameter
oms.bidir
oms.undefined

The third argument "type", should be any of the following,

oms.signal_type_real
oms.signal_type_integer
oms.signal_type_boolean
oms.signal_type_string
oms.signal_type_enum
oms.signal_type_bus
```

14.6.7 addConnectorToBus

Adds a connector to a bus.

```
status = oms.addConnectorToBus(busCref, connectorCref)
```

14.6.8 addResources

Adds an external resources to an existing SSP. The external resources should be a ".ssv" or ".ssm" file

```
status = oms.addResources(cref, path)

## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("addExternalResources1.ssp")
## add list of external resources from filesystem to ssp
oms.addResources("addExternalResources", "../../resources/externalRoot.ssv")
oms.addResources("addExternalResources:externalSystem.ssv", "../../resources/
↪externalSystem1.ssv")
oms.addResources("addExternalResources", "../../resources/externalGain.ssv")
## export the ssp with new resources
oms_export("addExternalResources", "addExternalResources1.ssp")
```

14.6.9 addSignalsToResults

Add all variables that match the given regex to the result file.

```
status = oms.addSignalsToResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.6.10 addSubModel

Adds a component to a system.

```
status = oms.addSubModel(cref, fmuPath)
```

14.6.11 addSystem

Adds a (sub-)system to a model or system.

```
status = oms.addSystem(cref, type)
```

14.6.12 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
oms.compareSimulationResults(filenameA, filenameB, var, relTol, absTol)
```

The following table describes the input values:

Input	Type	Description
filenameA	String	Name of first result file to compare.
filenameB	String	Name of second result file to compare.
var	String	Name of signal to compare.
relTol	Number	Relative tolerance.
absTol	Number	Absolute tolerance.

The following table describes the return values:

Type	Description
Integer	1 if the signal is considered as equal, 0 otherwise

14.6.13 copySystem

Copies a system.

```
status = oms.copySystem(source, target)
```

14.6.14 delete

Deletes a connector, component, system, or model object.

```
status = oms.delete(cref)
```

14.6.15 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
status = oms.deleteConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

14.6.16 deleteConnectorFromBus

Deletes a connector from a given bus.

```
status = oms.deleteConnectorFromBus(busCref, connectorCref)
```

14.6.17 deleteResources

Deletes the reference and resource file in a SSP. Deletion of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) deleting only the reference file in ".ssd".

2) deleting both reference and resource files in ".ssp".

To delete only the reference file in ssd, the user should provide the full qualified cref of the ".ssv" file associated with a system or subsystem or component (e.g) "model.root:root1.ssv".

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the ".ssv" file (e.g) "model:root1.ssv".

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file then the ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
status = oms.deleteResources(cref))

## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("deleteResources1.ssp")
## delete only the references in ".ssd" file
oms.deleteResources("deleteResources.root:root.ssv")
## delete both references and resources
oms.deleteResources("deleteResources:root.ssv")
oms.export("deleteResources1.ssp")
```

14.6.18 doStep

Simulates a macro step of the given composite model. The step size will be determined by the master algorithm and is limited by the definied minimal and maximal step sizes.

```
status = oms.doStep(cref)
```

14.6.19 duplicateVariant

This API provides support to develop a multi-variant modelling in OMSimulator [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd]. When duplicating a variant, the new variant becomes the current variant and all the changes made by the users are applied to the new variants only, and all the ssv and ssm resources associated with the new variant will be given new name based on the variant name provided by the user. This allows the bundling of multiple variants of a system structure definition referencing a similar set of packaged resources as a single SSP. However there must still be one SSD file named SystemStructure.ssd at the root of the ZIP archive which will be considered as default variant.

```
status = oms.duplicateVariant(crefA, crefB)
```

An example of creating a multi-variant modelling is presented below

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_setReal("model.root.A.param1", "10")
oms_duplicateVariant("model", "varB")
oms_addSubModel("varB.root.B" ,"B.fmu")
```

(continues on next page)

(continued from previous page)

```
oms_setReal("varB.root.A.param2", "20")
oms_export("varB", "variant.ssp")
```

The variant.ssp file will have the following structure

```
Variant.ssp
    SystemStructure.ssd
    varB.ssd
    resources\
        A.fmu
        B.fmu
```

14.6.20 export

Exports a composite model to a SPP file.

```
status = oms.export(cref, filename)
```

14.6.21 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
status = oms.exportDependencyGraphs(cref, initialization, event, simulation)
```

14.6.22 exportSSMTemplate

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

```
status = oms.exportSSMTemplate(cref, filename)
```

14.6.23 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.

```
status = oms.exportSSVTemplate(cref, filename)
```

14.6.24 exportSnapshot

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms.exportSnapshot(cref)
```

14.6.25 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

```
oms.freeMemory(obj)
```

14.6.26 getBoolean

Get boolean value of given signal.

```
value, status = oms.getBoolean(cref)
```

14.6.27 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

```
value, status = oms.getDirectionalDerivative(cref)
```

14.6.28 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
stepSize, status = oms.getFixedStepSize(cref)
```

14.6.29 getInteger

Get integer value of given signal.

```
value, status = oms.getInteger(cref)
```

14.6.30 getReal

Get real value.

```
value, status = oms.getReal(cref)
```

14.6.31 getResultFile

Gets the result filename and buffer size of the given model cref.

```
filename, bufferSize, status = oms.getResultFile(cref)
```

14.6.32 getSolver

Gets the selected solver method of the given system.

```
solver, status = oms.getSolver(cref)
```

14.6.33 getStartTime

Get the start time from the model.

```
startTime, status = oms.getStartTime(cref)
```

14.6.34 getStopTime

Get the stop time from the model.

```
stopTime, status = oms.getStopTime(cref)
```

14.6.35 getString

Get string value.

Memory is allocated for *value*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
value, status = oms.getString(cref)
```

14.6.36 getSubModelPath

Returns the path of a given component.

```
path, status = oms.getSubModelPath(cref)
```

14.6.37 getSystemType

Gets the type of the given system.

```
type, status = oms.getSystemType(cref)
```

14.6.38 getTime

Get the current simulation time from the model.

```
time, status = oms.getTime(cref)
```

14.6.39 getTolerance

Gets the tolerance of a given system or component.

```
relativeTolerance, status = oms.getTolerance(cref)
```

14.6.40 getVariableStepSize

Gets the step size parameters.

```
initialStepSize, minimumStepSize, maximumStepSize, status = oms.
↪getVariableStepSize(cref)
```

14.6.41 getVersion

Returns the library's version string.

```
oms = OMSimulator()
oms.getVersion()
```

14.6.42 importFile

Imports a composite model from a SSP file.

```
cref, status = oms.importFile(filename)
```

14.6.43 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
newCref, status = oms.importSnapshot(cref, snapshot)
```

14.6.44 initialize

Initializes a composite model.

```
status = oms.initialize(cref)
```

14.6.45 instantiate

Instantiates a given composite model.

```
status = oms.instantiate(cref)
```

14.6.46 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms.list(cref)
```

14.6.47 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms.listUnconnectedConnectors(cref)
```

14.6.48 listVariants

This API shows the number of variants available [(e.g). SystemStructure.ssd, VarA.ssd, VarB.ssd] from a ssp file.

```
status = oms.listVariants(cref)
```

An example for finding the number of available variants in a ssp file

```
oms_newModel("model")
oms_addSystem("model.root", "system_wc")
oms_addSubModel("model.root.A", "A.fmu")
oms_duplicateVariant("model", "varA")
oms_duplicateVariant("varA", "varB")

oms_listVariants("varB")
```

The API will list the available variants like below

```
<oms:Variants>
  <oms:variant name="model" />
  <oms:variant name="varB" />
  <oms:variant name="varA" />
</oms:Variants>
```

14.6.49 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
newCref, status = oms.loadSnapshot(cref, snapshot)
```

14.6.50 newModel

Creates a new and yet empty composite model.

```
status = oms.newModel(cref)
```

14.6.51 newResources

Adds a new empty resources to the SSP. The resource file is a ".ssv" file where the parameter values set by the users using "oms_setReal()", "oms_setInteger()" and "oms_setReal()" are writtern to the file. Currently only ".ssv" files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of cref. (e.g) ":root.ssv"

```
status = oms.newResources(cref)

## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.newModel("newResources")

oms.addSystem("newResources.root", oms_system_wc)
oms.addConnector("newResources.root.Input1", oms.input, oms_signal_type_real)
oms.addConnector("newResources.root.Input2", oms.input, oms_signal_type_real)

## add Top level resources, the filename is provided using the colon suffix ":root.ssv
↪"
oms.newResources("newResources.root:root.ssv")
oms.setReal("newResources.root.Input1", 10)
## export the ssp with new resources
oms.export("newResources", "newResources.ssp")
```

14.6.52 referenceResources

Switches the references of ".ssv" and ".ssm" in a SSP file. Referencing of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1) Referencing only the ".ssv" file.

2) Referencing both the ".ssv" along with the ".ssm" file.

This API should be used in combination with "oms_deleteResources".To switch with a new reference, the old reference must be deleted first using "oms_deleteResources" and then reference with new resources.

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file, then the reference of ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" seperately as

the ssm file is binded to a ssv file. Hence it is not possible to switch the reference of ".ssm" file alone. So inorder to switch the reference of ".ssm" file, the users need to bind the reference of ".ssm" file along with the ".ssv".

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) ":root.ssv", and the ".ssm" file is optional and is provided by the user as the second argument to the API.

```
status = oms.referenceResources(cref, ssmFile)

## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("referenceResources1.ssp")
## delete only the references in ".ssd" file
oms.deleteResources("referenceResources1.root:root.ssv")
## usage-1 switch with new references, only ssv file
oms.referenceResources("referenceResources1.root:Config1.ssv")
## usage-2 switch with new references, both ssv and ssm file
oms.referenceResources("referenceResources1.root:Config1.ssv", "Config1.ssm")
```

14.6.53 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
status = oms.removeSignalsFromResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.6.54 rename

Renames a model, system, or component.

```
status = oms.rename(cref, newCref)
```

14.6.55 replaceSubModel

Replaces an existing fmu component, with a new component provided by the user, When replacing the fmu checks are made in all ssp concepts like in ssd, ssv and ssm, so that connections and parameter settings are not lost. It is possible that the namings of inputs and parameters match, but the start values might have been changed, in such cases new start values will be applied in ssd, ssv and ssm. In case if the Types of inputs and outputs and parameters differed, then the variables are updated according to the new changes and the connections will be removed with warning messages to user. In case when replacing a fmu, if the fmu contains parameter mapping associated with the ssv file, then only the ssm file entries are updated and the start values in the ssv files will not be changed.

```
status = oms.replaceSubModel(cref, fmuPath)
```

> It is possible to import an partially developed fmu (i.e contains only modeldescription.xml without any binaries) in OMSimulator, and later can be replaced with a fully develped fmu. An example to use the API, oms_addSubModel("model.root.A", "../resources/replaceA.fmu") oms_export("model", "test.ssp") oms_import("test.ssp") oms_replaceSubModel("model.root.A", "../resources/replaceA_extended.fmu")

14.6.56 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
status = oms.reset(cref)
```

14.6.57 setBoolean

Sets the value of a given boolean signal.

```
status = oms.setBoolean(cref, value)
```

14.6.58 setCommandLineOption

Sets special flags.

```
status = oms.setCommandLineOption(cmd)
```

Available flags:

```
usage: OMSimulator [-h] [--validate] [--version]
                   [--addParametersToCSV | --no-addParametersToCSV]
                   [--algLoopSolver {fixedpoint,kinsol}] [--clearAllOptions]
                   [--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS]
                   [--CVODEMaxNLSFailures CVODEMAXNLSFAILURES]
                   [--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS]
                   [--CVODEMaxSteps CVODEMAXSTEPS]
                   [--deleteTempFiles | --no-deleteTempFiles]
                   [--directionalDerivatives | --no-directionalDerivatives]
                   [--dumpAlgLoops | --no-dumpAlgLoops]
                   [--emitEvents | --no-emitEvents]
                   [--ignoreInitialUnknowns | --no-ignoreInitialUnknowns]
                   [--initialStepSize INITIALSTEPSIZE]
                   [--inputExtrapolation | --no-inputExtrapolation]
                   [--intervals INTERVALS] [--logFile LOGFILE]
                   [--logLevel LOGLEVEL] [--master MASTER]
                   [--maxEventIteration MAXEVENTITERATION]
                   [--maxLoopIteration MAXLOOPITERATION]
                   [--minimumStepSize MINIMUMSTEPSIZE] [--mode {cs,me}]
                   [--numProcs NUMPROCS] [--progressBar | --no-progressBar]
                   [--realTime | --no-realTime] [--resultFile RESULTFILE]
                   [--skipCSVHeader | --no-skipCSVHeader]
                   [--solver {euler,cvode}] [--solverStats | --no-solverStats]
                   [--startTime STARTTIME] [--stepSize STEPSIZE]
                   [--stopTime STOPTIME] [--stripRoot | --no-stripRoot]
                   [--suppressPath | --no-suppressPath] [--tempDir TEMPDIR]
                   [--timeout TIMEOUT] [--tolerance TOLERANCE]
                   [--wallTime | --no-wallTime] [--workingDir WORKINGDIR]
                   [--zeroNominal | --no-zeroNominal]
                   model

Command line entry point for OMSimulator. Allows running a model file
directly, e.g.: OMSimulator test.ssp OMSimulator model.fmu without having to
write a driver script. All simulation settings (start/stop time, result file,
tolerance, ...) are taken from the model itself; use
--resultFile/--startTime/--stopTime/--tolerance/ --stepSize to override them.
For FMUs that export both model exchange and co-simulation, use --mode to pick
which one to run. Pass --stripRoot to drop the "model.root" prefix from
exported signal names. Files are validated against their schema (FMI for .fmu,
SSP for .ssp) before simulation; pass --validate to only validate the file and
skip simulation e.g.: OMSimulator --validate test.ssp OMSimulator --validate
model.fmu

positional arguments:
```

(continues on next page)

(continued from previous page)

```
  model                 Path to a .ssp or .fmu file to simulate

options:
  -h, --help            show this help message and exit
  --validate            Only validate the file against its schema; do not
                        simulate
  --version             show program's version number and exit
  --addParametersToCSV, --no-addParametersToCSV
                        Export parameters to a .csv file (default: false)
  --algLoopSolver {fixedpoint,kinsol}
                        Specifies the loop solver method used for algebraic
                        loops spanning multiple components (default: kinsol)
  --clearAllOptions     Reset all flags to their default values
  --CVODEMaxErrTestFails CVODEMAXERRTESTFAILS
                        Maximum number of error test failures for CVODE
                        (default: 100)
  --CVODEMaxNLSFailures CVODEMAXNLSFAILURES
                        Maximum number of nonlinear convergence failures for
                        CVODE (default: 100)
  --CVODEMaxNLSIterations CVODEMAXNLSITERATIONS
                        Maximum number of nonlinear solver iterations for
                        CVODE (default: 5)
  --CVODEMaxSteps CVODEMAXSTEPS
                        Maximum number of steps for CVODE (default: 1000)
  --deleteTempFiles, --no-deleteTempFiles
                        Delete temporary files as soon as they are no longer
                        needed (default: true)
  --directionalDerivatives, --no-directionalDerivatives
                        Use directional derivatives to calculate the Jacobian
                        for algebraic loops (default: true)
  --dumpAlgLoops, --no-dumpAlgLoops
                        Dump information for algebraic loops (default: false)
  --emitEvents, --no-emitEvents
                        Emit events during simulation (default: true)
  --ignoreInitialUnknowns, --no-ignoreInitialUnknowns
                        Ignore initial unknowns from the modelDescription.xml
                        (default: false)
  --initialStepSize INITIALSTEPSIZE
                        Specify the initial step size (default: 1e-06)
  --inputExtrapolation, --no-inputExtrapolation
                        Enable input extrapolation using derivative
                        information (default: false)
  --intervals INTERVALS
                        Specify the number of communication points (arg > 1)
                        (default: 500)
  --logFile LOGFILE     Specify the log file (default: stdout)
  --logLevel LOGLEVEL   Set the log level (0: default, 1: debug, 2:
                        debug+trace) (default: 0)
  --master MASTER       Specify the master algorithm (ma) (default: ma)
  --maxEventIteration MAXEVENTITERATION
                        Specify the maximum number of iterations for handling
                        a single event (default: 100)
  --maxLoopIteration MAXLOOPITERATION
                        Specify the maximum number of iterations for solving
                        algebraic loops between system-level components.
                        Internal algebraic loops of components are not
```

(continues on next page)

(continued from previous page)

```
                      affected. (default: 10)
  --minimumStepSize MINIMUMSTEPSIZE
                      Specify the minimum step size (default: 1e-12)
  --mode {cs,me}      Force 'cs' (co-simulation) or 'me' (model exchange)
                      for FMUs that export both kinds (.fmu only) (default:
                      co-simulation)
  --numProcs NUMPROCS Specify the maximum number of processors to use
                      (0=auto, 1=default) (default: 1)
  --progressBar, --no-progressBar
                      Show a progress bar for the simulation progress in the
                      terminal (default: false)
  --realTime, --no-realTime
                      Enable experimental feature for (soft) real-time co-
                      simulation (default: false)
  --resultFile RESULTFILE
                      Specify the name of the output result file (default:
                      the model name plus '_res.mat')
  --skipCSVHeader, --no-skipCSVHeader
                      Skip the CSV delimiter row in the header of .csv
                      result files (default: true)
  --solver {euler,cvode}
                      Set the ODE solver for model-exchange FMUs (.fmu,
                      mode=me only) (default: cvode)
  --solverStats, --no-solverStats
                      Add solver stats to the result file, e.g., step size;
                      not supported for all solvers (default: false)
  --startTime STARTTIME
                      Specify the start time (default: from the model)
  --stepSize STEPSIZE Specify the (maximum) step size (.fmu only) (default:
                      from the model)
  --stopTime STOPTIME Specify the stop time (default: from the model)
  --stripRoot, --no-stripRoot
                      Remove the root system prefix from all exported
                      signals (default: false)
  --suppressPath, --no-suppressPath
                      Suppress path information in info messages; especially
                      useful for testing (default: true)
  --tempDir TEMPDIR   Specify the temporary directory (default: the working
                      directory)
  --timeout TIMEOUT   Specify the maximum allowed time in seconds for
                      running a simulation (default: 0 (disabled))
  --tolerance TOLERANCE
                      Specify the relative tolerance (default: from the
                      model)
  --wallTime, --no-wallTime
                      Add wall time information to the result file (default:
                      false)
  --workingDir WORKINGDIR
                      Specify the working directory (default: the current
                      directory)
  --zeroNominal, --no-zeroNominal
                      Accept FMUs with invalid nominal values and replace
                      the invalid nominal values with 1.0 (default: false)
```

14.6.59 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
status = oms.setFixedStepSize(cref, stepSize)
```

14.6.60 setInteger

Sets the value of a given integer signal.

```
status = oms.setInteger(cref, value)
```

14.6.61 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
status = oms.setLogFile(filename)
```

14.6.62 setLoggingInterval

Set the logging interval of the simulation.

```
status = oms.setLoggingInterval(cref, loggingInterval)
```

14.6.63 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
oms.setLoggingLevel(logLevel)
```

14.6.64 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
oms.setMaxLogFileSize(size)
```

14.6.65 setReal

Sets the value of a given real signal.

```
status = oms.setReal(cref, value)
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

14.6.66 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
status = oms.setRealInputDerivative(cref, value)
```

14.6.67 setResultFile

Set the result file of the simulation.

```
status = oms.setResultFile(cref, filename)
status = oms.setResultFile(cref, filename, bufferSize)
```

The creation of a result file is omitted if the filename is an empty string.

14.6.68 setSolver

Sets the solver method for the given system.

```
status = oms.setSolver(cref, solver)
```

solver	Type	Description
oms.solver_sc_explicit_euler	sc-system	Explicit euler with fixed step size
oms.solver_sc_cvode	sc-system	CVODE with adaptive stepsize
oms.solver_wc_ma	wc-system	default master algorithm with fixed step size
oms.solver_wc_mav	wc-system	master algorithm with adaptive stepsize
oms.solver_wc_mav2	wc-system	master algorithm with adaptive stepsize (double-step)

14.6.69 setStartTime

Set the start time of the simulation.

```
status = oms.setStartTime(cref, startTime)
```

14.6.70 setStopTime

Set the stop time of the simulation.

```
status = oms.setStopTime(cref, stopTime)
```

14.6.71 setString

Sets the value of a given string signal.

```
status = oms.setString(cref, value)
```

14.6.72 setTempDirectory

Set new temp directory.

```
status = oms.setTempDirectory(newTempDir)
```

14.6.73 setTolerance

Sets the tolerance for a given model or system.

```
status = oms.setTolerance(const char* cref,  double relativeTolerance)
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", relativeTolerance);
oms_setTolerance("model.root", relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

14.6.74 setUnit

Sets the unit of a given signal.

```
status = oms.setUnit(cref, value)
```

14.6.75 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
status = oms.getVariableStepSize(cref, initialStepSize, minimumStepSize,␣
↪maximumStepSize)
```

14.6.76 setWorkingDirectory

Set a new working directory.

```
status = oms.setWorkingDirectory(newWorkingDir)
```

14.6.77 simulate

Simulates a composite model.

```
status = oms.simulate(cref)
```

14.6.78 stepUntil

Simulates a composite model until a given time value.

```
status = oms.stepUntil(cref, stopTime)
```

14.6.79 terminate

Terminates a given composite model.

```
status = oms.terminate(cref)
```

This example uses a simple Modelica model and FMI-based batch simulation to approximate the value of pi.

A Modelica model is used to calculate two uniform distributed pseudo-random numbers between 0 and 1 based on a seed value and evaluates if the resulting coordinate is inside the unit circle or not.

```
model Circle
  parameter Integer globalSeed = 30020 "global seed to initialize random number␣
↪generator";
  parameter Integer localSeed = 614657 "local seed to initialize random number␣
↪generator";
  Real x;
  Real y;
  Boolean inside = x*x + y*y < 1.0;
protected
  Integer state128[4];
algorithm
  when initial() then
    state128 := Modelica.Math.Random.Generators.Xorshift128plus.
↪initialState(localSeed, globalSeed);
    (x, state128) := Modelica.Math.Random.Generators.Xorshift128plus.random(state128);
    (y, state128) := Modelica.Math.Random.Generators.Xorshift128plus.random(state128);
  end when;
  annotation(uses(Modelica(version="4.0.0")));
end Circle;
```

The model is then exported using the FMI interface and the generated FMU can then be used to run a million simulations in just a few seconds.

Listing 14.1: Batch simulation of the simple *Cirlce* model with different seed values. All OMSimulator-related comands are highlighted for convenience.

```
1  import math
2  import matplotlib.pyplot as plt
3  import OMSimulator as oms
4
5  # redirect logging to file and limit the file size to 65MB
6  oms.setLogFile('pi.log', 65)
7
8  model = oms.newModel('pi')
9  root = model.addSystem('root', oms.Types.System.SC)
10 root.addSubModel('circle', 'Circle.fmu')
11
12 model.resultFile = '' # no result file
13 model.instantiate()
14
15 results = list()
16 inside = 0
17
18 MIN = 100
19 MAX = 1000000
20 for i in range(0, MAX+1):
21   if i > 0:
22     model.reset()
23   model.setInteger('root.circle.globalSeed', i)
24   model.initialize()
25   if model.getBoolean("root.circle.inside"):
26     inside = inside + 1
```

(continues on next page)

(continued from previous page)

```
  if i >= MIN:
    results.append(4.0*inside/i)
model.terminate()
model.delete()

plt.plot([MIN, MAX], [math.pi, math.pi], 'r--', range(MIN, MAX+1), results)
plt.xscale('log')
plt.ylabel('Approximation of pi')
plt.savefig('pi.png')
```

The following figure shows the approximation of pi in relation to the number of samples.

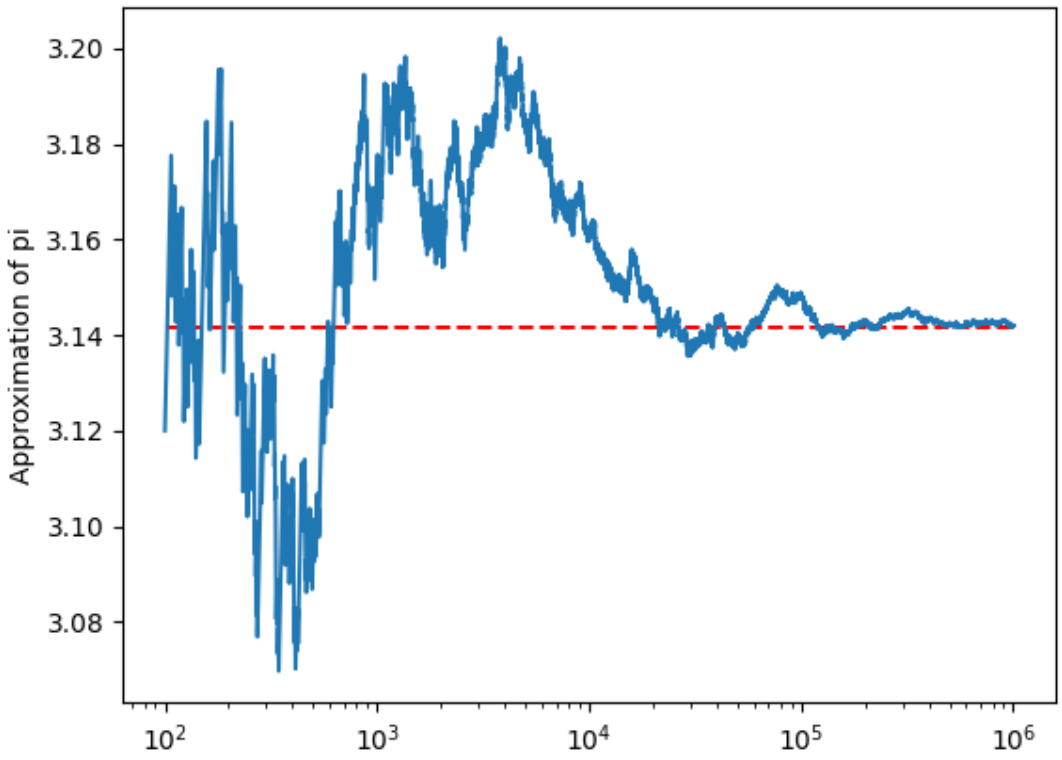


Figure 14.1: Results of the above batch simulation which approximates the value of pi

14.7 OpenModelicaScripting

This is a shared library that provides a OpenModelica Scripting interface for the OMSimulatorLib library.

14.7.1 Examples

```
loadOMSimulator();
oms_setTempDirectory("./temp/");
oms_newModel("model");
oms_addSystem("model.root", OpenModelica.Scripting.oms_system.oms_system_sc);

// instantiate FMUs
```

(continues on next page)

(continued from previous page)

```
oms_addSubModel("model.root.system1", "FMUs/System1.fmu");
oms_addSubModel("model.root.system2", "FMUs/System2.fmu");

// add connections
oms_addConnection("model.root.system1.y", "model.root.system2.u");
oms_addConnection("model.root.system2.y", "model.root.system1.u");

// simulation settings
oms_setResultFile("model", "results.mat");
oms_setStopTime("model", 0.1);
oms_setFixedStepSize("model.root", 1e-4);

oms_instantiate("model");
oms_setReal("model.root.system1.x_start", 2.5);

oms_initialize("model");
oms_simulate("model");
oms_terminate("model");
oms_delete("model");
unloadOMSimulator();
```

14.7.2 OpenModelica Scripting Commands

14.7.3 addBus

Adds a bus to a given component.

```
status := oms_addBus(cref);
```

14.7.4 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
status := oms_addConnection(crefA, crefB, suppressUnitConversion);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary. The third argument suppressUnitConversion is optional and the default value is *false* which allows automatic unit conversion between connections, if set to *true* then automatic unit conversion will be disabled.

14.7.5 addConnector

Adds a connector to a given component.

```
status := oms_addConnector(cref, causality, type);

The second argument "causality", should be any of the following,

"OpenModelica.Scripting.oms_causality.oms_causality_input"
"OpenModelica.Scripting.oms_causality.oms_causality_output"
"OpenModelica.Scripting.oms_causality.oms_causality_parameter"
"OpenModelica.Scripting.oms_causality.oms_causality_bidir"
"OpenModelica.Scripting.oms_causality.oms_causality_undefined"

The third argument type, should be any of the following,
```

(continues on next page)

(continued from previous page)

```
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_real"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_integer"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_boolean"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_string"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_enum"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_bus"
```

14.7.6 addConnectorToBus

Adds a connector to a bus.

```
status := oms_addConnectorToBus(busCref, connectorCref);
```

14.7.7 addSignalsToResults

Add all variables that match the given regex to the result file.

```
status := oms_addSignalsToResults(cref, regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.7.8 addSubModel

Adds a component to a system.

```
status := oms_addSubModel(cref, fmuPath);
```

14.7.9 addSystem

Adds a (sub-)system to a model or system.

```
status := oms_addSystem(cref, type);

The second argument type, should be any of the following,

"OpenModelica.Scripting.oms_system.oms_system_none"
"OpenModelica.Scripting.oms_system.oms_system_wc"
"OpenModelica.Scripting.oms_system.oms_system_sc"
```

14.7.10 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
status := oms_compareSimulationResults(filenameA, filenameB, var, relTol, absTol);
```

The following table describes the input values:

Input	Type	Description
filenameA	String	Name of first result file to compare.
filenameB	String	Name of second result file to compare.
var	String	Name of signal to compare.
relTol	Number	Relative tolerance.
absTol	Number	Absolute tolerance.

The following table describes the return values:

Type	Description
Integer	1 if the signal is considered as equal, 0 otherwise

14.7.11 copySystem

Copies a system.

```
status := oms_copySystem(source, target);
```

14.7.12 delete

Deletes a connector, component, system, or model object.

```
status := oms_delete(cref);
```

14.7.13 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
status := oms_deleteConnection(crefA, crefB);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

14.7.14 deleteConnectorFromBus

Deletes a connector from a given bus.

```
status := oms_deleteConnectorFromBus(busCref, connectorCref);
```

14.7.15 export

Exports a composite model to a SPP file.

```
status := oms_export(cref, filename);
```

14.7.16 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
status := oms_exportDependencyGraphs(cref, initialization, event, simulation);
```

14.7.17 exportSnapshot

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
(contents, status) := oms_exportSnapshot(cref);
```

14.7.18 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

This function is not needed for OpenModelicaScripting Interface

14.7.19 getBoolean

Get boolean value of given signal.

```
(value, status) := oms_getBoolean(cref);
```

14.7.20 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
(stepSize, status) := oms_setFixedStepSize(cref);
```

14.7.21 getInteger

Get integer value of given signal.

```
(value, status) := oms_getInteger(cref);
```

14.7.22 getModelState

Gets the model state of the given model cref.

```
(modelState, status) := oms_getModelState(cref);
```

14.7.23 getReal

Get real value.

```
(value, status) := oms_getReal(cref);
```

14.7.24 getSolver

Gets the selected solver method of the given system.

```
(solver, status) := oms_getSolver(cref);
```

14.7.25 getStartTime

Get the start time from the model.

```
(startTime, status) := oms_getStartTime(cref);
```

14.7.26 getStopTime

Get the stop time from the model.

```
(stopTime, status) := oms_getStopTime(cref);
```

14.7.27 getSubModelPath

Returns the path of a given component.

```
(path, status) := oms_getSubModelPath(cref);
```

14.7.28 getSystemType

Gets the type of the given system.

```
(type, status) := oms_getSystemType(cref);
```

14.7.29 getTime

Get the current simulation time from the model.

```
(time, status) := oms_getTime(cref);
```

14.7.30 getTolerance

Gets the tolerance of a given system or component.

```
(relativeTolerance, status) := oms_getTolerance(cref);
```

14.7.31 getVariableStepSize

Gets the step size parameters.

```
(initialStepSize, minimumStepSize, maximumStepSize, status) := oms_
↪getVariableStepSize(cref);
```

14.7.32 getVersion

Returns the library's version string.

```
version := oms_getVersion();
```

14.7.33 importFile

Imports a composite model from a SSP file.

```
(cref, status) := oms_importFile(filename);
```

14.7.34 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
status := oms_importSnapshot(cref, snapshot);
```

14.7.35 initialize

Initializes a composite model.

```
status := oms_initialize(cref);
```

14.7.36 instantiate

Instantiates a given composite model.

```
status := oms_instantiate(cref);
```

14.7.37 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
(contents, status) := oms_list(cref);
```

14.7.38 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
(contents, status) := oms_listUnconnectedConnectors(cref);
```

14.7.39 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```
status := oms_loadSnapshot(cref, snapshot);
```

14.7.40 newModel

Creates a new and yet empty composite model.

```
status := oms_newModel(cref);
```

14.7.41 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
status := oms_removeSignalsFromResults(cref, regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

14.7.42 rename

Renames a model, system, or component.

```
status := oms_rename(cref, newCref);
```

14.7.43 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
status := oms_reset(cref);
```

14.7.44 setBoolean

Sets the value of a given boolean signal.

```
status := oms_setBoolean(cref, value);
```

14.7.45 setCommandLineOption

Sets special flags.

```
status := oms_setCommandLineOption(cmd);
```

Available flags:

```
usage: OMSimulator [-h] [--validate] [--version]
                   [--addParametersToCSV | --no-addParametersToCSV]
                   [--algLoopSolver {fixedpoint,kinsol}] [--clearAllOptions]
                   [--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS]
                   [--CVODEMaxNLSFailures CVODEMAXNLSFAILURES]
                   [--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS]
                   [--CVODEMaxSteps CVODEMAXSTEPS]
                   [--deleteTempFiles | --no-deleteTempFiles]
                   [--directionalDerivatives | --no-directionalDerivatives]
                   [--dumpAlgLoops | --no-dumpAlgLoops]
                   [--emitEvents | --no-emitEvents]
                   [--ignoreInitialUnknowns | --no-ignoreInitialUnknowns]
                   [--initialStepSize INITIALSTEPSIZE]
                   [--inputExtrapolation | --no-inputExtrapolation]
                   [--intervals INTERVALS] [--logFile LOGFILE]
                   [--logLevel LOGLEVEL] [--master MASTER]
                   [--maxEventIteration MAXEVENTITERATION]
                   [--maxLoopIteration MAXLOOPITERATION]
                   [--minimumStepSize MINIMUMSTEPSIZE] [--mode {cs,me}]
                   [--numProcs NUMPROCS] [--progressBar | --no-progressBar]
                   [--realTime | --no-realTime] [--resultFile RESULTFILE]
                   [--skipCSVHeader | --no-skipCSVHeader]
                   [--solver {euler,cvode}] [--solverStats | --no-solverStats]
                   [--startTime STARTTIME] [--stepSize STEPSIZE]
                   [--stopTime STOPTIME] [--stripRoot | --no-stripRoot]
                   [--suppressPath | --no-suppressPath] [--tempDir TEMPDIR]
                   [--timeout TIMEOUT] [--tolerance TOLERANCE]
                   [--wallTime | --no-wallTime] [--workingDir WORKINGDIR]
                   [--zeroNominal | --no-zeroNominal]
                   model

Command line entry point for OMSimulator. Allows running a model file
directly, e.g.: OMSimulator test.ssp OMSimulator model.fmu without having to
write a driver script. All simulation settings (start/stop time, result file,
tolerance, ...) are taken from the model itself; use
--resultFile/--startTime/--stopTime/--tolerance/ --stepSize to override them.
For FMUs that export both model exchange and co-simulation, use --mode to pick
which one to run. Pass --stripRoot to drop the "model.root" prefix from
exported signal names. Files are validated against their schema (FMI for .fmu,
SSP for .ssp) before simulation; pass --validate to only validate the file and
skip simulation e.g.: OMSimulator --validate test.ssp OMSimulator --validate
model.fmu

positional arguments:
  model                 Path to a .ssp or .fmu file to simulate

options:
```

(continues on next page)

(continued from previous page)

```
-h, --help            show this help message and exit
--validate            Only validate the file against its schema; do not
                      simulate
--version             show program's version number and exit
--addParametersToCSV, --no-addParametersToCSV
                      Export parameters to a .csv file (default: false)
--algLoopSolver {fixedpoint,kinsol}
                      Specifies the loop solver method used for algebraic
                      loops spanning multiple components (default: kinsol)
--clearAllOptions     Reset all flags to their default values
--CVODEMaxErrTestFails CVODEMAXERRTESTFAILS
                      Maximum number of error test failures for CVODE
                      (default: 100)
--CVODEMaxNLSFailures CVODEMAXNLSFAILURES
                      Maximum number of nonlinear convergence failures for
                      CVODE (default: 100)
--CVODEMaxNLSIterations CVODEMAXNLSITERATIONS
                      Maximum number of nonlinear solver iterations for
                      CVODE (default: 5)
--CVODEMaxSteps CVODEMAXSTEPS
                      Maximum number of steps for CVODE (default: 1000)
--deleteTempFiles, --no-deleteTempFiles
                      Delete temporary files as soon as they are no longer
                      needed (default: true)
--directionalDerivatives, --no-directionalDerivatives
                      Use directional derivatives to calculate the Jacobian
                      for algebraic loops (default: true)
--dumpAlgLoops, --no-dumpAlgLoops
                      Dump information for algebraic loops (default: false)
--emitEvents, --no-emitEvents
                      Emit events during simulation (default: true)
--ignoreInitialUnknowns, --no-ignoreInitialUnknowns
                      Ignore initial unknowns from the modelDescription.xml
                      (default: false)
--initialStepSize INITIALSTEPSIZE
                      Specify the initial step size (default: 1e-06)
--inputExtrapolation, --no-inputExtrapolation
                      Enable input extrapolation using derivative
                      information (default: false)
--intervals INTERVALS
                      Specify the number of communication points (arg > 1)
                      (default: 500)
--logFile LOGFILE     Specify the log file (default: stdout)
--logLevel LOGLEVEL   Set the log level (0: default, 1: debug, 2:
                      debug+trace) (default: 0)
--master MASTER       Specify the master algorithm (ma) (default: ma)
--maxEventIteration MAXEVENTITERATION
                      Specify the maximum number of iterations for handling
                      a single event (default: 100)
--maxLoopIteration MAXLOOPITERATION
                      Specify the maximum number of iterations for solving
                      algebraic loops between system-level components.
                      Internal algebraic loops of components are not
                      affected. (default: 10)
--minimumStepSize MINIMUMSTEPSIZE
                      Specify the minimum step size (default: 1e-12)
```

(continues on next page)

(continued from previous page)

```
  --mode {cs,me}        Force 'cs' (co-simulation) or 'me' (model exchange)
                        for FMUs that export both kinds (.fmu only) (default:
                        co-simulation)
  --numProcs NUMPROCS   Specify the maximum number of processors to use
                        (0=auto, 1=default) (default: 1)
  --progressBar, --no-progressBar
                        Show a progress bar for the simulation progress in the
                        terminal (default: false)
  --realTime, --no-realTime
                        Enable experimental feature for (soft) real-time co-
                        simulation (default: false)
  --resultFile RESULTFILE
                        Specify the name of the output result file (default:
                        the model name plus '_res.mat')
  --skipCSVHeader, --no-skipCSVHeader
                        Skip the CSV delimiter row in the header of .csv
                        result files (default: true)
  --solver {euler,cvode}
                        Set the ODE solver for model-exchange FMUs (.fmu,
                        mode=me only) (default: cvode)
  --solverStats, --no-solverStats
                        Add solver stats to the result file, e.g., step size;
                        not supported for all solvers (default: false)
  --startTime STARTTIME
                        Specify the start time (default: from the model)
  --stepSize STEPSIZE   Specify the (maximum) step size (.fmu only) (default:
                        from the model)
  --stopTime STOPTIME   Specify the stop time (default: from the model)
  --stripRoot, --no-stripRoot
                        Remove the root system prefix from all exported
                        signals (default: false)
  --suppressPath, --no-suppressPath
                        Suppress path information in info messages; especially
                        useful for testing (default: true)
  --tempDir TEMPDIR     Specify the temporary directory (default: the working
                        directory)
  --timeout TIMEOUT     Specify the maximum allowed time in seconds for
                        running a simulation (default: 0 (disabled))
  --tolerance TOLERANCE
                        Specify the relative tolerance (default: from the
                        model)
  --wallTime, --no-wallTime
                        Add wall time information to the result file (default:
                        false)
  --workingDir WORKINGDIR
                        Specify the working directory (default: the current
                        directory)
  --zeroNominal, --no-zeroNominal
                        Accept FMUs with invalid nominal values and replace
                        the invalid nominal values with 1.0 (default: false)
```

14.7.46 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
status := oms_setFixedStepSize(cref, stepSize);
```

14.7.47 setInteger

Sets the value of a given integer signal.

```
status := oms_setInteger(cref, value);
```

14.7.48 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
status := oms_setLogFile(filename);
```

14.7.49 setLoggingInterval

Set the logging interval of the simulation.

```
status := oms_setLoggingInterval(cref, loggingInterval);
```

14.7.50 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
oms_setLoggingLevel(logLevel);
```

14.7.51 setReal

Sets the value of a given real signal.

```
status := oms_setReal(cref, value);
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

14.7.52 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
status := oms_setRealInputDerivative(cref, value);
```

14.7.53 setResultFile

Set the result file of the simulation.

```
status := oms_setResultFile(cref, filename);
status := oms_setResultFile(cref, filename, bufferSize);
```

The creation of a result file is omitted if the filename is an empty string.

14.7.54 setSolver

Sets the solver method for the given system.

```
status := oms_setSolver(cref, solver);

The second argument "solver" should be any of the following,

"OpenModelica.Scripting.oms_solver.oms_solver_none"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_min"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_explicit_euler"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_cvode"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_max"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_min"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_ma"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_mav"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_mav2"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_max"
```

14.7.55 setStartTime

Set the start time of the simulation.

```
status := oms_setStartTime(cref, startTime);
```

14.7.56 setStopTime

Set the stop time of the simulation.

```
status := oms_setStopTime(cref, stopTime);
```

14.7.57 setTempDirectory

Set new temp directory.

```
status := oms_setTempDirectory(newTempDir);
```

14.7.58 setTolerance

Sets the tolerance for a given model or system.

```
status := oms_setTolerance(const char* cref, double relativeTolerance);
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", relativeTolerance);
oms_setTolerance("model.root", relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

14.7.59 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
status := oms_getVariableStepSize(cref, initialStepSize, minimumStepSize,␣
↪maximumStepSize);
```

14.7.60 setWorkingDirectory

Set a new working directory.

```
status := oms_setWorkingDirectory(newWorkingDir);
```

14.7.61 simulate

Simulates a composite model.

```
status := oms_simulate(cref);
```

14.7.62 stepUntil

Simulates a composite model until a given time value.

```
status := oms_stepUntil(cref, stopTime);
```

14.7.63 terminate

Terminates a given composite model.

```
status := oms_terminate(cref);
```

14.8 Graphical Modelling

OMSimulator provides a graphical modelling environment through OMEdit, the OpenModelica Connection Editor. This feature requires a full OpenModelica installation that includes OMSimulator.

Composite models are imported and exported in the System Structure Description (SSD) format, which is part of the System Structure and Parameterization (SSP) standard.

See also FMI documentation and SSP documentation.

14.8.1 Architecture

OMEdit communicates with OMSimulator through a Python-based ZMQ server:

- **OMSimulatorGuiServer.py** — handles all GUI-driven requests (model management, element queries, solver settings, connections, etc.)
- **OMSimulatorSimulationServer.py** — handles simulation execution and result streaming

The GUI server is started automatically when the first SSP model is opened or created. A notification in the Messages Browser confirms the server is running, including the script path and ZMQ endpoint.

14.8.2 New SSP Model

A new and empty SSP model can be created from `File -> New -> SSP` menu item.

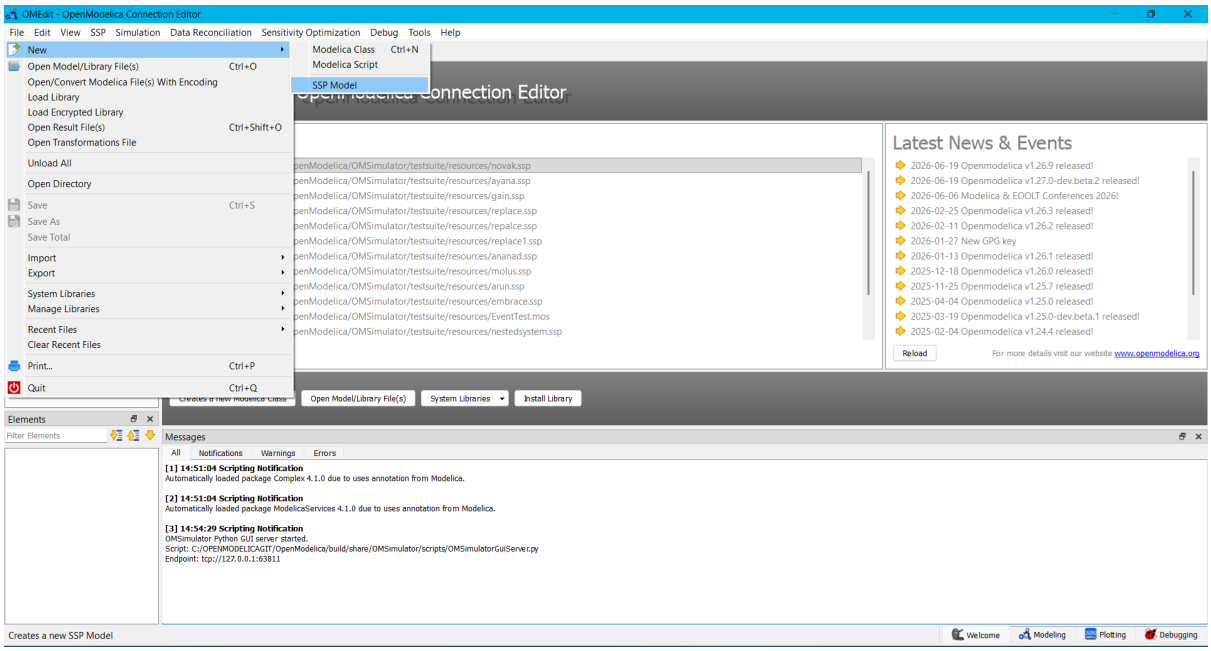


Figure 14.2: OMEdit: New SSP Model

A dialog opens to enter the model name and the name of the root system.

14.8.3 Open SSP Model

An existing SSP file (`.ssp`) can be opened from `File -> Open Model/Library`. If a model with the same name is already loaded, an error is reported and the file is not loaded again.

14.8.4 Add System and Sub-Systems

A root system is always created together with the model. Additional subsystems can be added inside the root system via `SSP -> Add System`.

A dialog opens to enter the name of the new system.

14.8.5 Add SubModel

A sub-model is typically an FMU, but it can also be a result file (CSV). To add a sub-model, select the target system in the Libraries Browser and choose `SSP -> Add SubModel`.

A file browser opens to select an FMU (`.fmu`) or result file (`.csv`). A dialog then opens to set the name for the new sub-model.

14.8.6 Replace SubModel

An existing sub-model can be replaced with a different FMU by right-clicking on the component in the diagram view and selecting `Replace SubModel` from the context menu.

A file browser opens to select the new FMU. The replacement is performed in two steps:

1. **Dry run** (`dryRun=true`) — OMEdit first performs a dry run without making any changes to the model. It checks for interface differences between the old and new FMU, such as added, removed, or modified connectors and parameters. If any changes are detected, they are reported in the Messages Browser and a warning dialog is shown. The model remains unchanged at this stage, allowing the user to review the impact before deciding to proceed.

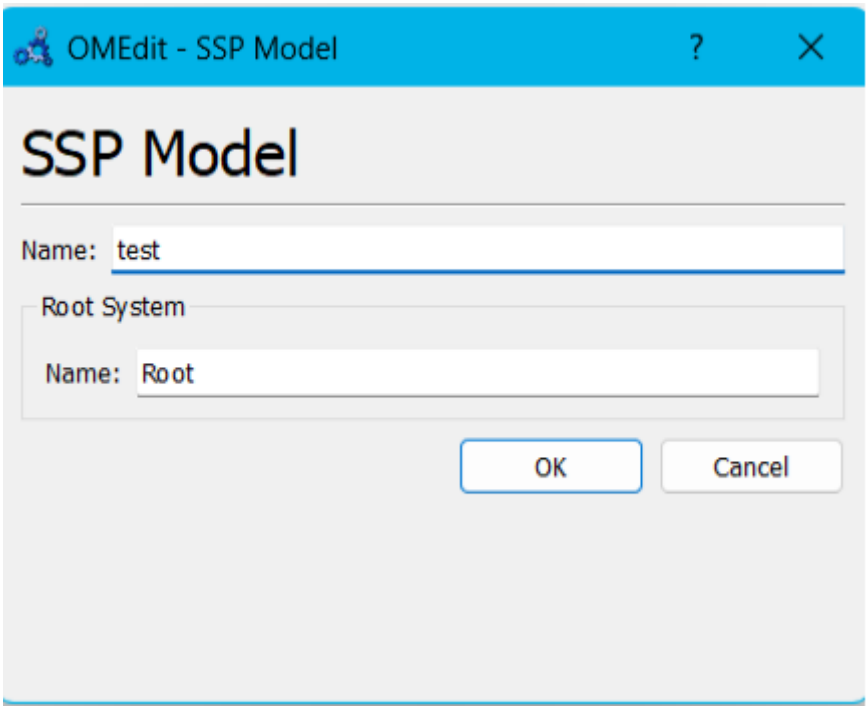


Figure 14.3: OMEdit: New SSP Model Dialog

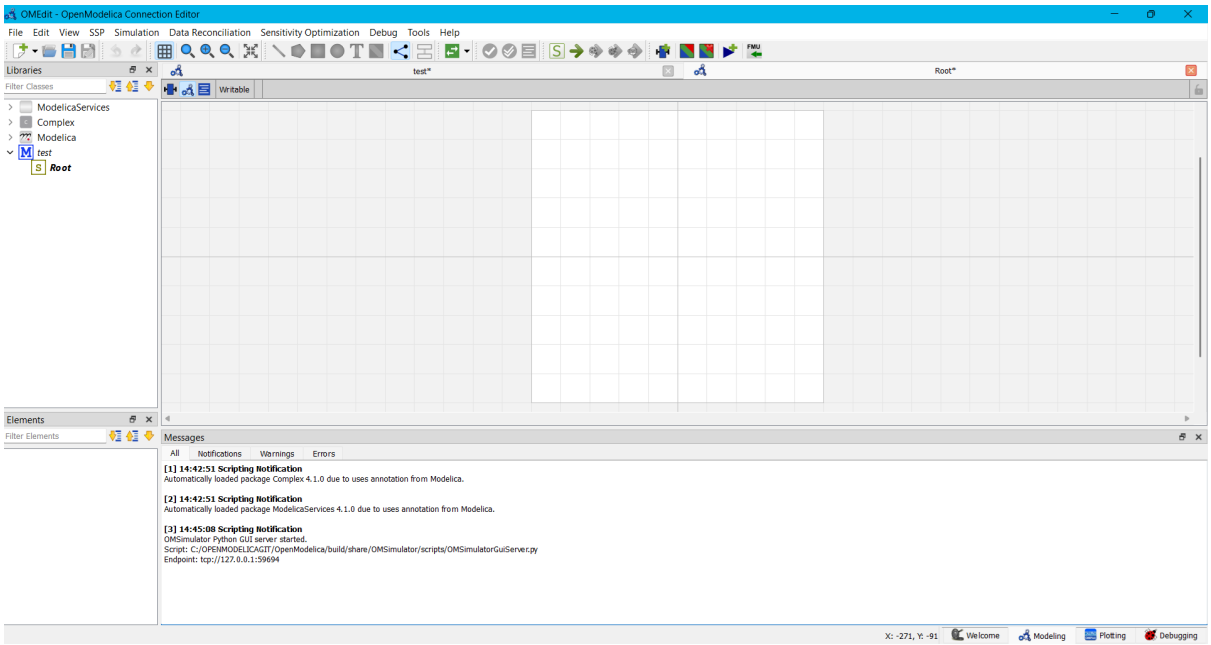


Figure 14.4: OMEdit: Newly created empty root system of SSP model

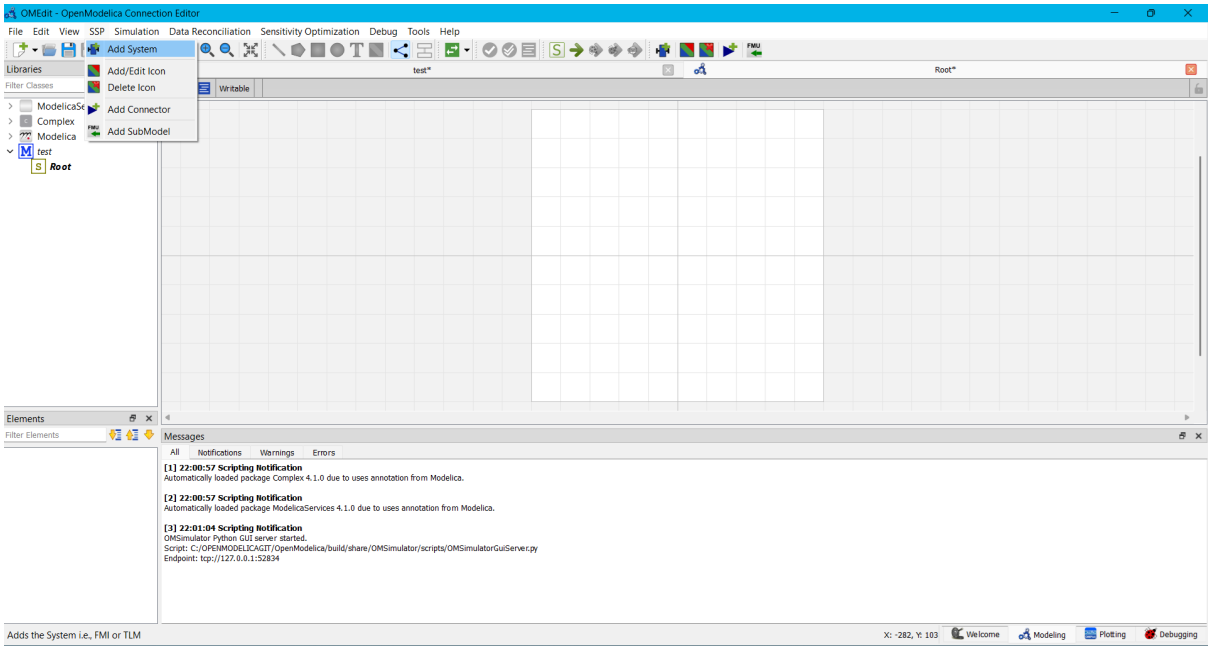


Figure 14.5: OMEdit: Add System

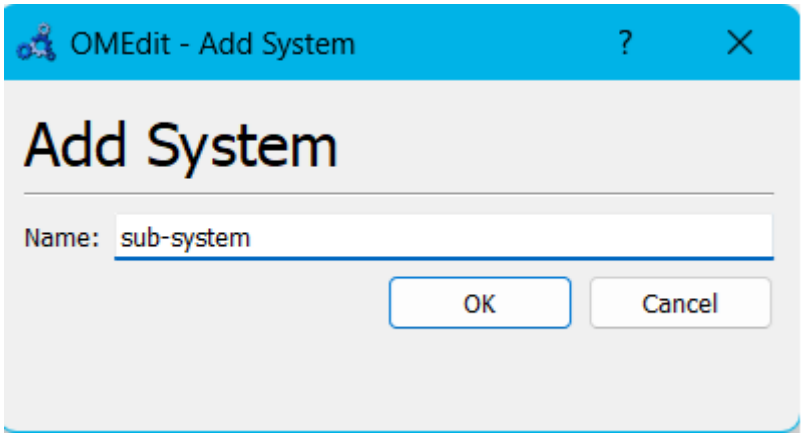


Figure 14.6: OMEdit: Add System Dialog

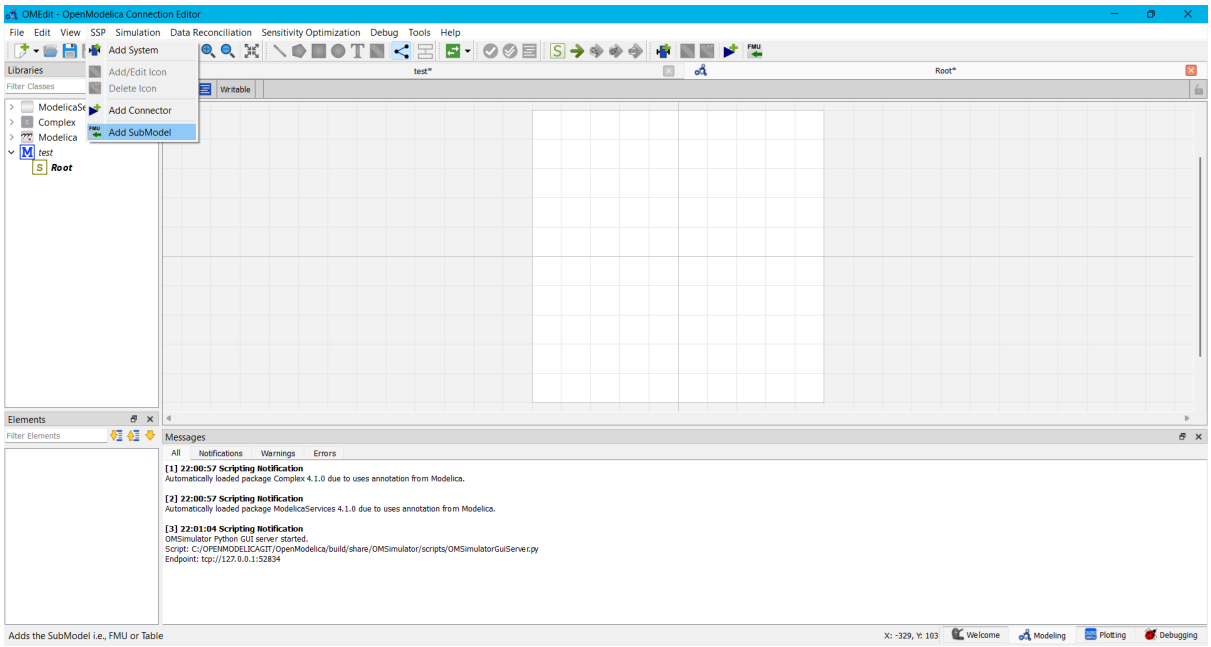


Figure 14.7: OMEdit: Add SubModel

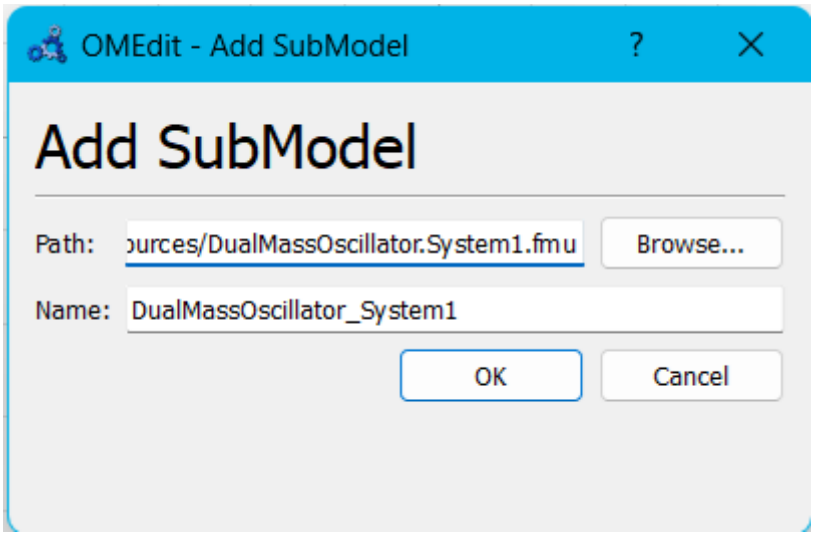


Figure 14.8: OMEdit: Add SubModel Dialog

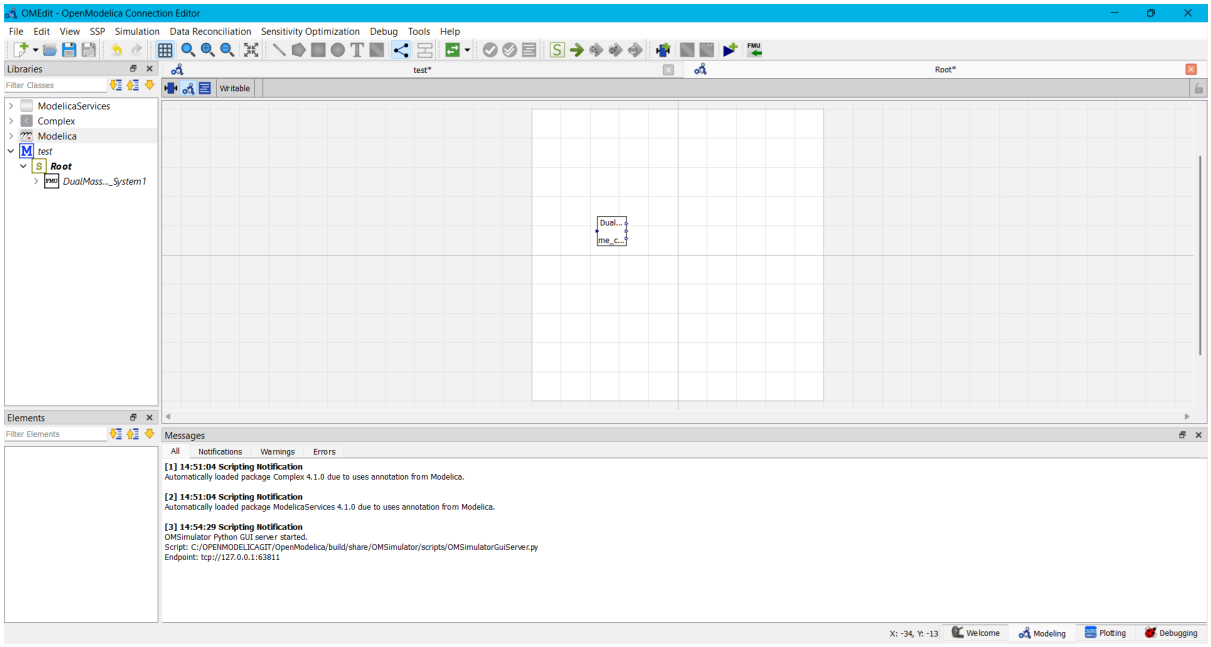


Figure 14.9: OMEdit: Root system with added FMU.

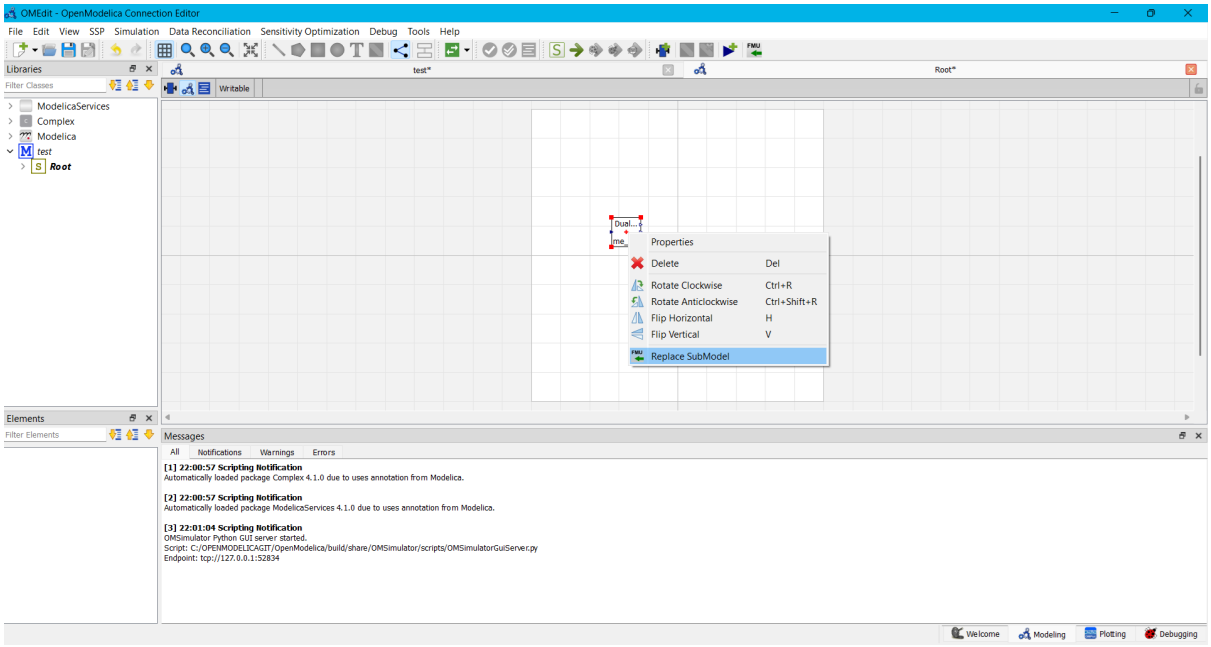


Figure 14.10: OMEdit: Replace SubModel Dialog.

2. **Replacement** (`dryRun=false`) — If the user confirms, the replacement is carried out and all detected interface changes are applied to the model. Existing connections to removed or renamed connectors are dropped, and new connectors are available for connecting.

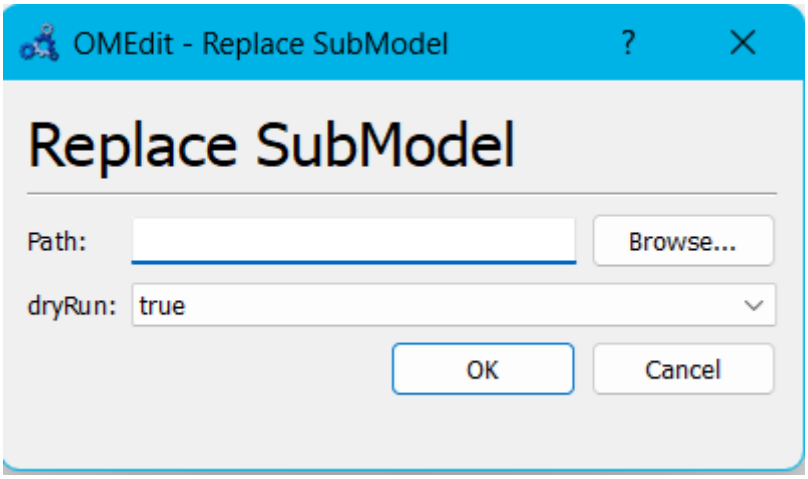


Figure 14.11: OMEdit: Replace SubModel context menu.

14.8.7 Simulation Setup

Select `Simulation -> Simulation Setup` to configure simulation parameters before running. The dialog has two tabs: **General** and **Solver Settings**.

The **General** tab covers start time, stop time, and result file settings.

The **Solver Settings** tab is divided into two parts:

Solver Configurations

This table lists the named solver configurations available in the model. Each configuration has a name and a method. Click **Add** to create a new solver configuration — a new row appears in the table with a default method of `oms-ma`. The method can be changed using the combo box in the Method column.

To fine-tune a solver's numerical parameters, select the row and click **Edit**. This opens the Solver Settings dialog:

The available fields depend on the solver method:

- **Fixed-step solvers** (`oms-ma`, `euler`) — Fixed Step Size is editable; Initial Step Size, Minimum Step Size, and Maximum Step Size are disabled.
- **Variable-step solvers** (`oms-mav`, `oms-mav-2`, `cvode`) — Initial Step Size, Minimum Step Size, Maximum Step Size, and Relative Tolerance are editable; Fixed Step Size is disabled.

Click **Remove** to delete the selected solver configuration.

Component Assignments

This table lists all FMU components in the model. Each component can be assigned one of the named solver configurations using the combo box in the Solver column. The available solvers are filtered by the FMI kind of each component:

- **Co-Simulation FMUs** — only co-simulation masters are selectable (`oms-ma`, `oms-mav`, `oms-mav-2`).
- **Model-Exchange FMUs** — only ODE integrators are selectable (`cvode`, `euler`).
- **me+cs FMUs** — all solvers are available.

Incompatible solvers are shown as disabled in the combo box with a tooltip explaining why. If no solver is assigned, `(none)` is used and the system default applies.

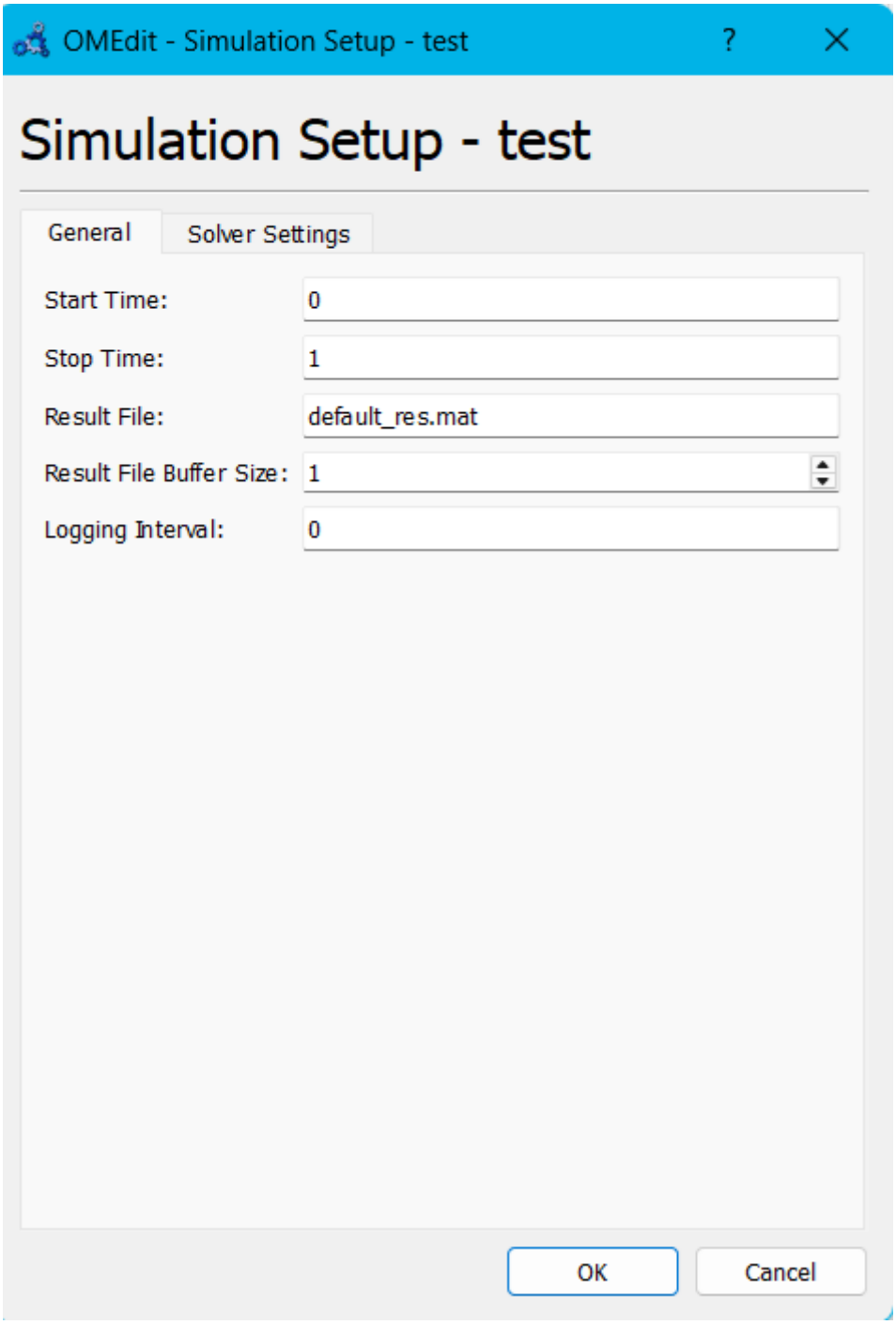


Figure 14.12: OMEdit: Simulation Setup — Solver Settings tab with a solver configuration.

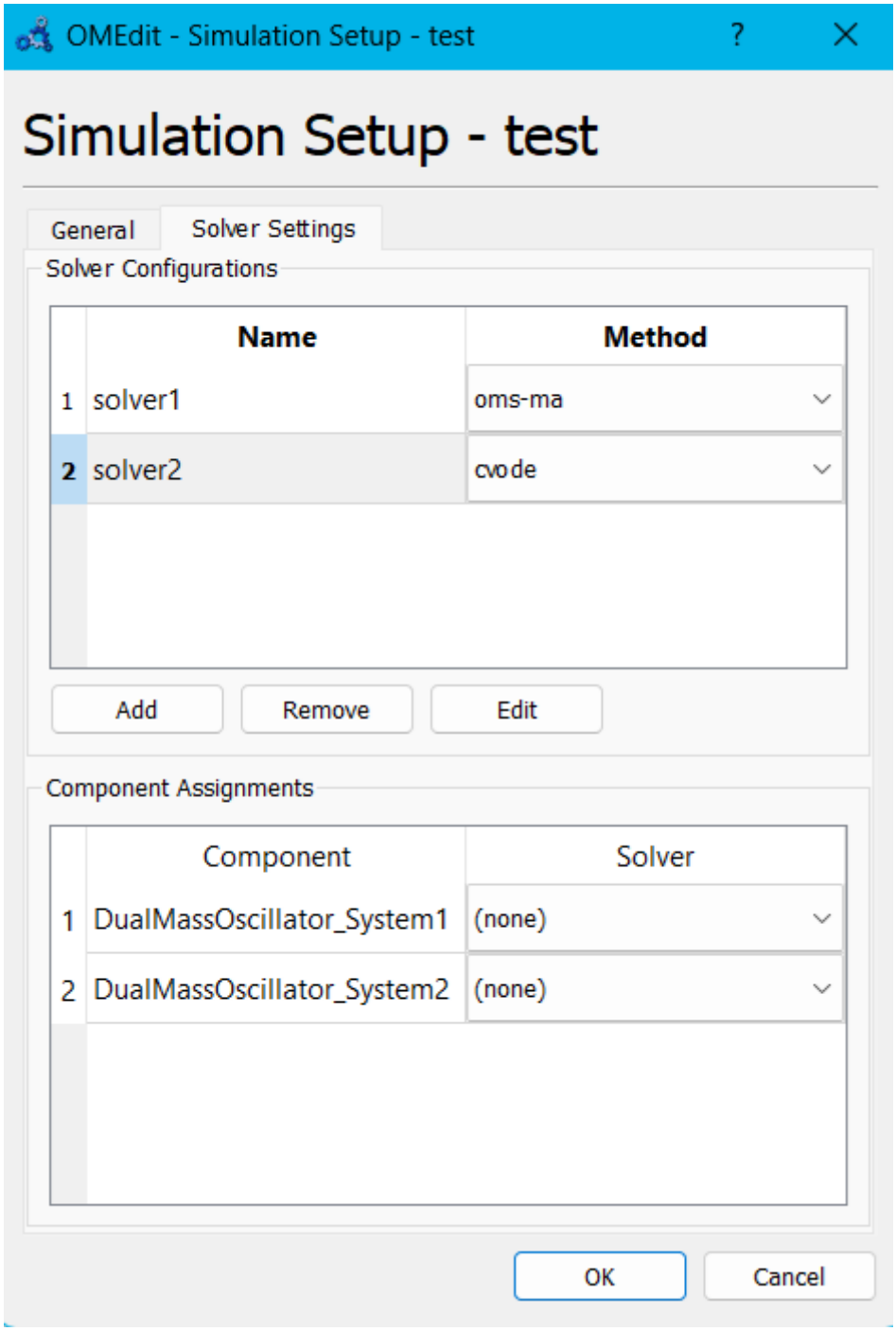


Figure 14.13: OMEdit: Simulation Setup — Solver Settings tab with a solver configuration.

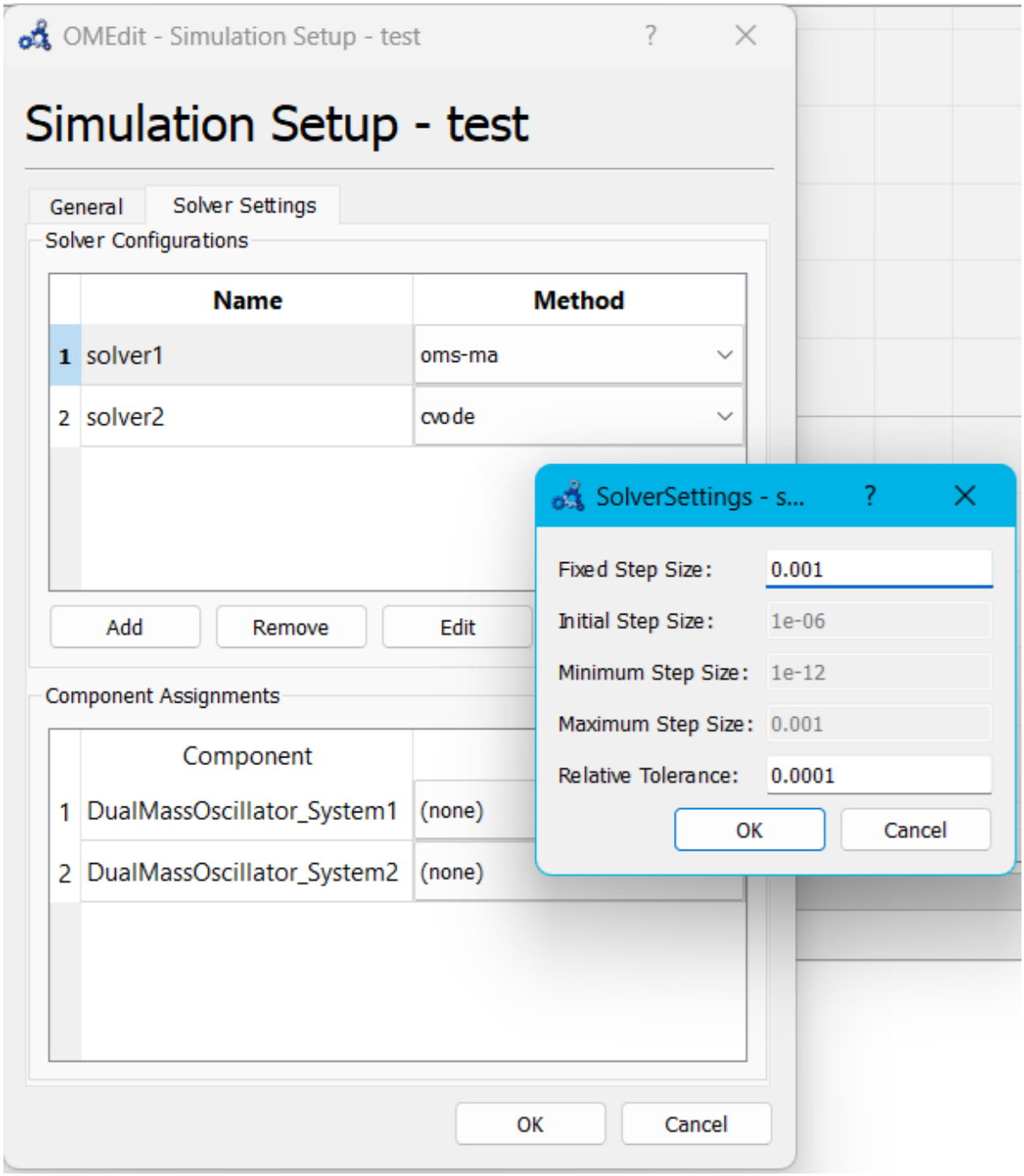


Figure 14.14: OMEdit: Solver Settings dialog.

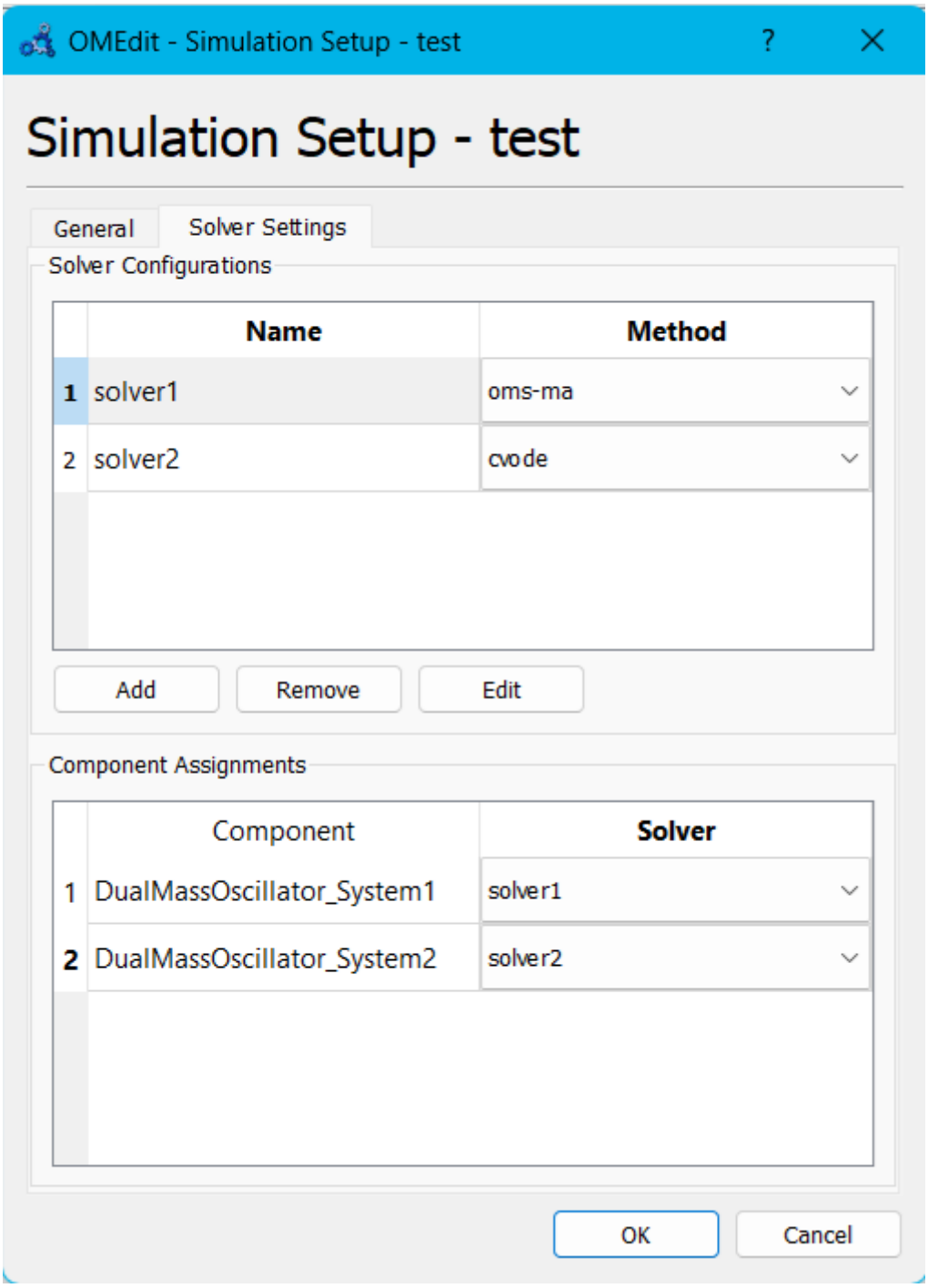


Figure 14.15: OMEdit: Solver Assignments in Components

14.8.8 Simulate

Select the simulate button (green arrow) or `Simulation -> Simulate` to run the SSP model. Results are streamed back in real time via the simulation server.

14.8.9 Dual Mass Oscillator Example

The dual mass oscillator example from the test suite demonstrates a typical SSP workflow. The model is split into two sub-models, each exported as an FMU, and then connected in an SSP.

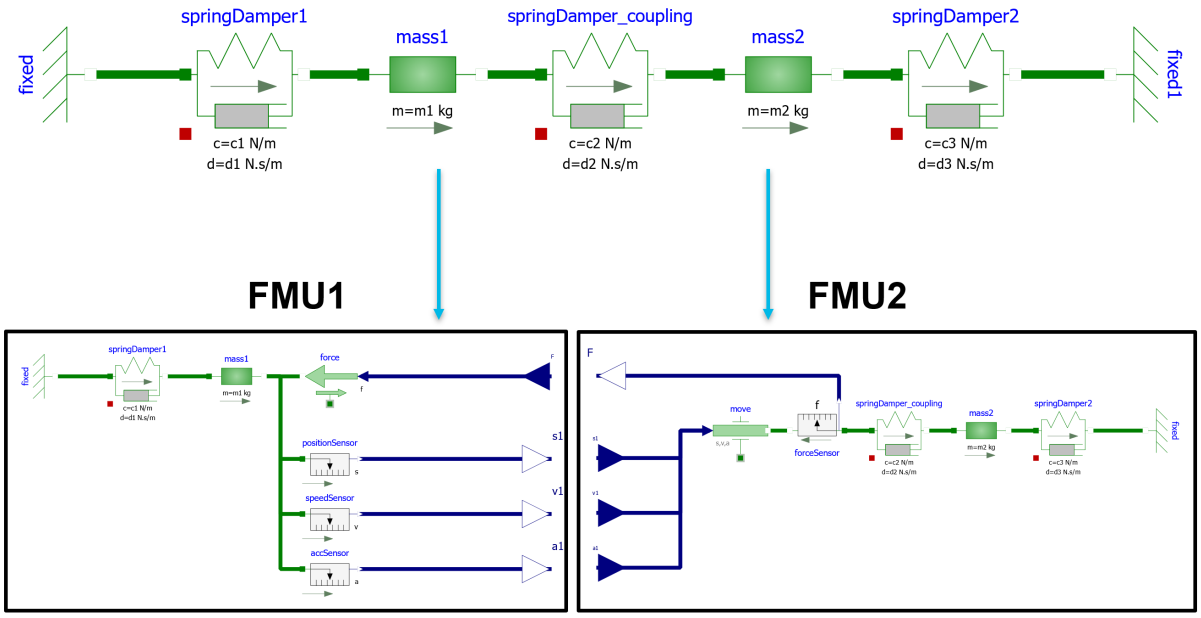


Figure 14.16: Dual mass oscillator Modelica model (diagram view) and FMUs

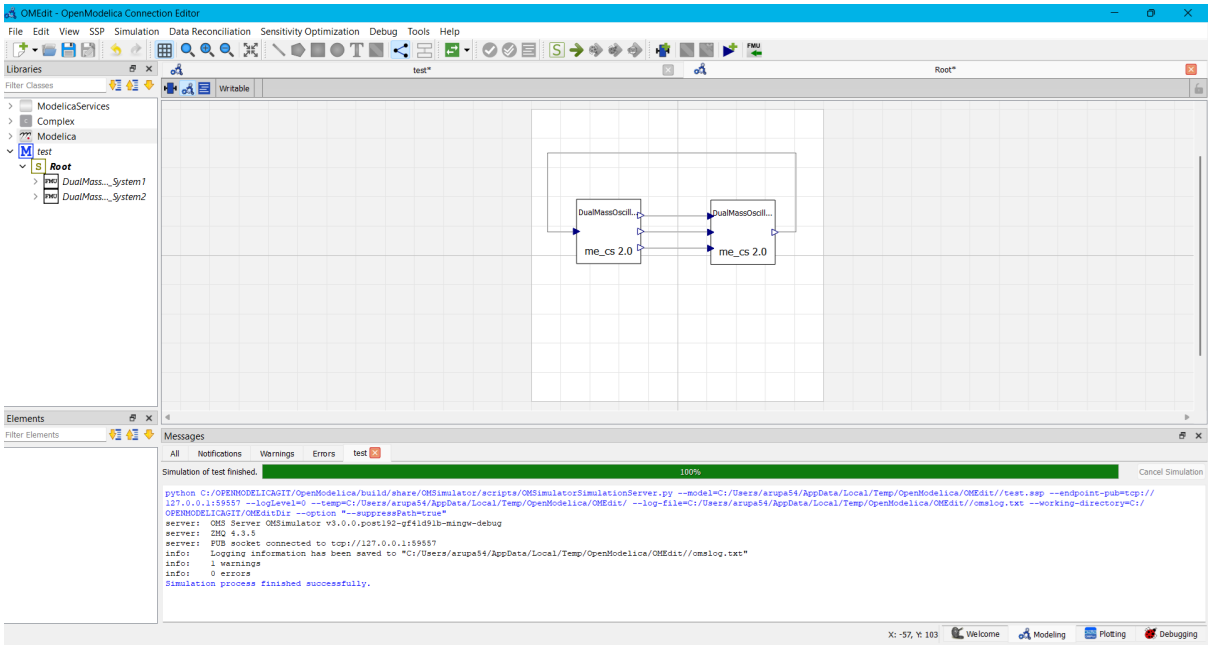


Figure 14.17: OMEdit: Simulate Dual Mass Oscillator SSP model

14.9 SSP Support

Composite models are imported and exported in the *System Structure Description (SSD)* format, which is part of the *System Structure and Parameterization (SSP)* standard.

14.9.1 Bus Connections

Bus connections are saved as annotations to the SSD file. Bus connectors are only allowed in weakly coupled and strongly coupled systems. Bus connections can exist in any system type. Bus connectors are used to hide SSD connectors and bus connections are used to hide existing SSD connections in the graphical user interface. It is not required that all connectors referenced in a bus are connected. One bus may be connected to multiple other buses, and also to SSD connectors.

The example below contains a root system with two subsystems, `WC1` and `WC2`. Bus connector `WC1.bus1` is connected to `WC2.bus2`. Bus connector `WC2.bus2` is also connected to SSD connector `WC1.C3`.

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription name="Test" version="Draft20180219">
  <ssd:System name="Root">
    <ssd:Elements>
      <ssd:System name="WC2">
        <ssd:Connectors>
          <ssd:Connector name="C1" kind="input" type="Real"/>
          <ssd:Connector name="C2" kind="output" type="Real"/>
        </ssd:Connectors>
        <ssd:Annotations>
          <ssc:Annotation type="org.openmodelica">
            <oms:Bus name="bus2">
              <oms:Signals>
                <oms:Signal name="C1"/>
                <oms:Signal name="C2"/>
              </oms:Signals>
            </oms:Bus>
          </ssc:Annotation>
        </ssd:Annotations>
      </ssd:System>
      <ssd:System name="WC1">
        <ssd:Connectors>
          <ssd:Connector name="C1" kind="output" type="Real"/>
          <ssd:Connector name="C2" kind="input" type="Real"/>
          <ssd:Connector name="C3" kind="input" type="Real"/>
        </ssd:Connectors>
        <ssd:Annotations>
          <ssc:Annotation type="org.openmodelica">
            <oms:Bus name="bus1">
              <oms:Signals>
                <oms:Signal name="C1"/>
                <oms:Signal name="C2"/>
              </oms:Signals>
            </oms:Bus>
          </ssc:Annotation>
        </ssd:Annotations>
      </ssd:System>
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection startElement="WC2" startConnector="C1"
                      endElement="WC1" endConnector="C1"/>
      <ssd:Connection startElement="WC2" startConnector="C2"
```

(continues on next page)

(continued from previous page)

```
                    endElement="WC1" endConnector="C2"/>
      <ssd:Connection startElement="WC2" startConnector="C2"
                    endElement="WC1" endConnector="C3"/>
    </ssd:Connections>
    <ssd:Annotations>
      <ssc:Annotation type="org.openmodelica">
        <oms:Connections>
          <oms:Connection startElement="WC1" startConnector="bus1"
                         endElement="WC2" endConnector="bus2"/>
          <oms:Connection startElement="WC2" startConnector="bus2"
                         endElement="WC1" endConnector="C3"/>
        </oms:Connections>
      </ssc:Annotation>
    </ssd:Annotations>
  </ssd:System>
</ssd:SystemStructureDescription>
```

CHAPTER

FIFTEEN

SYSTEM IDENTIFICATION

System Identification (OMSysIdent) is part of the OpenModelica tool suite, but not bundled together with the main OpenModelica distribution and thus must be fetched separately from its project site.

OMSysIdent is a module for the parameter estimation for linear and nonlinear parametric dynamic models (wrapped as FMUs) on top of the *OMSimulator* API. It uses the Ceres solver (http://ceres-solver.org/) for the optimization task. The module provides a *Python scripting API* as well as an *C API*.

Note

Notice that this module was previously part of OMSimulator. It has been extracted out of the OMSimulator project and reorganized as a separate project in September 2020. As of 2020-10-07 the project is working on Linux but some more efforts are needed for migrating the Windows build and make the build and usage of the module more convenient.

Version: a65a0ed

15.1 Examples

There are examples in the testsuite which use the scripting API, as well as examples which directly use the C API.

Below is a basic example from the testsuite (*HelloWorld_cs_Fit.py*) which uses the Python scripting API. It determines the parameters for the following "hello world" style Modelica model:

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

The goal is to estimate the value of the coefficent *a* and the initial value *x_start* of the state variable *x*. Instead of real measurements, the script simply uses simulation data generated from the *HelloWorld* examples as measurement data. The array *data_time* contains the time instants at which a sample is taken and the array *data_x* contains the value of *x* that corresponds to the respective time instant.

The estimation parameters are defined by calls to function *simodel.addParameter(..)* in which the name of the parameter and a first guess for the parameter's value is stated.

Listing 15.1: HelloWorld_cs_Fit.py

```
from OMSimulator import OMSimulator
from OMSysIdent import OMSysIdent
import numpy as np

```

(continues on next page)

(continued from previous page)

```
oms = OMSimulator()

oms.setLogFile("HelloWorld_cs_Fit_py.log")
oms.setTempDirectory("./HelloWorld_cs_Fit_py/")
oms.newModel("HelloWorld_cs_Fit")
oms.addSystem("HelloWorld_cs_Fit.root", oms.system_wc)
# oms.setTolerance("HelloWorld_cs_Fit.root", 1e-5)

# add FMU
oms.addSubModel("HelloWorld_cs_Fit.root.HelloWorld", "../resources/HelloWorld.fmu")

# create simodel for model
simodel = OMSysIdent("HelloWorld_cs_Fit")
# simodel.describe()

# Data generated from simulating HelloWorld.mo for 1.0s with Euler and 0.1s step size
kNumSeries = 1
kNumObservations = 11
data_time = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
inputvars = []
measurementvars = ["root.HelloWorld.x"]
data_x = np.array([1, 0.9, 0.8100000000000001, 0.7290000000000001, 0.6561, 0.
↪5904900000000001, 0.5314410000000001, 0.4782969000000001, 0.43046721, 0.387420489,␣
↪0.3486784401])

simodel.initialize(kNumSeries, data_time, inputvars, measurementvars)
# simodel.describe()

simodel.addParameter("root.HelloWorld.x_start", 0.5)
simodel.addParameter("root.HelloWorld.a", -0.5)
simodel.addMeasurement(0, "root.HelloWorld.x", data_x)
# simodel.describe()

simodel.setOptions_max_num_iterations(25)
simodel.solve("BriefReport")

status, state = simodel.getState()
# print('status: {0}; state: {1}').format(OMSysIdent.status_str(status), OMSysIdent.
↪omsi_simodelstate_str(state))

status, startvalue1, estimatedvalue1 = simodel.getParameter("root.HelloWorld.a")
status, startvalue2, estimatedvalue2 = simodel.getParameter("root.HelloWorld.x_start")
# print('HelloWorld.a startvalue1: {0}; estimatedvalue1: {1}'.format(startvalue1,␣
↪estimatedvalue1))
# print('HelloWorld.x_start startvalue2: {0}; estimatedvalue2: {1}'.format(startvalue2,
↪ estimatedvalue2))
is_OK1 = estimatedvalue1 > -1.1 and estimatedvalue1 < -0.9
is_OK2 = estimatedvalue2 > 0.9 and estimatedvalue2 < 1.1
print('HelloWorld.a estimation is OK: {0}'.format(is_OK1))
print('HelloWorld.x_start estimation is OK: {0}'.format(is_OK2))

# del simodel
oms.terminate("HelloWorld_cs_Fit")
oms.delete("HelloWorld_cs_Fit")
```

Running the script generates the following console output:

```
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter␣
↪iter_time  total_time
 0  4.069192e-01    0.00e+00    2.20e+00   0.00e+00   0.00e+00  1.00e+04        0   ␣
↪7.91e-03    7.93e-03
 1  4.463938e-02    3.62e-01    4.35e-01   9.43e-01   8.91e-01  1.92e+04        1   ␣
↪7.36e-03    1.53e-02
 2  7.231043e-04    4.39e-02    5.16e-02   3.52e-01   9.85e-01  5.75e+04        1   ␣
↪7.26e-03    2.26e-02
 3  1.046555e-07    7.23e-04    4.74e-04   4.40e-02   1.00e+00  1.73e+05        1   ␣
↪7.31e-03    3.00e-02
 4  2.192358e-15    1.05e-07    5.77e-08   6.05e-04   1.00e+00  5.18e+05        1   ␣
↪7.15e-03    3.71e-02
 5  7.377320e-26    2.19e-15    2.05e-13   9.59e-08   1.00e+00  1.55e+06        1   ␣
↪7.42e-03    4.46e-02
Ceres Solver Report: Iterations: 6, Initial cost: 4.069192e-01, Final cost: 7.377320e-
↪26, Termination: CONVERGENCE

=====================================
Total duration for parameter estimation: 44msec.
Result of parameter estimation (check 'Termination' status above whether solver␣
↪converged):

HelloWorld_cs_Fit.root.HelloWorld.a(start=-0.5, *estimate*=-1)
HelloWorld_cs_Fit.root.HelloWorld.x_start(start=0.5, *estimate*=1)

=====================================
HelloWorld.a estimation is OK: True
HelloWorld.x_start estimation is OK: True
info:    Logging information has been saved to "HelloWorld_cs_Fit_py.log"
```

15.2 Python and C API

15.2.1 addInput

Add input values for external model inputs.

If there are several measurement series, all series need to be conducted with the same external inputs!

Python

Args:

var
(str) Name of variable..

values
(np.array) Array of input values for respective time instants in *simodel.initialize()*.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.addInput(var, values)
```

C

```
oms_status_enu_t omsi_addInput(void* simodel, const char* var, const double* values,␣
↪size_t nValues);
```

15.2.2 addMeasurement

Add measurement values for a fitting variable.

Python

Args:

iSeries
(int) Index of measurement series.

var
(str) Name of variable..

values
(np.array) Array of measured values for respective time instants in *simodel.initialize()*.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.addMeasurement(iSeries, var, values)
```

C

```
oms_status_enu_t omsi_addMeasurement(void* simodel, size_t iSeries, const char* var,␣
↪const double* values, size_t nValues);
```

15.2.3 addParameter

Add parameter that should be estimated.

PYTHON

Args:

var
(str) Name of parameter.

startvalue
(float) Start value of parameter.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.addParameter(var, startvalue)
```

C

```
oms_status_enu_t omsi_addParameter(void* simodel, size_t iSeries, const char* var,␣
↪const double* values, size_t nValues);
```

15.2.4 describe

Print summary of SysIdent model.

PYTHON

```
status = simodel.describe()
```

C

```
oms_status_enu_t omsi_describe(void* simodel);
```

15.2.5 freeSysIdentModel

Unloads a model.

PYTHON

Not available in Python. Related external C function called by class destructor.

C

```
void omsi_freeSysIdentModel(void* simodel);
```

15.2.6 getParameter

Get parameter that should be estimated.

PYTHON

Args:

var
(str) Name of parameter.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

startvalue
(float) Start value of parameter.

estimatedvalue
(float) Estimated value of parameter.

```
status, startvalue, estimatedvalue = simodel.getParameter(var)
```

C

```
oms_status_enu_t omsi_getParameter(void* simodel, const char* var, double* startvalue,
↪ double* estimatedvalue);
```

15.2.7 getState

Get state of SysIdent model object.

PYTHON

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

state
(int) State of SysIdent model (*omsi_simodelstate_t*).

```
status, state = simodel.getState()
```

C

```
oms_status_enu_t omsi_getState(void* simodel, omsi_simodelstate_t* state);
```

15.2.8 initialize

This function initializes a given composite model. After this call, the model is in simulation mode.

PYTHON

Args:

nSeries
(int) Number of measurement series.

time
(numpy.array) Array of measurement/input time instants.

inputvars
(list of str) List of names of input variables (empty list if none).

measurementvars
(list of str) List of names of observed measurement variables.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.initalize(nSeries, time, inputvars, measurementvars)
```

C

```
oms_status_enu_t omsi_initialize(void* simodel, size_t nSeries, const double* time,␣
↪size_t nTime, char const* const* inputvars, size_t nInputvars, char const* const*␣
↪measurementvars, size_t nMeasurementvars);
```

15.2.9 newSysIdentModel

Creates an empty model for parameter estimation.

PYTHON

The corresponding Python function is the class constructor.

Args:

ident
(str) Name of the model instance.

Returns:

simodel
SysIdent model instance.

```
simodel = OMSysIdent(ident)
```

C

```
void* omsi_newSysIdentModel(const char* ident);
```

15.2.10 oms_status_str

Mapping of enum C-API status code (oms_status_enu_t) to string.

The C enum is reproduced below for convenience.

```
typedef enum {
  oms_status_ok,
  oms_status_warning,
  oms_status_discard,
  oms_status_error,
  oms_status_fatal,
  oms_status_pending
} oms_status_enu_t;
```

PYTHON

Args:

status
(int) The C-API status code.

Returns:

status_str
(str) String representation of status code.

The range of values of `status` corresponds to the C enum (by implicit conversion). This is a static Python method (`@staticmethod`).

```
status_str = oms_status_str(status)
```

C

Not available.

15.2.11 omsi_simodelstate_str

Mapping of enum C-API state code (omsi_simodelstate_t) to string.

The C enum is reproduced below for convenience.

```
typedef enum {
  omsi_simodelstate_constructed,    //!< After omsi_newSysIdentModel
  omsi_simodelstate_initialized,    //!< After omsi_initialize
  omsi_simodelstate_convergence,    //!< After omsi_solve if Ceres minimizer returned␣
↪with ceres::TerminationType::CONVERGENCE
  omsi_simodelstate_no_convergence, //!< After omsi_solve if Ceres minimizer returned␣
↪with ceres::TerminationType::NO_CONVERGENCE
  omsi_simodelstate_failure         //!< After omsi_solve if Ceres minimizer returned␣
↪with ceres::TerminationType::FAILURE
} omsi_simodelstate_t;
```

PYTHON

Args:

state
(int) State of SysIdent model.

Returns:

simodelstate_str
(str) String representation of state code.

The range of values of `state` corresponds to the C enum (by implicit conversion). This is a static Python method (`@staticmethod`).

```
simodelstate_str = omsi_simodelstate_str(state)
```

C

Not available.

15.2.12 setOptions_max_num_iterations

Set Ceres solver option *Solver::Options::max_num_iterations*.

PYTHON

Args:

max_num_iterations
(int) Maximum number of iterations for which the solver should run (default: 25).

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.setOptions_max_num_iterations(max_num_iterations)
```

C

```
oms_status_enu_t omsi_setOptions_max_num_iterations(void* simodel, size_t max_num_
↪iterations);
```

15.2.13 solve

Solve parameter estimation problem.

PYTHON

Args:

reporttype
(str) Print report and progress information after call to Ceres solver. Supported report types: *""*, *"BriefReport"*, *"FullReport"*, where *""* denotes no output.

Returns:

status
(int) The C-API status code (*oms_status_enu_t*).

```
status = simodel.solve(reporttype)
```

C

```
oms_status_enu_t omsi_solve(void* simodel, const char* reporttype);
```

CHAPTER

SIXTEEN

OPENMODELICA ENCRYPTION

The encryption module allows the library developers to encrypt their libraries for different platforms. Note that you need a special version of OpenModelica with encryption support to do that, which is only released in binary form. This version contains an OpenModelica-specific private key that is used internally to decrypt the encrypted libraries for code generation only, not for display purposes.

If you are a library developer and are interested in distributing your library in encrypted form for use with OpenModelica, please contact us for further information. Please note that distributing the special version of OpenModelica with encryption support to the users of your library requires you to be a Level 2 member of the Open Source Modelica Consortium.

If you are a user of an encrypted library that is supported by OpenModelica, please contact your library supplier for information on how to get the special version of OpenModelica that runs it.

16.1 Encrypting the Library

In order to encrypt the Modelica package call *buildEncryptedPackage(TopLevelPackageName)* from mos script or from **OMEdit** right click the package in Libraries Browser and select *Export Encrypted Package* or select *Export > Export Encrypted Package* from the menu.

All the Modelica files are encrypted and the whole library is zipped into a single file i.e., *PackageName.mol*. Note that you can only encrypt Modelica packages saved in a folder structure. The complete folder structure remains as it is. No encryption is done on the resource files.

16.2 Loading an Encrypted Library

To load the encrypted package call *loadEncryptedPackage(EncryptedPackage.mol)* from the mos script or from **OMEdit** *File > Load Encrypted Package*.

16.3 Notes

- Encryption support in OpenModelica does not include any license management, i.e., restricting the usage of a certain libraries based on some conditions, e.g., having paid a fee. It is only meant to prevent end users from seeing the Modelica source code of the encrypted parts of the libraries, for reasons of confidentiality or IP protection.
- The parts of the library that are protected by encryption are specified by the access control annotations defined by the Modelica Language Specification, Section 18.9.
- The generated C code corresponding to the encrypted parts of the library is obfuscated: all comments are removed, and all component names are replaced by generic names such as n1, n2, n3, etc. This prevents easy reverse-engineering of the encrypted library starting from generated simulation code.
- Encryption in OpenModelica is based on the SEMLA (Standardized Encryption of Modelica Libraries and Artifacts) module from Modelon AB, which provides a tool-independent framework for Modelica library encryption.

CHAPTER

SEVENTEEN

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.

17.1 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.

17.1.1 Mathematica Notebooks

Literate Programming [Knu84] is a form of programming where programs are integrated with documentation in the same document. Mathematica notebooks [Wol96] is one of the first WYSIWYG (What-You-See-Is-What-You-Get) systems that support Literate Programming. Such notebooks are used, e.g., in the MathModelica modeling and simulation environment, see e.g. Figure 17.1 below and Chapter 19 in [Fri04].

17.1.2 OMNotebook

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

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

A more advanced electronic notebook tool, also supporting mathematical typesetting and many other facilities, is provided by Mathematica notebooks in the MathModelica environment, see Figure 17.1.

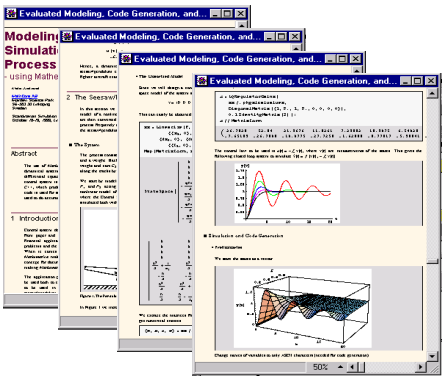


Figure 17.1: Examples of Mathematica notebooks in the MathModelica modeling and simulation environment.

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.

17.2 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 17.2).

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

Now, let us consider that the link "*HelloWorld*" in DrModelica Section is clicked by the user. The new HelloWorld notebook (see Figure 17.3), to which the user is being linked, is not only a textual description but also contains one or more examples explaining the specific keyword. In this class, HelloWorld, a differential equation is specified.

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

When a class has been successfully evaluated the user can simulate and plot the result, as previously depicted in Figure 17.3 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 17.4 shows part of Chapter 9 of the DrModelica teaching material. Here the user can read about language constructs, like algorithm sections, when-statements, and reinit equations, and then practice these constructs by solving the exercises corresponding to the recently studied section.

Exercise 1 from Chapter 9 is shown in Figure 17.5. 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 17.6.

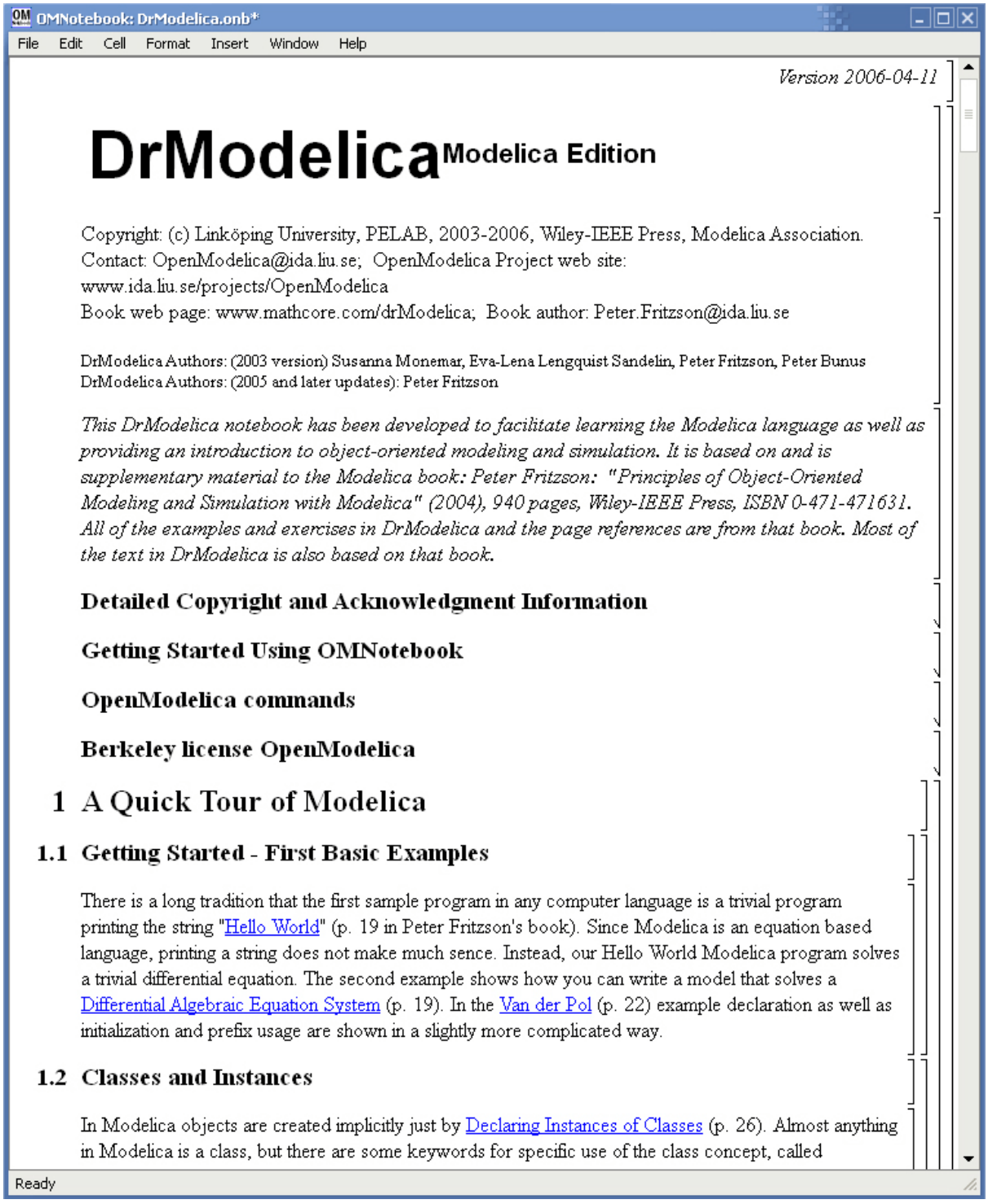


Figure 17.2: The front-page notebook of the OMNotebook version of the DrModelica tutoring system.

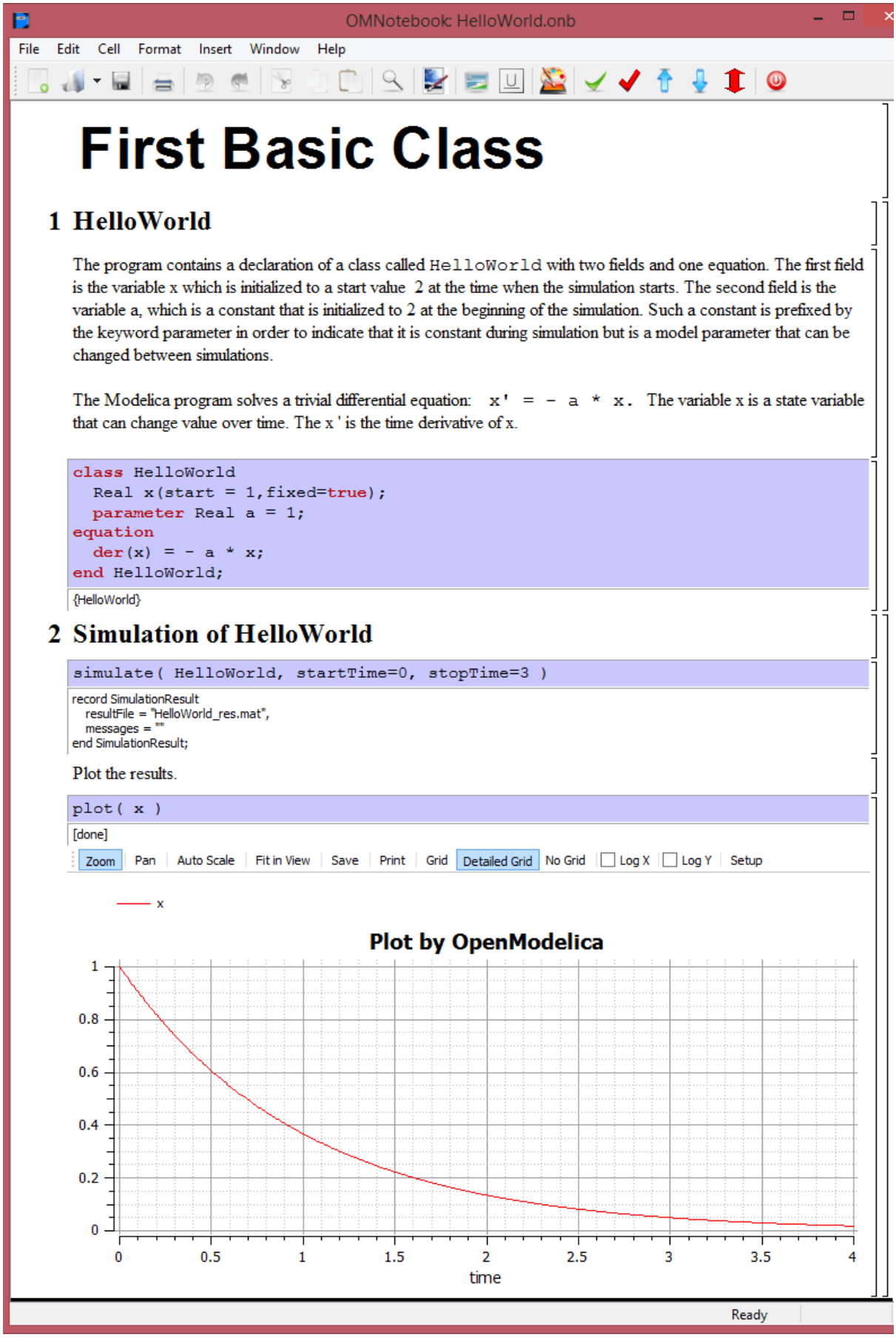


Figure 17.3: The HelloWorld class simulated and plotted using the OMNotebook version of DrModelica.

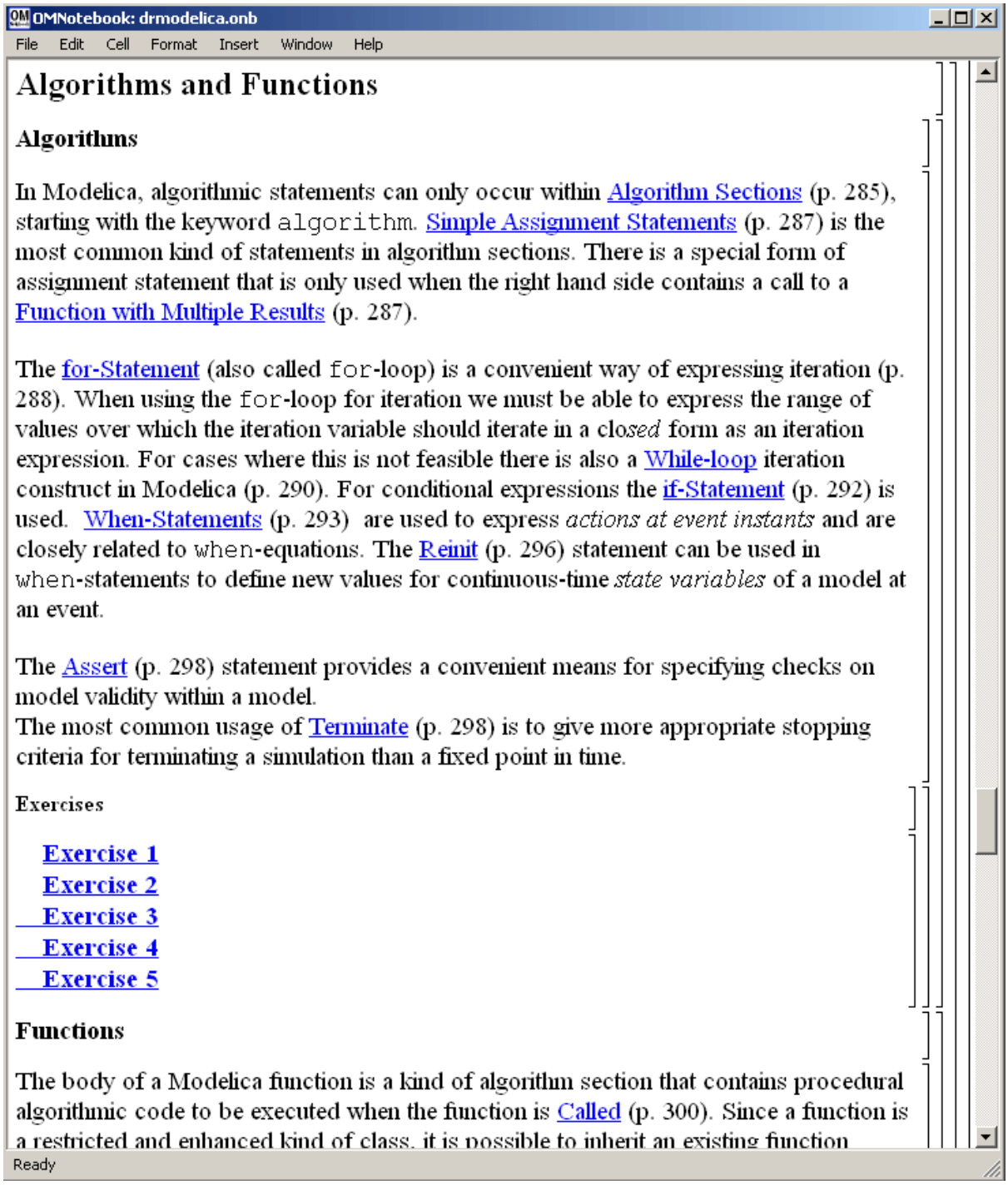


Figure 17.4: DrModelica Chapter on Algorithms and Functions in the main page of the OMNotebook version of DrModelica.

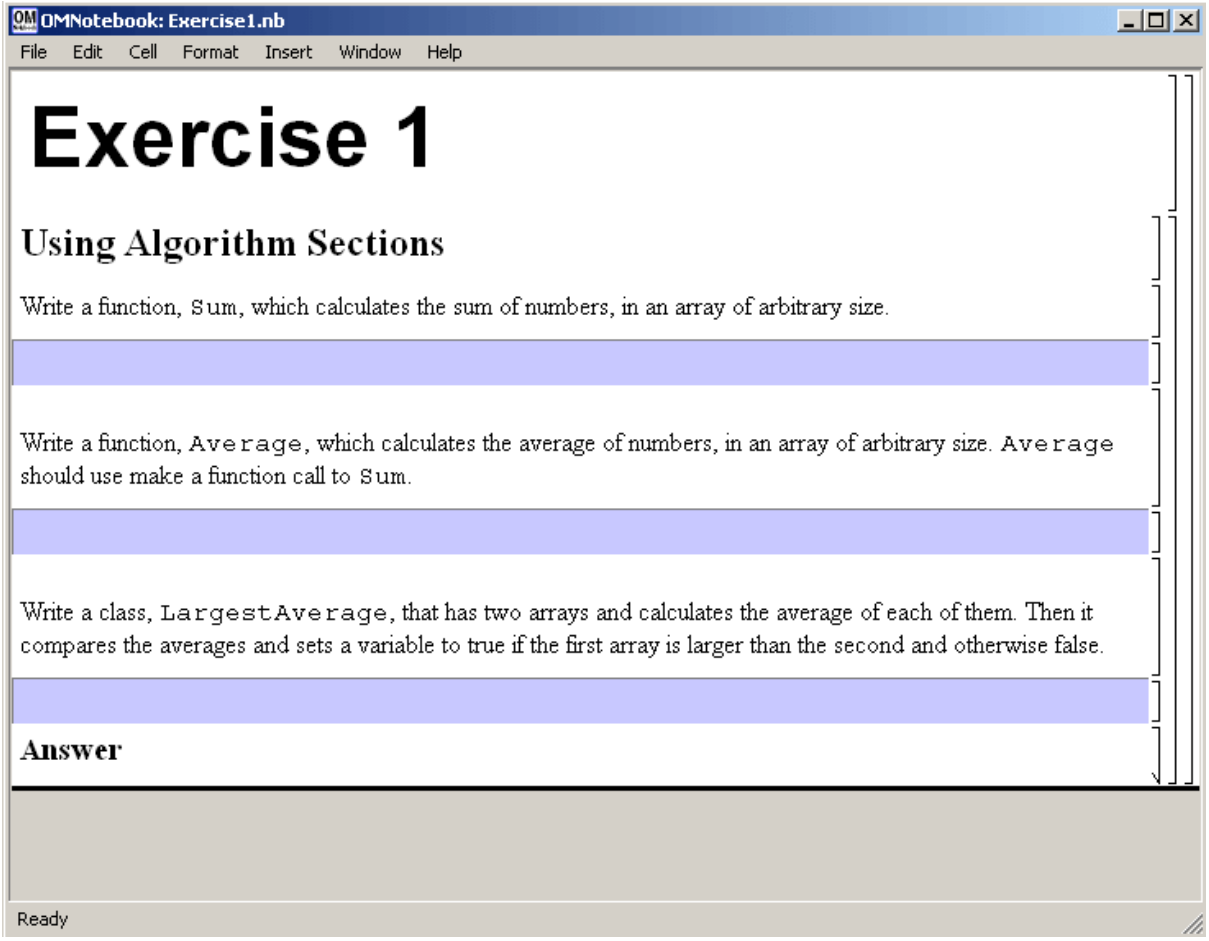


Figure 17.5: Exercise 1 in Chapter 9 of DrModelica.

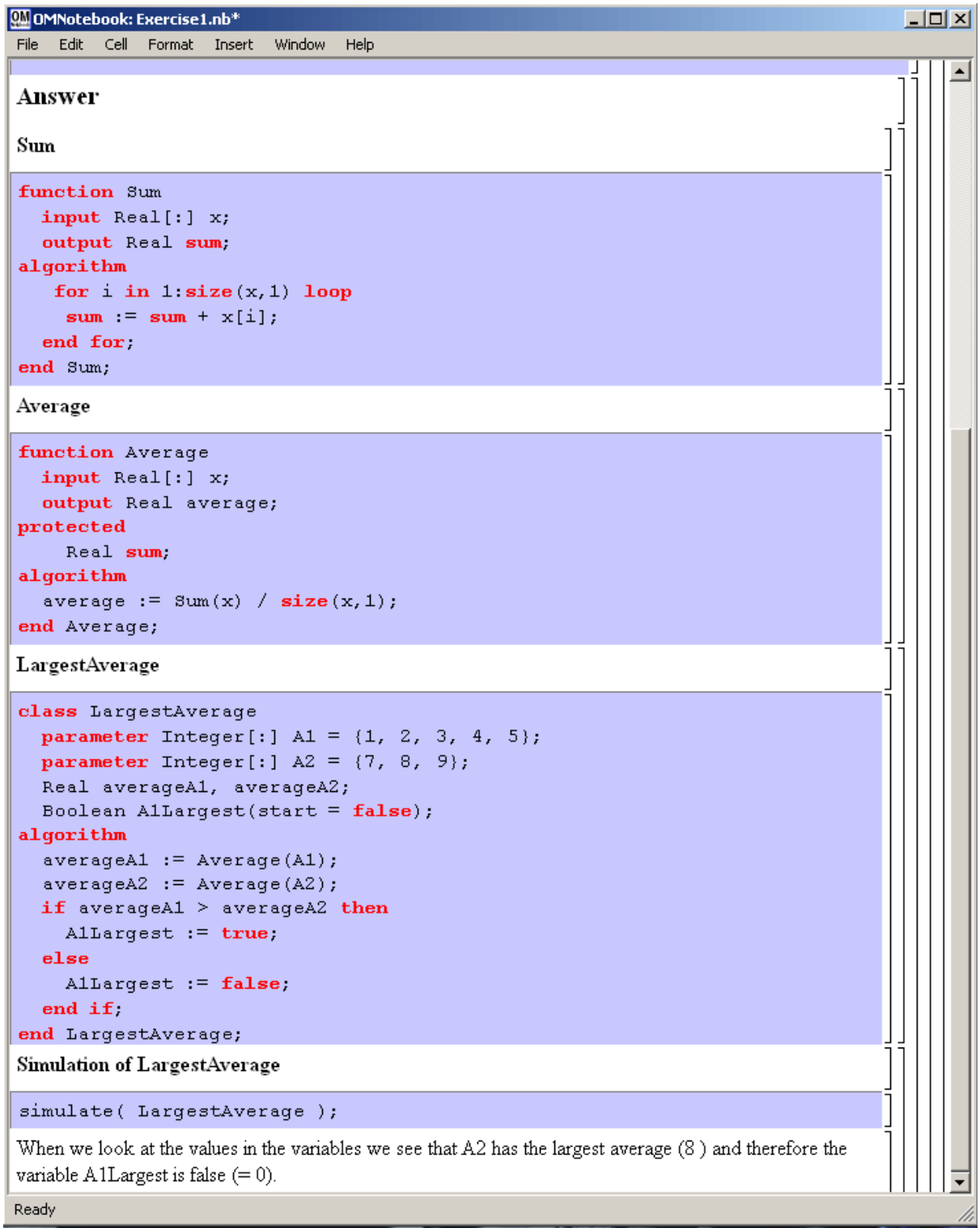


Figure 17.6: The answer section to Exercise 1 in Chapter 9 of DrModelica.

17.3 DrControl Tutorial for Teaching Control Theory

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

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

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

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

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

17.3.1 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 17.7, control of a simple car model is illustrated where the car velocity on a road is controlled, first with an open loop control, and then compared to a closed loop system with a feedback loop. The car has a mass m, velocity y, and aerodynamic coefficient α. The ϑ is the road slope, which in this case can be regarded as noise.

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

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

```
model noFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y;                              // output signal without noise, theta =␣
↪0 -> v(t) = 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;
```

(continues on next page)

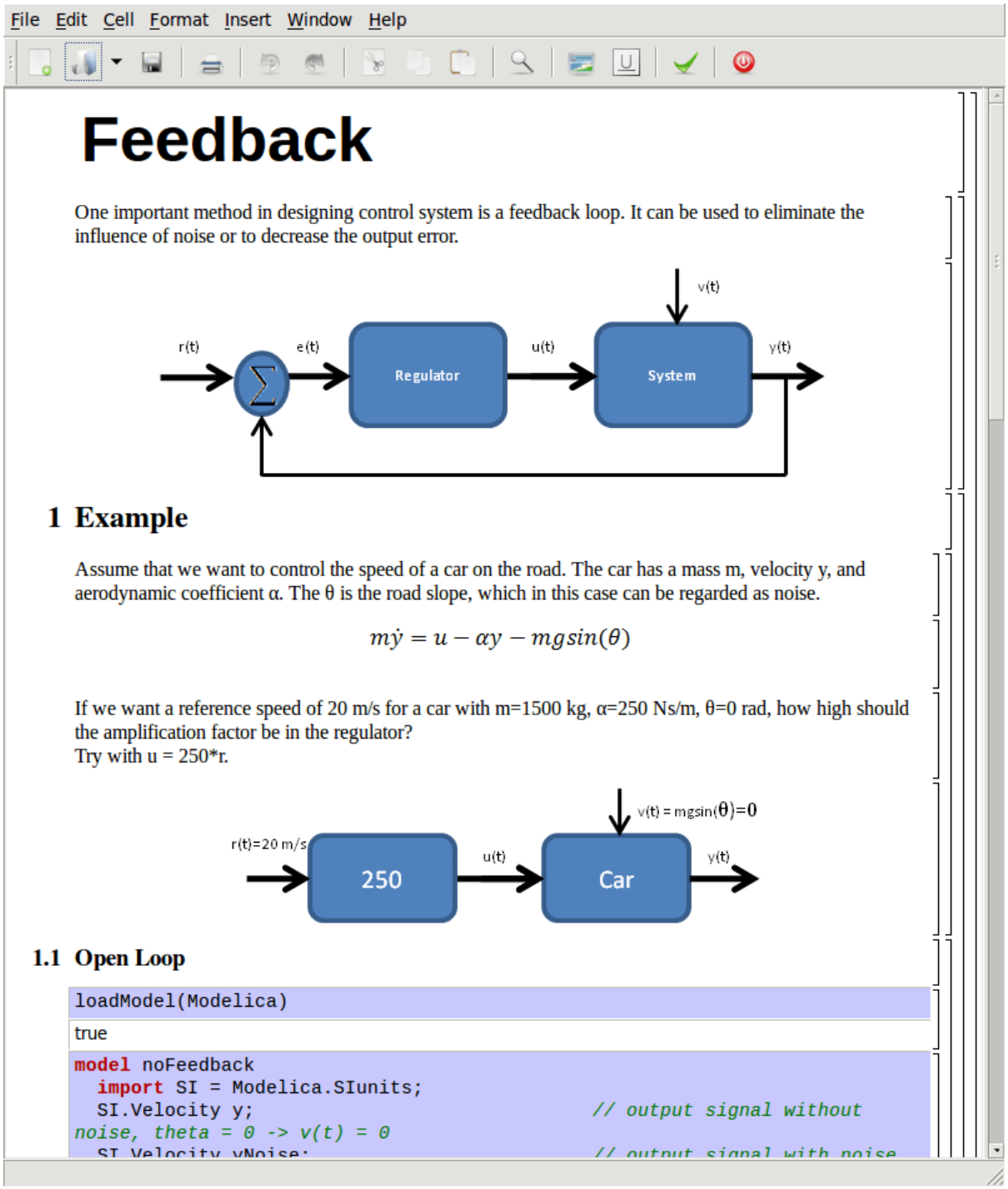


Figure 17.7: Feedback loop.

(continued from previous page)

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

By applying a road slope angle different from zero the car velocity is influenced which can be regarded as noise in this model. The output signal in Figure 17.8 is stable but an overshoot can be observed compared to the reference signal. Naturally the overshoot is not desired and the student will in the next exercise learn how to get rid of this undesired behavior of the system.

```
>>> loadModel(Modelica, {"3.2.3"})
true
>>> simulate(noFeedback, stopTime=100)
record SimulationResult
    resultFile = "«DOCHOME»/noFeedback_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 100.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'noFeedback', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.002998286,
    timeBackend = 0.006447299,
    timeSimCode = 0.0024222130000000003,
    timeTemplates = 0.0020020330000000003,
    timeCompile = 0.168138182,
    timeSimulation = 0.005669668,
    timeTotal = 0.18779235700000002
end SimulationResult;
```

Warning

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->Show additional information from the initialization process, in OMNotebook call setCommandLineOptions("-d=initialization").

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

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

```
model withFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y;                               // output signal with feedback link␣
↪and without noise, theta = 0 -> v(t) = 0
  SI.Velocity yNoise;                          // output signal with feedback link␣
↪and noise,    theta <> 0 -> v(t) <> 0
  parameter SI.Mass m = 1500;
  parameter Real alpha = 250;
```

(continues on next page)

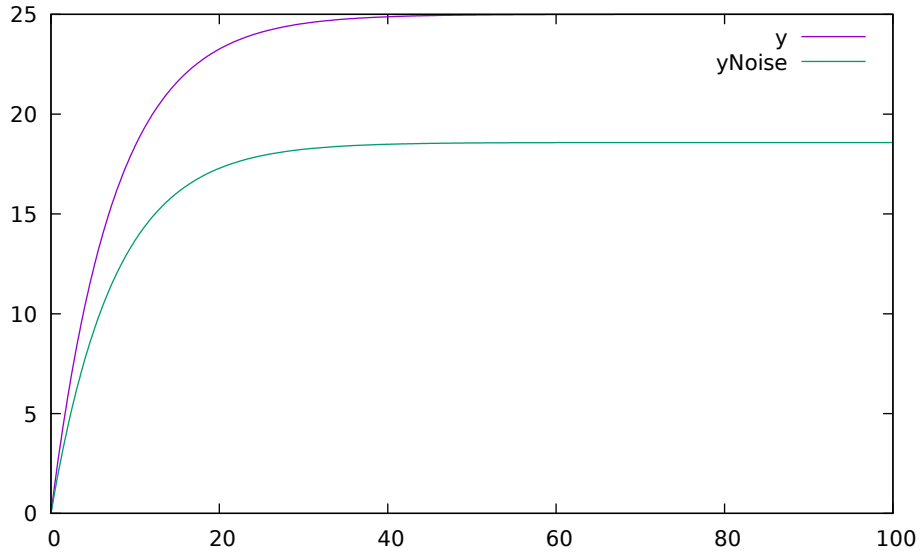


Figure 17.8: Open loop control example.

(continued from previous page)

```
  parameter SI.Angle theta = 5*3.141592/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Force uNoise;
  SI.Velocity r=20;
equation
  m*der(y)=u-alpha*y;
  m*der(yNoise)=uNoise-alpha*yNoise-m*g*sin(theta);
  u = 5000*(r-y);
  uNoise = 5000*(r-yNoise);
end withFeedback;
```

By using the information about the current level of the output signal and re-tune the regulator the output quantity can be controlled towards the reference signal smoothly and without an overshoot, as shown in Figure 17.9.

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

```
>>> loadModel(Modelica, {"3.2.3"})
true
>>> simulate(withFeedback, stopTime=10)
record SimulationResult
    resultFile = "«DOCHOME»/withFeedback_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'withFeedback', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.002432062,
    timeBackend = 0.0059932860000000004,
    timeSimCode = 0.0018439760000000002,
    timeTemplates = 0.001951538,
```

(continues on next page)

(continued from previous page)

```
    timeCompile = 0.166345252,
    timeSimulation = 0.005956136000000001,
    timeTotal = 0.184582312
end SimulationResult;
```

Warning

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->Show additional information from the initialization process, in OMNotebook call setCommandLineOptions("-d=initialization").

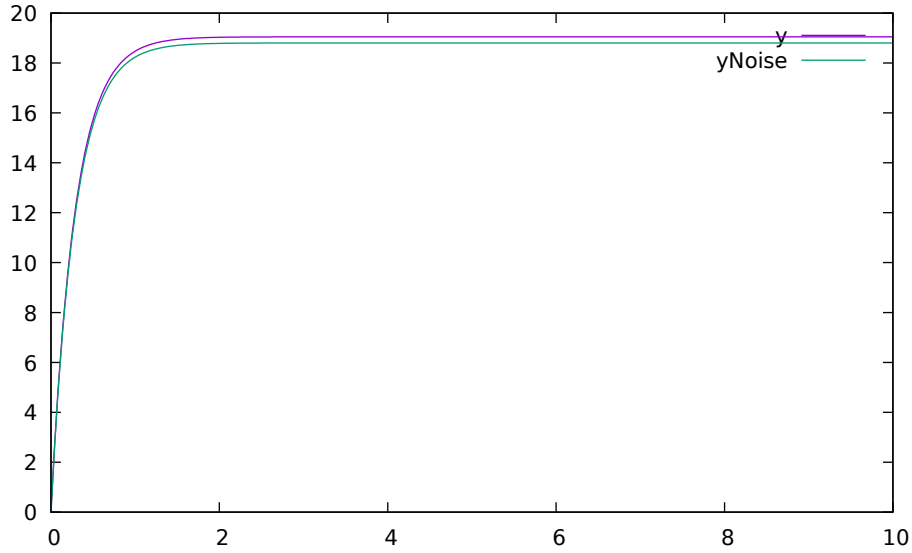


Figure 17.9: Closed loop control example.

17.3.2 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 17.10.

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

Now let us examine a second order system:

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

```
model NegRoots
  Real y;
  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 17.11 and Figure 17.12.

In the first example the values of a_1 and a_2 are chosen in such way that the characteristic equation has negative real roots and thereby a stable output response, see Figure 17.11.

```
>>> simulate(NegRoots, stopTime=10)
record SimulationResult
    resultFile = "«DOCHOME»/NegRoots_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'NegRoots', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0010930450000000002,
    timeBackend = 0.0038628910000000002,
    timeSimCode = 8.9415e-4,
    timeTemplates = 0.0016541890000000001,
    timeCompile = 0.161463365,
    timeSimulation = 0.005363963,
    timeTotal = 0.174389433
end SimulationResult;
```

Warning

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->Show additional information from the initialization process, in OMNotebook call setCommandLineOptions("-d=initialization").

The importance of the sign of the roots in the characteristic equation is illustrated in Figure 17.11 and Figure 17.12, e.g., a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

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

```
>>> simulate(ImgPosRoots, stopTime=10)
record SimulationResult
    resultFile = "«DOCHOME»/ImgPosRoots_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'ImgPosRoots', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.001084148,
```

(continues on next page)

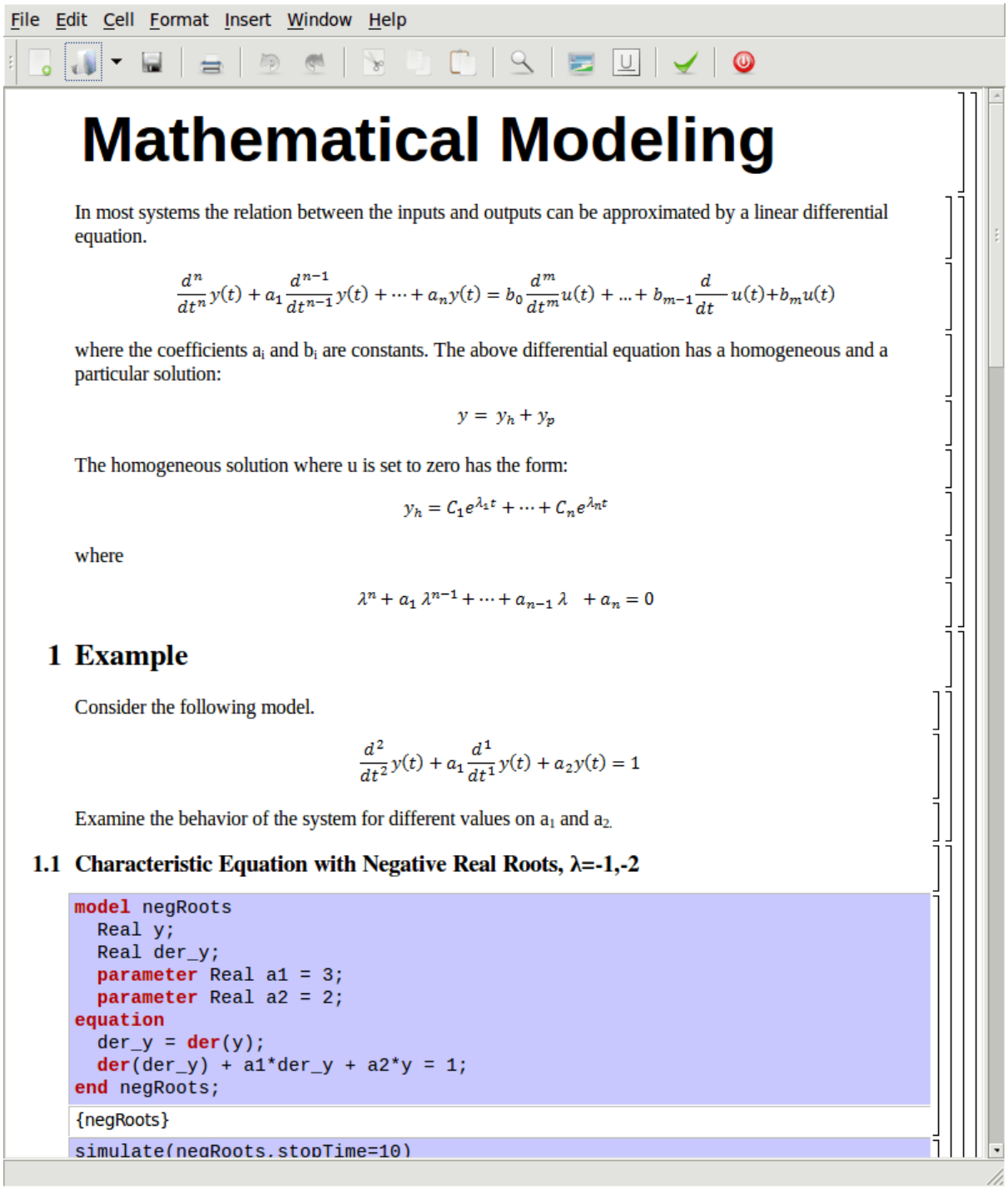


Figure 17.10: Mathematical modeling with characteristic equation.

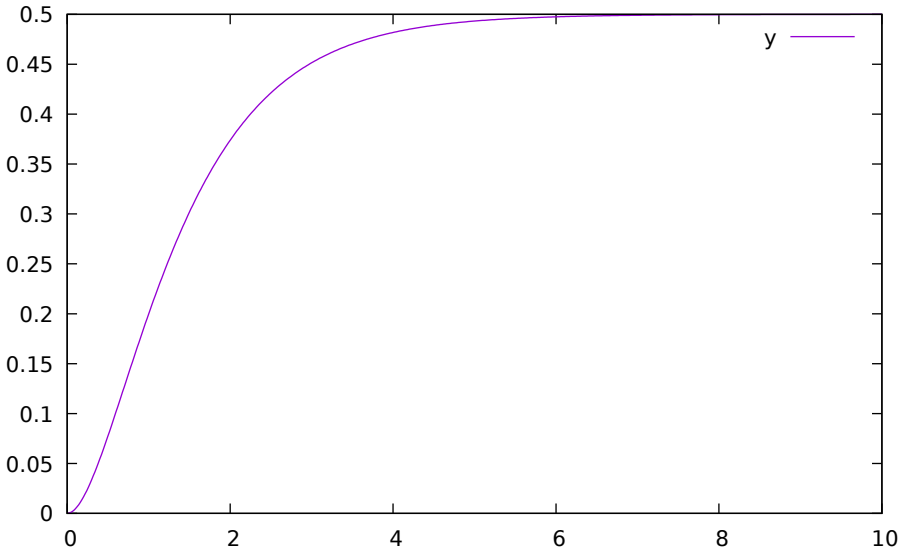


Figure 17.11: Characteristic equation with real negative roots.

(continued from previous page)

```
    timeBackend = 0.003825661,
    timeSimCode = 8.896520000000001e-4,
    timeTemplates = 0.001821965,
    timeCompile = 0.16053614300000002,
    timeSimulation = 0.005673114,
    timeTotal = 0.17390012300000002
end SimulationResult;
```

Warning

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->Show additional information from the initialization process, in OMNotebook call setCommandLineOptions("-d=initialization").

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

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

17.4 OpenModelica Notebook Commands

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

17.4.1 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

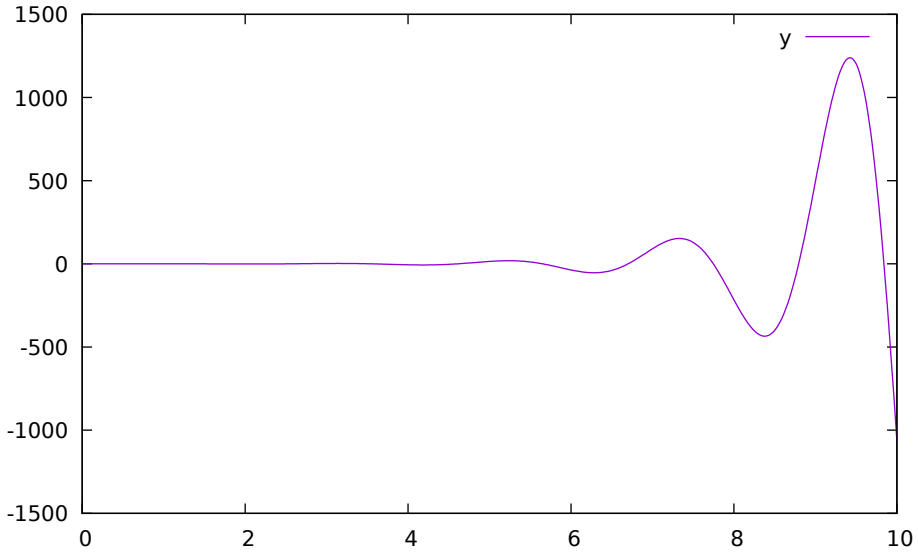


Figure 17.12: Characteristic equation with imaginary roots with positive real part.

the menu Format->Styles, example of different styles are: Text, Title, and Subtitle. The Textcell type also has support for following links to other notebook documents.

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

17.4.2 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:

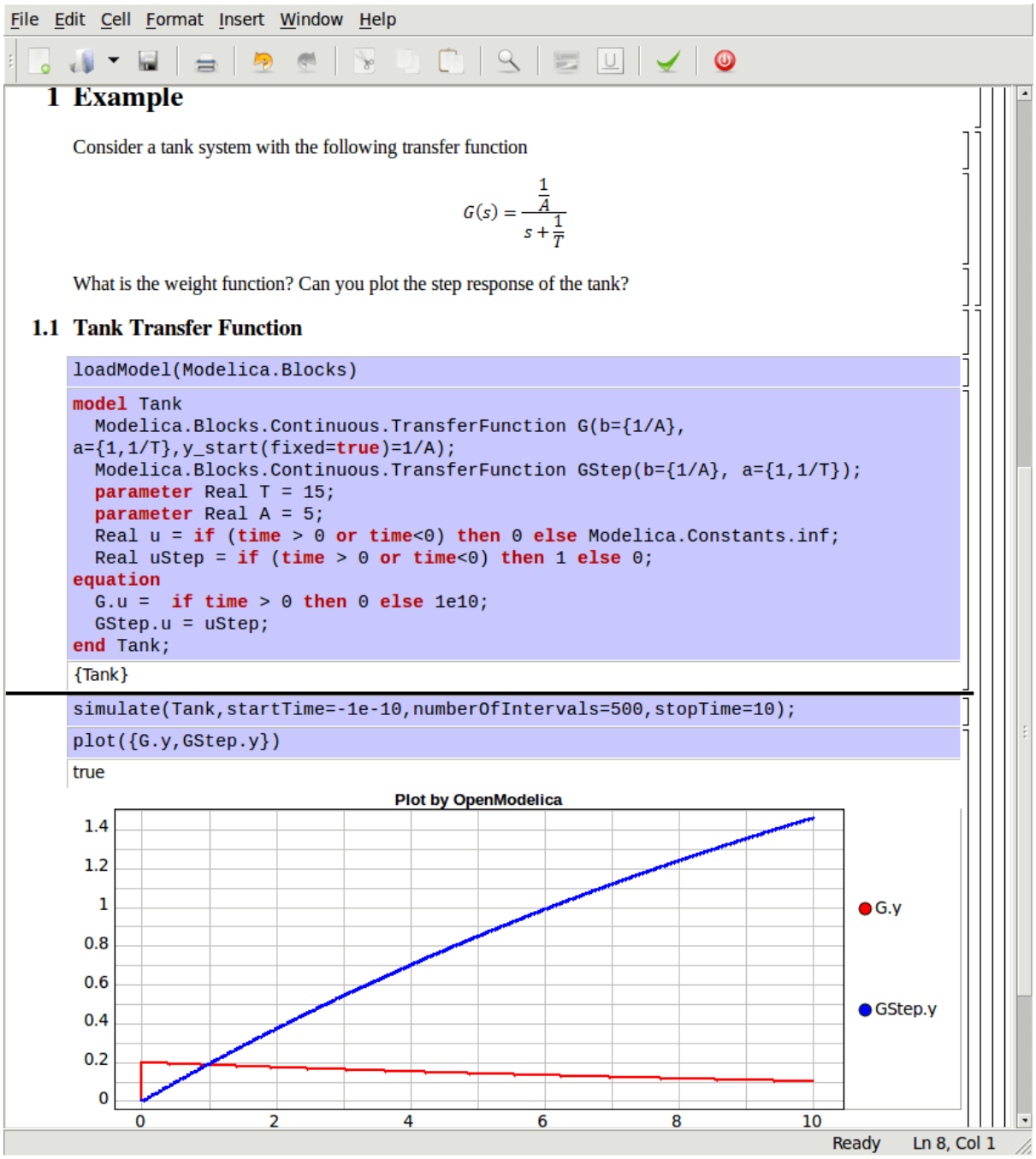


Figure 17.13: Step and pulse (weight function) response.

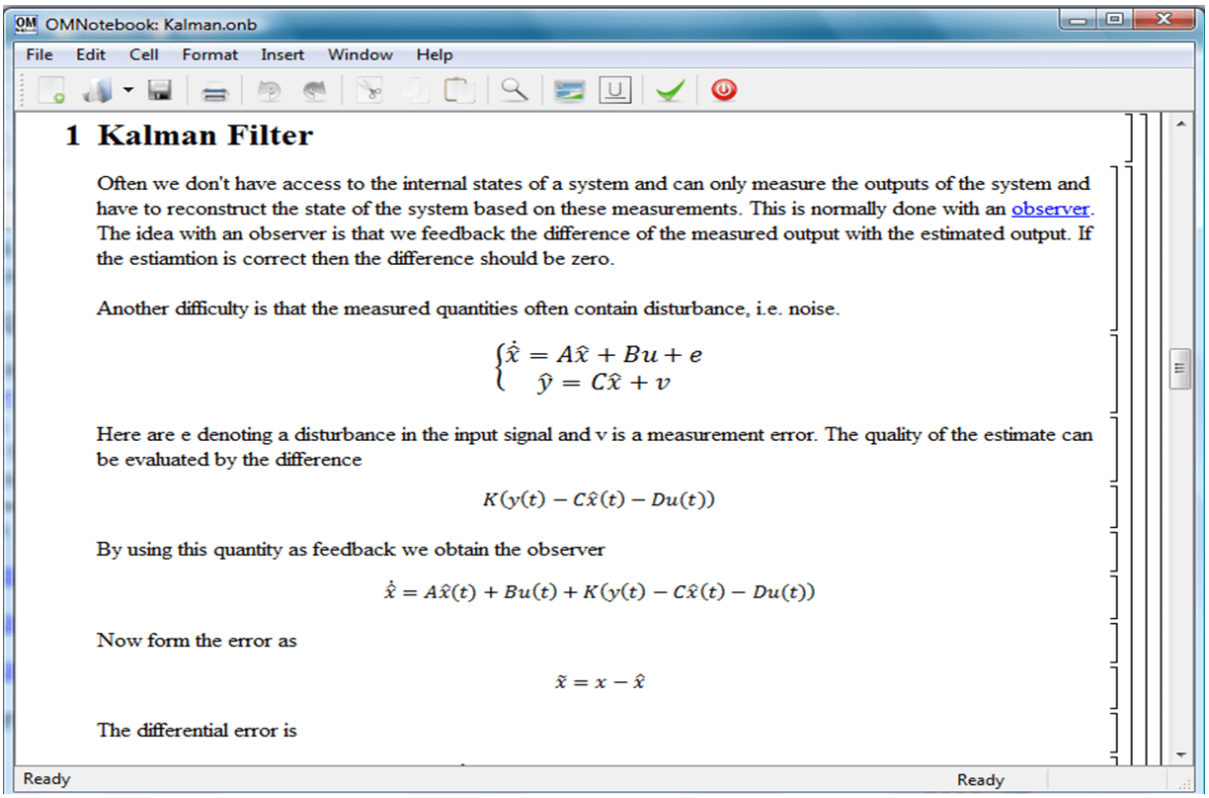


Figure 17.14: Theory background about Kalman filter.

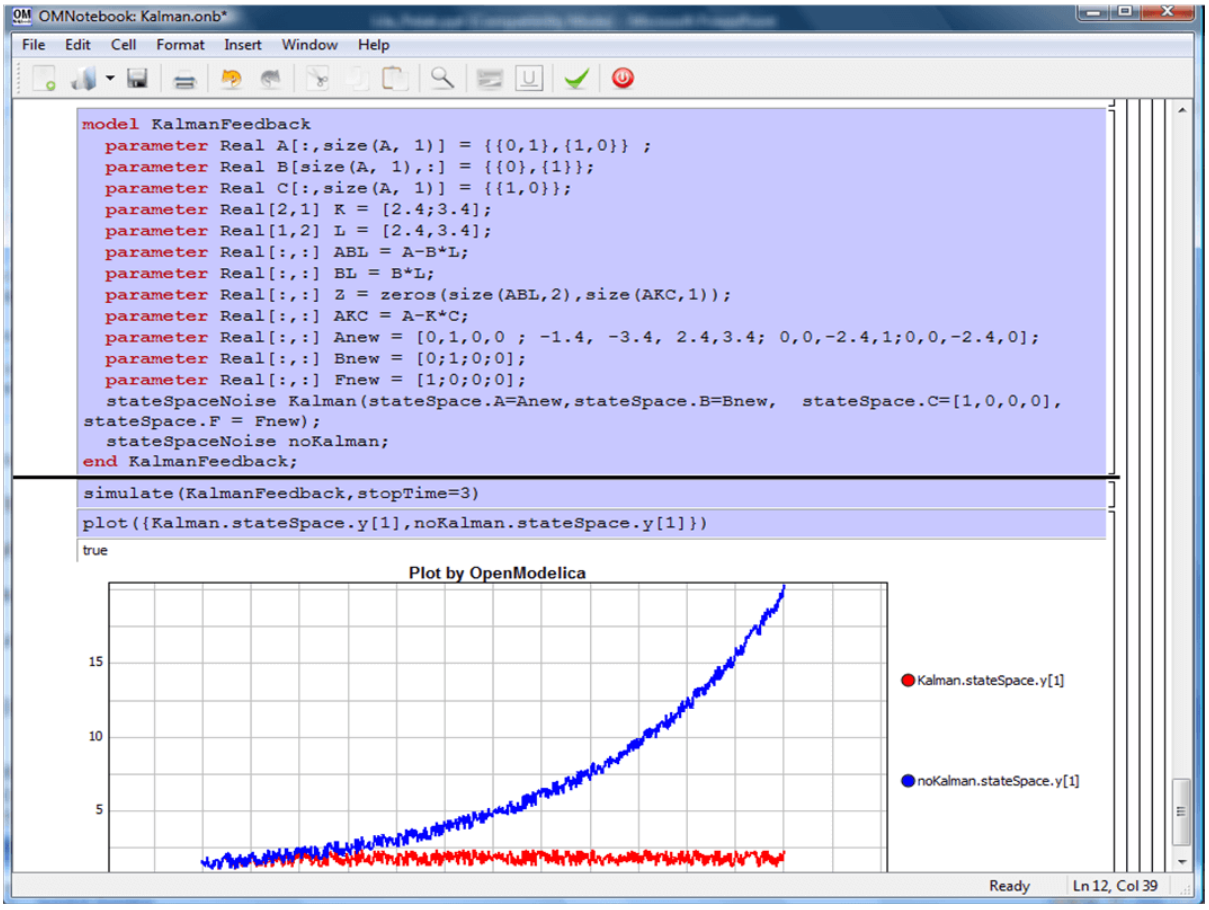


Figure 17.15: Comparison of a noisy system with feedback link in DrControl.

In order to paste cells at the cellcursor, the *dynamic cellcursor must be made active* by clicking on the main cellcursor (the horizontal line).

17.4.3 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.

17.4.4 File Menu

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

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

17.4.5 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.

17.4.6 Cell Menu

- ***Add textcell* - A new textcell is added with the menu item Cell->Add**
  Cell (previous cell style) or the key combination Alt+Enter. The new textcell gets the same style as the previous selected cell had.
- ***Add inputcell* - A new inputcell is added with the menu item**
  Cell->Add Inputcell or the key combination Ctrl+Shift+I.
- ***Add latexcell* - A new latexcell is added with the menu item**
  Cell->Add Latexcell or the key combination Ctrl+Shift+E.
- ***Add groupcell* - A new groupcell is inserted with the menu item**
  Cell->Groupcell or the key combination Ctrl+Shift+G. The selected cell will then become the first cell inside the groupcell.
- ***Ungroup groupcell* - A groupcell can be ungrouped by selecting it in**
  the tree view and using the menu item Cell->Ungroup Groupcell or by using the key combination Ctrl+Shift+U. Only one groupcell at a time can be ungrouped.
- ***Split cell* - 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.

17.4.7 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)

17.4.8 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.

17.4.9 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.

17.4.10 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.

17.4.11 Additional Features

- *Links* - **By clicking on a link, OMNotebook will open the document**
  that is referred to in the link.
- *Update link* - **All links are stored with relative file path.**
  Therefore OMNotebook has functions that automatically updating links if a document is resaved in another folder. Every time a document is saved, OMNotebook checks if the document is saved in the same folder as last time. If the folder has changed, the links are updated.
- *Evaluate whole Notebook* - **All the cells present in the Notebook can**
  be evaluated in one step by pressing the red color evalall button in the toolbar. The cells are evaluated in the same order as they are in the Notebook.However the latexcells cannot be evaluated by this feature.
- *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.
- *Moving and Reordering cells in a Notebook* - **It is possible to shift cells**
  to a new position and change the hierarchical order of the document.This can be done by clicking the cell and press the Up and Down arrow button in the tool bar to move either Up or Down. The cells are moved one cell above or below.It is also possible to move a cell directly to a new position with one single click by pressing the red color bidirectional UpDown arrow button in the toolbar. To do this the user has to place the cell cursor to a position where the selected cells must be moved. After selecting the cell cursor position, select the cells you want to shift and press the bidirectional UpDown arrow button. The cells are shifted in the same order as they are selected.This is especially very useful when shifting a group cell.
- *Command completion* - **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.

17.5 References

Todo

Add these into extrarefs.bib and cite them somewhere

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

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

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

CHAPTER

EIGHTEEN

OPTIMIZATION WITH OPENMODELICA

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

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

18.1 Built-in Dynamic Optimization using Annotations

This part of the OM manual is a contribution by Massimo Ceraolo and Vitalij Ruge.

18.1.1 Foreword

Dynamic optimization using Annotations is the most user-friendly way to perform Dynamic Optimization(DO) in OpenModelica, since it allows the OMEdit graphical user interface to be used.

It is also more powerful that the Built-in Dynamic Optimization using Optimica language extensions, since it allows final constraints.

18.1.2 Formulation limitations and algorithm

We can formulate the optimization problem as follows:

$$\min_{u(t)} \left[M\left(\mathbf{x}\left(t_f\right), \mathbf{u}\left(t_f\right), t_f\right) + \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), t\right) \mathrm{dt} \right] \quad \begin{array}{l}\text{optimization purpose} \\ (M \textit{ is the Mayer Term} \\ L \textit{ is the Lagrange Term})\end{array} \tag{18.1}$$

$$\mathbf{0} = \mathbf{f}\left(\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{u}(t), t\right) \quad \begin{array}{l}\text{dynamical description} \\ \text{of the system}\end{array} \tag{18.2}$$

$$\mathbf{0} \leq \mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), t\right) \quad \begin{array}{l}\text{path constraints} \\ \text{e.g. physical limits of components}\end{array} \tag{18.3}$$

$$\mathbf{0} = \mathbf{r}_0\left(\mathbf{x}\left(t_0\right), t_0\right) \quad \text{Initial constraint} \tag{18.4}$$

$$\mathbf{0} = \mathbf{r}_f(\mathbf{x}\left(t_f\right) \mathbf{u}\left(t_f\right), t_f) \quad \text{Final constraint} \tag{18.5}$$

Where $\mathbf{u}(t)$ is the vector of input variables, on which DO works to try to get the desired minimum, and $\mathbf{x}(t)$ is the vector of state variables.

The equations above can be implemented in OpenModelica using the full power of Modelica language, and therefore there is a good freedom and flexibility, under the obvious constraint that the system must be regular enough for the convergence to take place.

However, there are limitations in the possibility operational limits can be set.

The formulation of operational limits in (3) is general, since it allows to use non-boxed constrains. Consider for instance a battery. The energy the battery can deliver is a function of the power we use to charge or discharge it.

Therefore, the actual limit should be described as:

$$E_{\min}(P) \leq E_{\text{bat}} \leq E_{\max}(P)$$

Moreover, (3) is time-variant (the third argument of function g).

The OpenModelica optimization through annotations accepts as path constraints only time-invariant box constraints, so eq (3) is expressed in the simpler form:

$$\mathbf{x}_{\min} \leq \mathbf{x}(t) \leq \mathbf{x}_{\max}$$

$$\mathbf{u}_{\min} \leq \mathbf{u}(t) \leq \mathbf{u}_{\max}$$

$$g_{\min} \leq g(\mathbf{x}(t), \mathbf{u}(t)) \leq g_{\max}$$

OpenModelica uses the Radau IIA discretization scheme of order 1 or 5 depending on user input.

Using the first order the Radau IIA is equivalent to implicit Euler and to compute states, output and cost function only the values at the end of each interval are evaluated. E.g., if we have StopTime=1 and Interval=0.25, only values for t=0.25, t=0.5, t=0.75, t=1 will be considered. Therefore, the resulting value of the control variable at t=0 may be even totally wrong, since it has no influence on the result.

18.1.3 Syntax

OpenModelica provides specific annotations to allow the optimization problem to be described, as an alternative to the use of Optimica language. They are listed and commented below.

- *Request for an optimization*. We must use two simulation flags and a command line option. These can conveniently be inserted into the code, to avoid selecting them manually all the time, as follows:

```
__OpenModelica_simulationFlags(s = "optimization", optimizerNP="1"),
__OpenModelica_commandLineOptions = "+g=Optimica",
```

  OptimizerNP gives the number of colloction points and can be only 1 or 3. As already said the RadauIIA order is 2*OptimizerNP-1.

  The user is recommended to use as a first attempt optimizerNP=1. In case of questionable results, they can try optimizerNP=3.

  For the simulation, it is known that the stability ranges are different. At the same time, we lose stability with higher order (see[1]).

  Note that Optimica command-line option is added even if we do not use Optimica specific language constructs; this it is required for the use of optimization-related annotations.

- *Select optimization parameters*. We must specify StartTime, StopTime, Interval, and Tolerance. The first two have the same meaning as in time simulations. Interval not only defines the output interval (as in time simulations), but has a more specific meaning: it defines the interval between two successive collocation points. I.e., optimization is done splitting the whole timespan in sparts having interval as length. Therefore this value may have a huge effects on the simulation output. For typical runs, number of intervals values from 100 to 1000-2000 could be adequate. These values are obviously set through the experiment annotation, e.g.:

```
experiment(StartTime = 0, StopTime = 20, Tolerance = 1e-07, Interval = 0.02),
```

  the default tolerance is 1e-6. The user is warned that enlarging this value may affect the output quality adversely by large amounts (an example will be provided later). Going up to 1e-8 may be advisable in some cases.

- *Define the variable(s) to be determined by DO*. The variables the DO must determine must have the input attribute. They can have (with parameter variability) *min* and *max* attributes, which are interpreted as boxed path constraints on input. It is recommended to also define a start value for them, since it is used for initialization (see sect. *DO Initialization*). Here's an example:

[1] Hairer, Ernst and Wanner, Gerhard, Radau Methods, 2015, pp 1213-1216, DOI 10.1007/978-3-540-70529-1_139.

```
input Real u1(start=0.5, min=0.0, max = 100/par1);
input Real u2(start=0.5, min=0.0, max = 1.0);
```

- *Indicate the minimisation goal*. We can indicate whether we must just minimise a quantity, or the integral of a quantity (see (1)., as follows:

```
Real totalCost = xxx annotation(isMayer = true); //minimize totalCost(tf)
Real specificCost = xxx annotation(isLagrange = true); //minimize integral of␣
↪specificCost (tf)
```

  Several isMayer and isLagrange goals can be set. The actual goal will be the sum of all goals (isMayer goals as they are, isLagrange goals first integrated between t_0 and t_f).

  Obviously, it is possible in Modelica to use just isMayer=true also for Lagrange terms, integrating the Laplace integrand inside the model, but the internal numeric treatment will be different.

- *Describe the system*. This is done in the usual Modelica way. Here we can exploit the huge power of modelica language and tools to automatically convert a system described in physical (and possibly graphical) terms into DAE equations

- *Define path constraints*. As we said previously, they must be boxed and time-invariant. They are expressed using annotations as in the following example (taken from the full example described in the next section):

```
Real energyConstr(min = 0, max = energyMax) = storage.energy␣
↪annotation(isConstraint = true); //timespan constraint on storage energy
```

  Here, we see that the constraints are described through min and max values.

- *Define initial constraints*. These are set using the existing modelica syntax to indicate initial values

- *Define final constraints*. These are set using a specific annotation, as in the following example (taken from the full example described in the next section):

```
annotation(isFinalConstraint = true);
```

  Some special care must be taken when dealing with final constraints. We must be sure that OM front-end does not do alias elimination of the constrained variable, since in that case it could pass on bounds from final constraints to the merged variable, and these final constraints would become path constraints. To avoid this potentially harmful alias elimination we must add to the final constrant an auxiliary parameter, as follows.

```
parameter Real p = 1 "Auxiliary parameter for energy final constraint";
Real energyConstr(min = 0, max = energyMax) = storage.energy␣
↪annotation(isConstraint = true); //timespan constraint on storage energy
Real energyFinConstr(min = energyIni, max = energyIni) = p * storage.energy␣
↪annotation(isFinalConstraint = true); //final time constraint on storage energy
```

  The auxiliary parameter can also be used for scaling the constrained variable, so that it is roughly around one, so easing convergence. This could be done for instance as follows:

```
parameter Real p = 1e-3 "Auxiliary parameter for energy final constraint";
Real energyConstr(min = 0, max = energyMax) = storage.energy␣
↪annotation(isConstraint = true); //timespan constraint on storage energy
Real energyFinConstr(min = p*energyIni, max = p*energyIni) = p * storage.energy␣
↪annotation(isFinalConstraint = true); //final time constraint on storage energy
```

18.1.4 Preparing the system

To allow DO to operate in good conditions it is very important that the system has continuous derivatives.

Here we just give two examples:

- In case of a combiTimeTable is used to describe non-linear algebraic functions, it is highly recommended to use Continuous derivative for the smoothness parameter
- If we need to use the absolute value of a variable, we have derivative discontinuity around zero. This can be avoided, with negligible loss of precision, substituting $abs(x)$ with $sqrt(x^2 + \varepsilon)$, where eps is very low in comparison with the values x usually assumes during the simulation.
- Carefully consider having in the code asserts that cause simulation to stop. If for instance we have an assert that stops simulation when a variable gets outside its limiting value, it may happen that during the optimisation cycle the limit is hit and simulation is stopped, which may not be desirable. Asserts can still be left in place, adding a boolean variable, e.g.:

```
parameter Boolean insideDynOpt=true; // "true asks skipping the next assert␣
↪inside dyn. optimization";
assert(SOC <= SOCmax or insideDynOpt, "\n****\n" + "Battery is fully charged:\n"
+ "State of charge reached maximum limit (=" + String(SOCmax) + ")" + "\n****\n
↪");
```

18.1.5 Initialization

DO algorithm requires initialization. This is controlled through the simulation flag *ipopt_init*. This flag has three options: *SIM*, *CONST*, *FILE*. The preferred option can be selected adding the model the system annotation *__OpenModelica_simulationFlags*. For instance, the following example requires optimisation with *SIM* ` initialisation:

```
__OpenModelica_simulationFlags(s = "optimization", optimizerNP = "1", ipopt_
↪init= "SIM")
```

The three options operate as follows:

- *SIM* (the default). With this option, OM first makes an ordinary simulation; the simulation result is the initial "point" for the optimization. During this simulation the input is constantly kept at its value "start".
- *CONST*. With this option, the initial "point" for optimization is with all the quantities being constant, and equal to their "start" values
- *FILE*. In this case the initial point for optimization is taken from a file, created by a previous simulation, usually with a non-constant input (otherwise it would be simpler to use *SIM*). OpenModelica maps the variables between file and optimization via their name. The syntax is as in the following example:

```
__OpenModelica_simulationFlags(ipopt_init="FILE" -iif "simModel_res.mat
↪"),
```

18.1.6 Example 1: minimum time to destination

This example refers to a car, which is requested to cover the max possible distance using power from an engine which has a torque limitation and a power limitation.

The torque limitation is transformed in a maximum force that the wheels can transfer to the road to pus the car.

This is a very easy dynamic optimization problem, whose solution is the so-called bang-bang control: accelerate at the maximum possible degree, then, when half of the road is reached, decelerate with the maximum possible degree.

The code is very simple and it is as follows:

```
model BangBang2021 "Model to verify that optimization gives bang-bang optimal control"

  parameter Real m = 1;
  parameter Real p = 1 "needed for final constraints";

  Real a;
```

(continues on next page)

(continued from previous page)

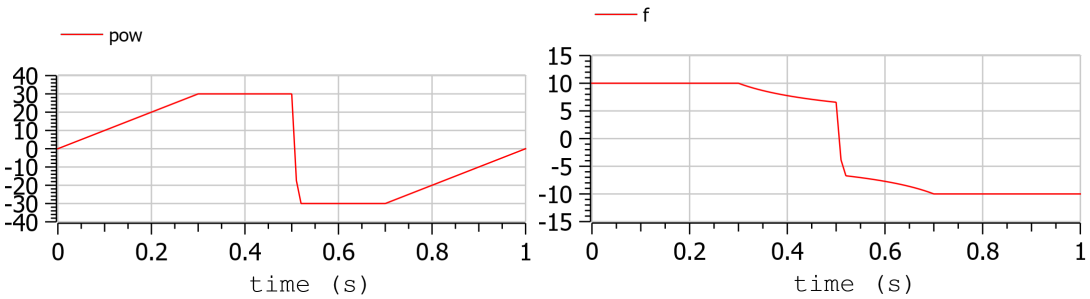
```
  Real v(start = 0);
  Real pos(start = 0);
  Real pow(min = -30, max = 30) = f * v annotation(isConstraint = true);

  input Real f(min = -10, max = 10);

  Real costPos(nominal = 1) = -pos "minimize -pos(tf)" annotation(isMayer=true);

  Real conSpeed(min = 0, max = 0) = p * v " 0<= p*v(tf) <=0"␣
↪annotation(isFinalConstraint = true);

equation

  der(pos) = v;
  der(v) = a;
  f = m * a;

annotation(experiment(StartTime = 0, StopTime = 1, Tolerance = 1e-07, Interval = 0.
↪01),
__OpenModelica_simulationFlags(s="optimization", optimizerNP="1"),
__OpenModelica_commandLineOptions="+g=Optimica");

end BangBang2021;
```

The constraint on power is especially worth considering. Above, we stated that path constraints can be $g_{\min} \leq g(\mathbf{x}(t), \mathbf{u}(t), t) \leq g_{\max}$. Here we have box limits on *pow*, which are expressed as limits on *g(v,f) = f*v = pow*

$$-30 \leq v * f = f * v \leq 30$$

This usage of constraints allows implementing variable (non-boxed) constraints, which are not allowed explicitly. Doing this the above code implements through *g(.)* a variable (so non-boxed) limit on force *f*.

The results can be expressed in terms of force and power applied to the vehicle. They are as follows:



and they are as expected.

In this model we don't use the Modelica capability to automatically determine the system equations from the graphical description of a system. In other words, the above general formulation $0 = f(\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{u}(t), t)$ is explicitly written as follows:

```
der(pos) = v;
der(v) = a;
```

In the following example, we will use this capability extensively.

18.1.7 Example 2: hybrid vehicle minimum consumption

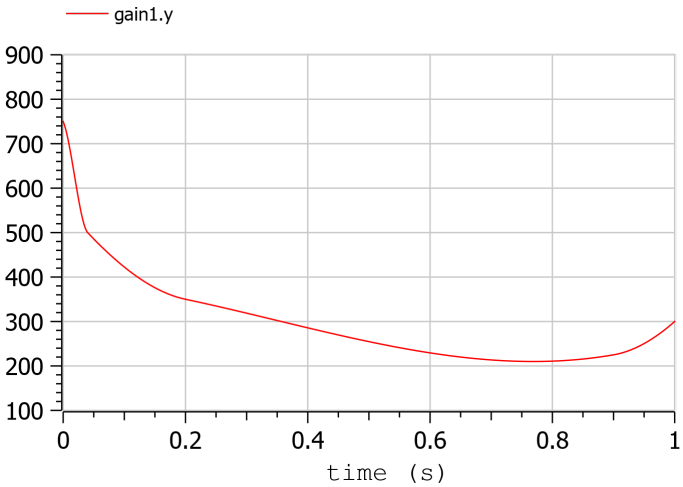
This example refers to the electricity generation of a hybrid vehicle. These vehicles can choose at any time which amount of the propulsion power must be taken from a battery and which from the Internal combustion engine.

In this example the engine can be switched ON and off without penalty, so the DO can choose both when the ICE must be ON /OFF; and the power it must deliver when it is ON.

Objective of the control is to minimise the fuel consumption. This must be done keeping the energy inside the storage at the final time, equal to the one at t=0 (otherwise it is easy to have zero consumption: just keep the Internal Combustion Engine OFF all the time!)

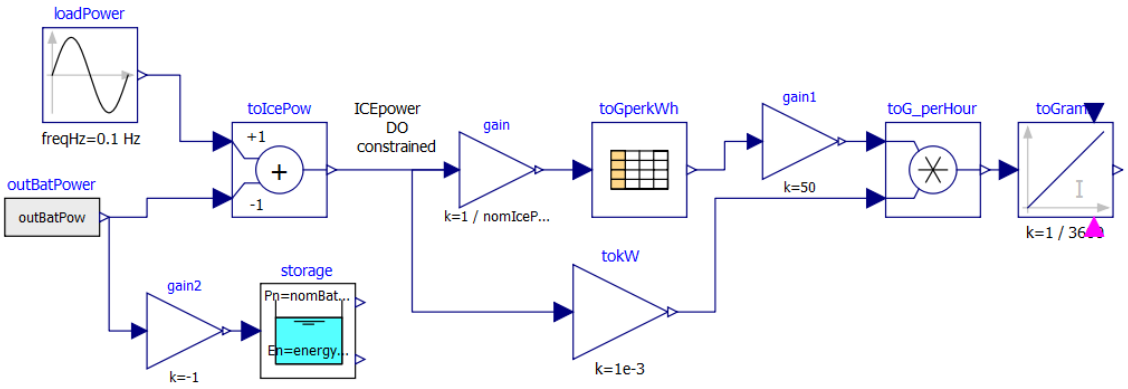
For simplicity's sake, the propulsion power, in this simple example is taken as being a sine wave plus an offset (needed to make the average positive). When the power is positive the wheels transfer power to the road, when negative they recover it (storing it into the battery).

To find the optimum, a fuel consumption curve is added, as follows:



The minimum is 210 g/kWh, and occurs when the ICE power is at the 76.8 % of the nominal power

The system diagram is as follows



The DO algorithm is required to determine the battery power outBatPower (positive when the battery delivers power) so that to minimise the fuel consumption toGrams. Block toGperkWh is normalised, so that it can be used for different engines, adapting the horizontal scale through gain, and the vertical's through gain1.

This diagram defines the system, whose equations will be automatically determined by OpenModelica, through model flattening. However, some code must be manually written to perform DO.

Here the code is as follows:

```
//*** DO-related rows

input Real outBatPow;

//
Real totalCost = toGrams.y "minimize totalCost(tf)" annotation( isMayer=true);
//

Real energyConstr(min = 0, max = energyMax) = storage.energy annotation(isConstraint␣
↪= true); //timespan constraint on storage energy

parameter Real fecp = 1 "final energy constraint parameter";

Real energyFinConstr(min = energyIni, max = energyIni) = fecp * storage.energy␣
↪annotation(isFinalConstraint = true);//final time constraint on storage energy

Real icePowerConstr(min = 0) = itoIcePow.y annotation(isConstraint=true); //timespan␣
↪constraint on Ice power

//*** End of DO-related rows
```

A few comments:

- The choice isMayer=true on the objective function totalCost requires its final value to be actually minimised, not its integral (as would have been in case of the keyword isLaplace=true)
- We have two different constraints on the storage energy: the storage energy must all the time be between 0 and the maximum allowed, and at the end of the simulation must be brought back to its initial value.
- ICE can only deliver power, not absorb; so, we expect all the (regenerative) braking power and energy to be sent into the storage

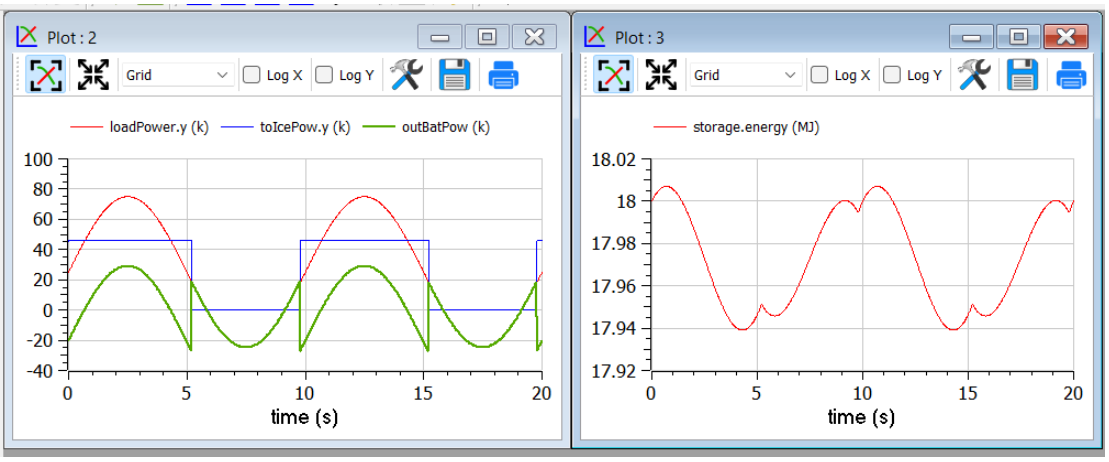
18.1.8 Ideal storage

Here we consider the storage to be ideal: the flow of power in and out causes no losses to occur

In this case the solution of our optimization problem is trivial:

- The Ice must supply the average power requested by the load, and when it does this it must do it at the optimal point which, as seen above is when its power is at the 76.8% of its nominal value
- The battery supplies the load power minus ICE power.

Using iceNominalPower=60 kW we get the following output:



We see, as expected that the ICE is switched ON and OFF; and when it is ON it delivers at its 76.8 % of nominal power, i.e. at 46.1 kW. The battery delivers the difference, and when the load is negative absorbs all the power

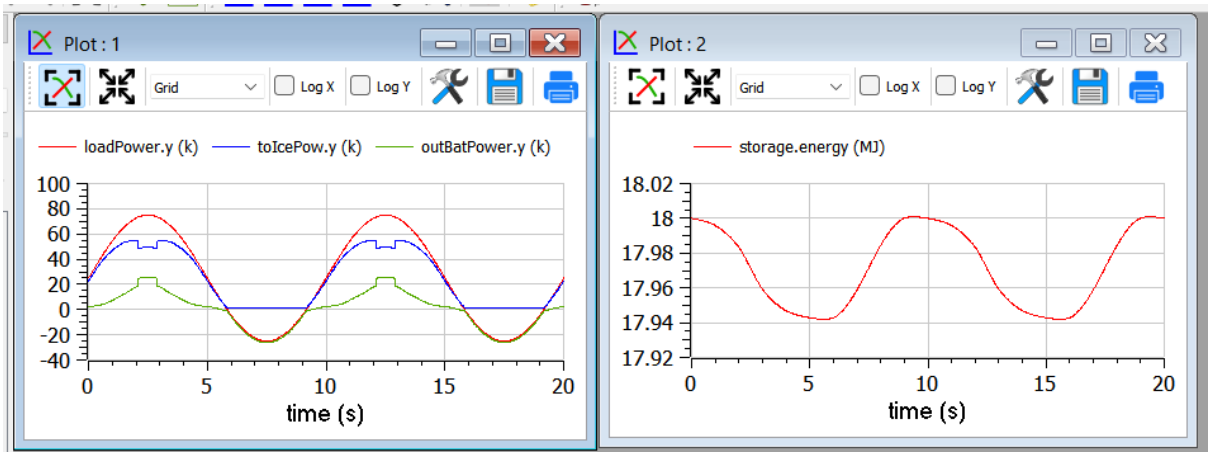
from it. The control is such that the energy at the end of the transient is the same as the one at t=0.

This result is good and confirms what we expected.

Effects of tolerance

We mentioned that reduction of tolerance may affect the result adversely by large, especially when the minimum, as in this case is very flat (since the specific fuel consumption curve used for our example is very flat).

In the following picture we see the result of the previous section as it appears when we release tolerance by changing it from 1e-7 to and 1e-6. Now the result is badly wrong (and the total cost has changed from to 25.6g 29.9g).



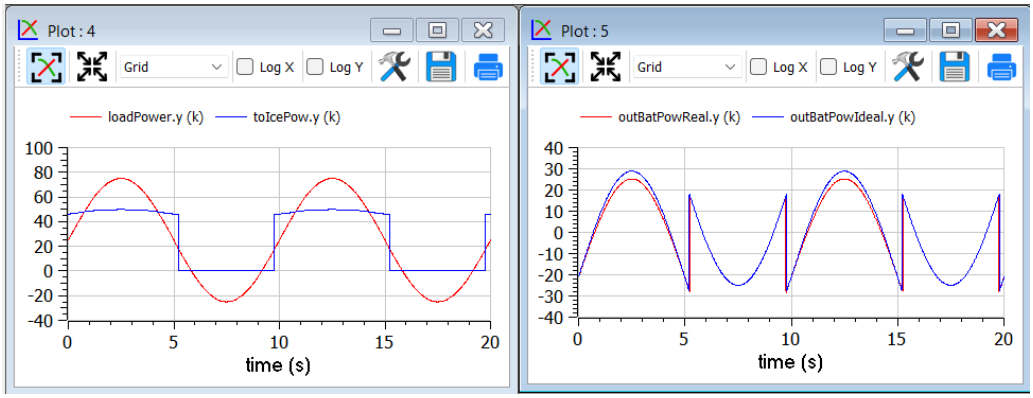
More realistic storage

To the DO to be useful, it must obviously go beyond what is exactly expected. Therefore, we repeat the simulation adding the simulation of some losses inside the battery. According to scientific literature, losses here are modelled through the following formula:

$$L(t) = 0.03|P_{\text{batt}}(t)| + 0.04\frac{P_{\text{batt}}^2(t)}{P_{batt,nom}}$$

Which reflects that they in part are proportional to the absolute value of battery current, partly to its square. The coefficients are typical for power electronic converters interfacing a battery.

Inside the code, however, the formula introduced is structurally different, since it has been transformed to avoid the derivative discontinuity of the absolute value of a quantity around zero, using a trick like the one reported in sect. 1.4.

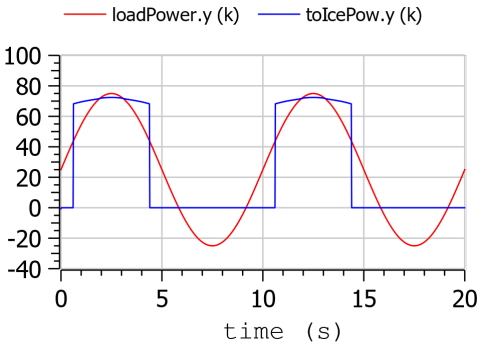


We see that now the optimiser changes the Ice power when in ON state, to reduce the battery power at its peak, since the losses formula pushes towards lower powers.

Adding storage power limitation

As a last case for example 2, we ass tome limitation on the power that can be exchanged by the battery. This can be physically due to limitations of either the battery or the inverter connected to it.

To show better the effect, we first rise the ICE power to 100 kW, so that the interval of ICE operation is smaller:



Then we change the following row of code:

```
input Real outBatPow;
```

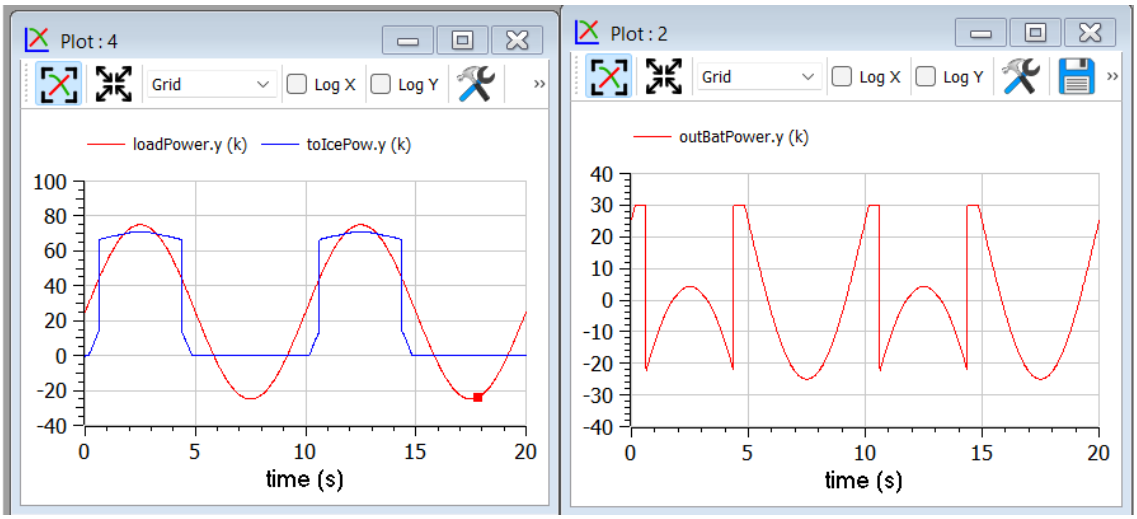
into:

```
input Real outBatPow(min = -maxBatPower, max = maxBatPower);
```

where

```
parameter Modelica.SIunits.Power maxBatPower = 30e3;
```

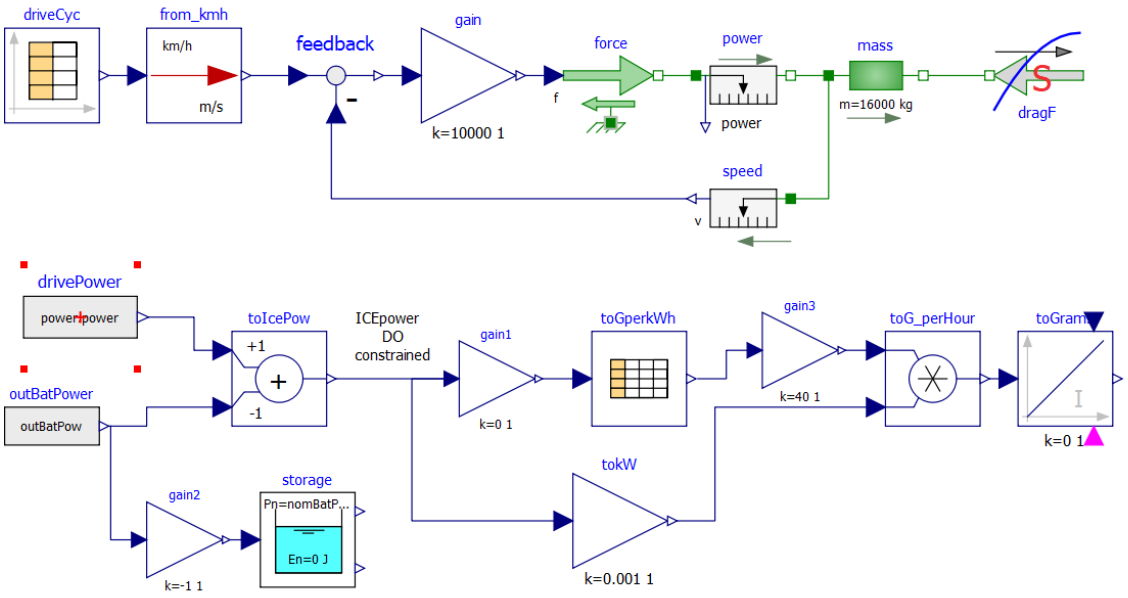
giving rise to the following results (with a much longer computation time than in the previous cases):



18.1.9 Example 3: Acausal vehicle

As a last example, we replicate the optimization of Example 2, but the power to be delivered directly deriving from simulation of a vehicle, modelled through its physical elements.

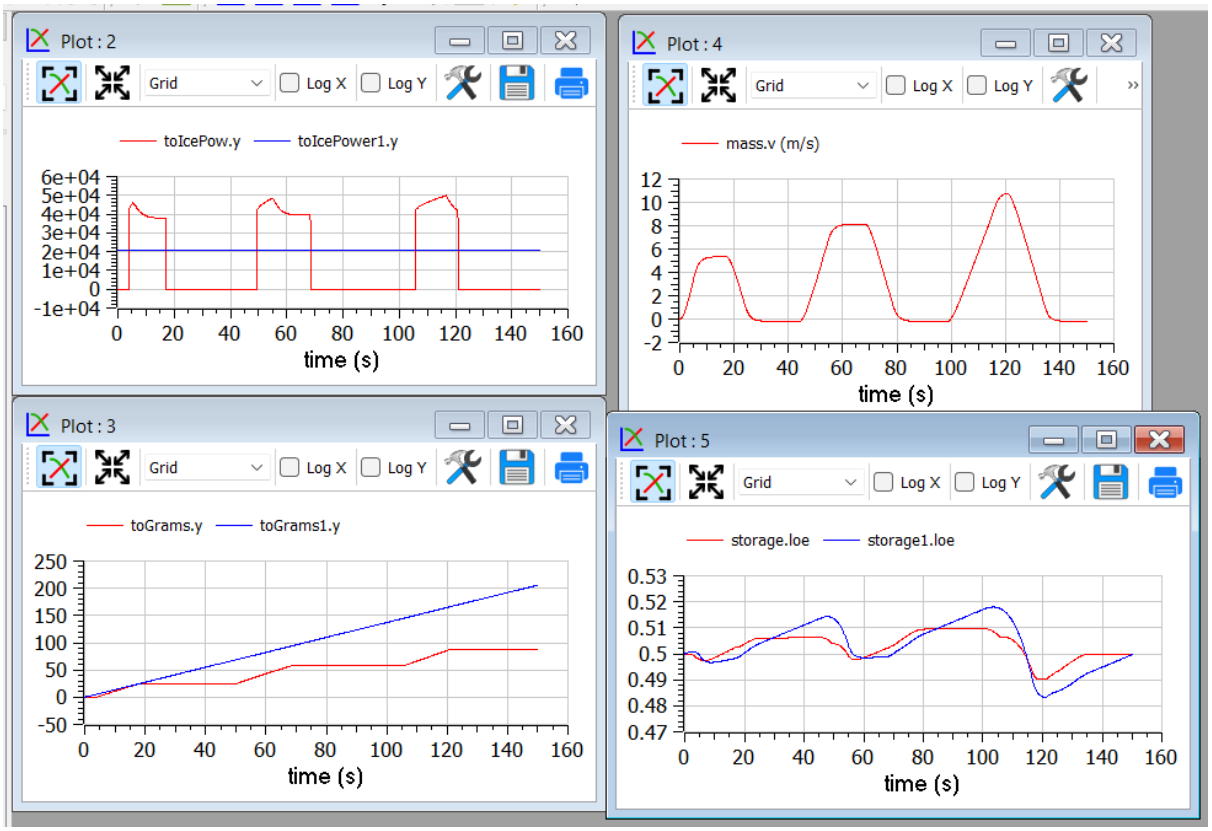
The considered diagram is as follows:



The upper part contains a vehicle model. The model follows a speed profile defined by the drive cycle driveCyc, through a simple proportional controller (simulating the driver). The power is applied to a mass; the drag force dragF sis the force against the movement due to friction (independent on speed) and air resistance (proportional to the square of vehicle speed).

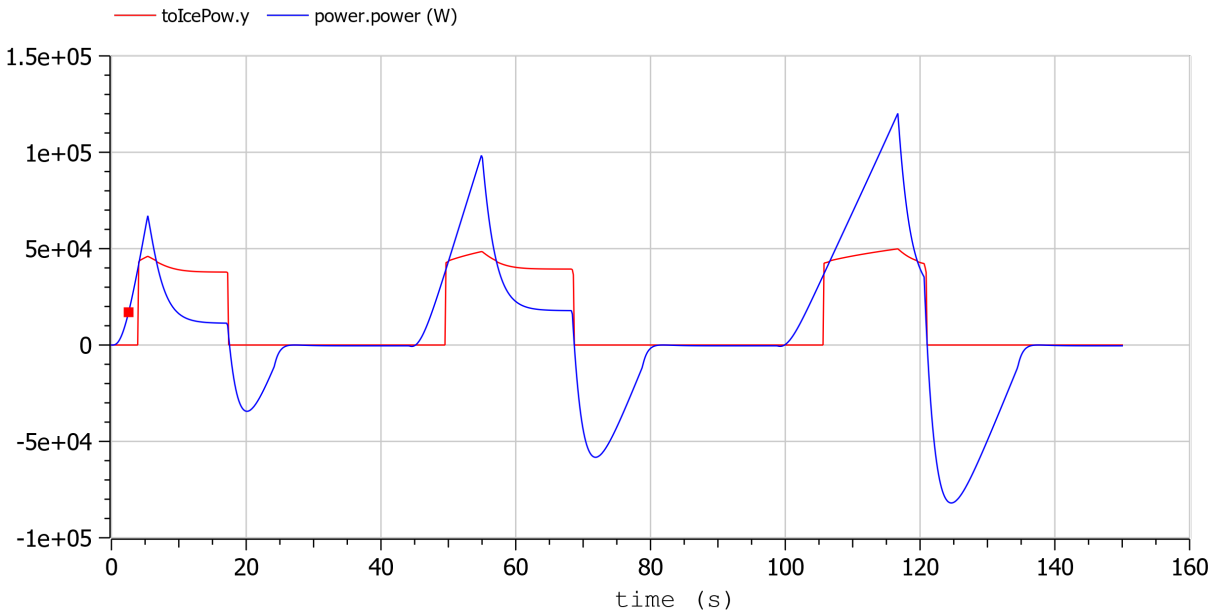
The lower part contains the management of the storage, and the optimization algorithm, already discussed in example 2.

The results are shown in the following picture, where the obtained cost (red curve) is compared to what obtainable in case the ICE is continuously kept ON (blue curve), at a power (blue curve) that allows the battery energy at the end of the simulation to be equal to the one as t=0, as in the case of the optimised solution.



This example shows that the optimizer can find an ON/OFF strategy that more than halves the hybrid vehicle fuel consumption.

The following plot shows the ICE power in comparison with total power needed to cover the given trip profile mass.v. The rest is supplied by the battery.



18.2 Built-in Dynamic Optimization using Optimica language extensions

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

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

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

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

18.2.1 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.

18.2.2 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.

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 using command line terminals similar to the options described below:

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

Listing 18.1: BatchReactor.mo

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

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

```
>>> optimize(nmpcBatchReactor, numberOfIntervals=16, stopTime=1, tolerance=1e-8)
record SimulationResult
    resultFile = "«DOCHOME»/nmpcBatchReactor_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 16,␣
↪tolerance = 1e-8, method = 'optimization', fileNamePrefix = 'nmpcBatchReactor',␣
↪options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = '
↪'",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.

Optimizer Variables
=============================================================
State[0]:x1(start = 1, nominal = 1, min = 0, max = 1, init = 1)
State[1]:x2(start = 0, nominal = 1, min = 0, max = 1, init = 0)
Input[2]:u(start = 0, nominal = 5, min = 0, max = 5)
-------------------------------------------------------------
number of nonlinear constraints: 0
=============================================================

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

LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.020553815,
    timeBackend = 0.010454372,
    timeSimCode = 0.0017541470000000002,
    timeTemplates = 0.002680038,
    timeCompile = 0.16623945299999998,
    timeSimulation = 0.009089205000000001,
    timeTotal = 0.2108736029999999
end SimulationResult;
```

The control and state trajectories of the optimization results:

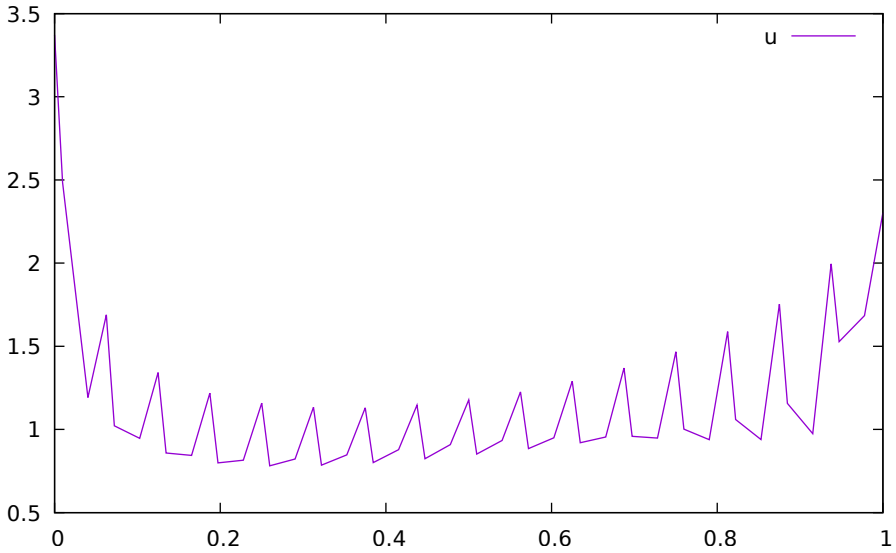


Figure 18.1: Optimization results for Batch Reactor model - input variables.

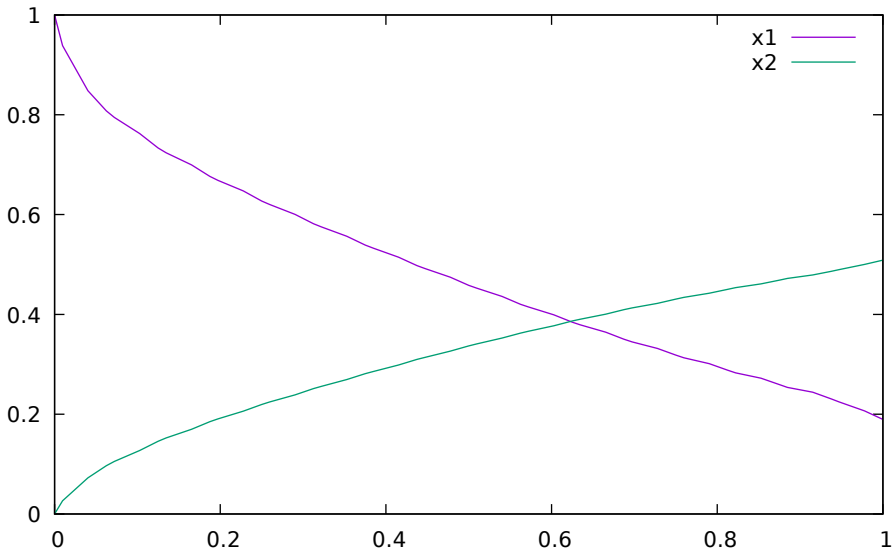


Figure 18.2: Optimization results for Batch Reactor model - state variables.

18.2.3 Different Options for the Optimizer IPOPT

Table 18.1: New meanings of the usual simualtion options for Ipopt.

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

Table 18.2: New simulation options for Ipopt.

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

18.3 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.

18.3.1 Compiling the Modelica code

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

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

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

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

18.3.2 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.

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.

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

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

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

```
>>> translateModelXML(BatchReactor)
"«DOCHOME»/BatchReactor.xml"
```

This will generate an XML file named *BatchReactor.xml* (Listing 18.2) that contains a symbolic representation of the optimal control problem and can be inspected in a standard XML editor.

Listing 18.2: BatchReactor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenModelicaModelDescription
  xmlns:exp="https://github.com/JModelica/JModelica/tree/master/XML/daeExpressions.xsd
↪"
  xmlns:equ="https://github.com/JModelica/JModelica/tree/master/XML/daeEquations.xsd"
  xmlns:fun="https://github.com/JModelica/JModelica/tree/master/XML/daeFunctions.xsd"
  xmlns:opt="https://github.com/JModelica/JModelica/tree/master/XML/daeOptimization.
↪xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  fmiVersion="1.0"
  modelName="BatchReactor"
  modelIdentifier="BatchReactor"
  guid="{a3e18813-a5ff-4ed4-ae28-3d4211b5f5b0}"
  generationDateAndTime="2026-09-22T08:25:24"
  variableNamingConvention="structured"
  numberOfContinuousStates="2"
  numberOfEventIndicators="0"
  >

  <VendorAnnotations>
    <Tool name="OpenModelica Compiler v1.28.0-dev.838+g4c2a253e5a.cmake"> </Tool>
  </VendorAnnotations>


  <ModelVariables>
    <ScalarVariable name="x1"  valueReference="0" variability="continuous" causality=
↪"local" alias="noAlias">
      <Real start="1.0" fixed="true" min="0.0" max="1.0"  />
      <QualifiedName>
        <exp:QualifiedNamePart name="x1"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>state</VariableCategory>
    </ScalarVariable>

    <ScalarVariable name="x2"  valueReference="1" variability="continuous" causality=
↪"local" alias="noAlias">
      <Real start="0.0" fixed="true" min="0.0" max="1.0"  />
      <QualifiedName>
        <exp:QualifiedNamePart name="x2"/>
      </QualifiedName>
    <isLinearTimedVariables>
```

(continues on next page)

(continued from previous page)

```
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>state</VariableCategory>
    </ScalarVariable>
    <ScalarVariable name="der(x1)"  valueReference="2" variability="continuous"␣
↪causality="local" alias="noAlias">
      <Real     />
      <QualifiedName>
        <exp:QualifiedNamePart name="x1"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>derivative</VariableCategory>
    </ScalarVariable>

    <ScalarVariable name="der(x2)"  valueReference="3" variability="continuous"␣
↪causality="local" alias="noAlias">
      <Real     />
      <QualifiedName>
        <exp:QualifiedNamePart name="x2"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>derivative</VariableCategory>
    </ScalarVariable>
    <ScalarVariable name="u"  valueReference="4" variability="continuous" causality=
↪"input" alias="noAlias">
      <Real  min="0.0" max="5.0"  />
      <QualifiedName>
        <exp:QualifiedNamePart name="u"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>algebraic</VariableCategory>
    </ScalarVariable>
  </ModelVariables>

  <equ:BindingEquations>
  </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>
```

(continues on next page)

(continued from previous page)

```
          <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:Mul>
            <exp:RealLiteral>-0.5</exp:RealLiteral>
            <exp:Pow>
              <exp:Identifier>
                <exp:QualifiedNamePart name="u"/>
              </exp:Identifier>
              <exp:RealLiteral>2.0</exp:RealLiteral>
            </exp:Pow>
          </exp:Mul>
          <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:Equation>
    <exp:Sub>
      <exp:Identifier>
```

(continues on next page)

(continued from previous page)

```
          <exp:QualifiedNamePart name="x1"/>
        </exp:Identifier>
        <exp:Identifier>
          <exp:QualifiedNamePart name="$START"/>
          <exp:QualifiedNamePart name="x1"/>
        </exp:Identifier>
      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>

      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>

      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>
        <exp:Identifier>
          <exp:QualifiedNamePart name="x2"/>
        </exp:Identifier>
        <exp:Identifier>
          <exp:QualifiedNamePart name="$START"/>
          <exp:QualifiedNamePart name="x2"/>
        </exp:Identifier>
      </exp:Sub>
    </equ:Equation>
  </equ:InitialEquations>

  <fun:Algorithm>
  </fun:Algorithm>

  <fun:RecordsList>
  </fun:RecordsList>

  <fun:FunctionsList>
  </fun:FunctionsList>

  <opt:Optimization>
    <opt:TimePoints>
      <opt:TimePoint  >
      </opt:TimePoint>
    </opt:TimePoints>
    <opt:PathConstraints>
    </opt:PathConstraints>
  </opt:Optimization>

</OpenModelicaModelDescription>
```

18.3.3 XML Import to CasADi via OpenModelica Python Script

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

The SymbolicOCP class contains symbolic representation of the optimal control problem designed to be general

and allow manipulation. For a more detailed description of this class and its functionalities, we refer to the API documentation of CasADi.

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

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

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

2. Perform compilation and generate the XML file

   a. Go to your working directory

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

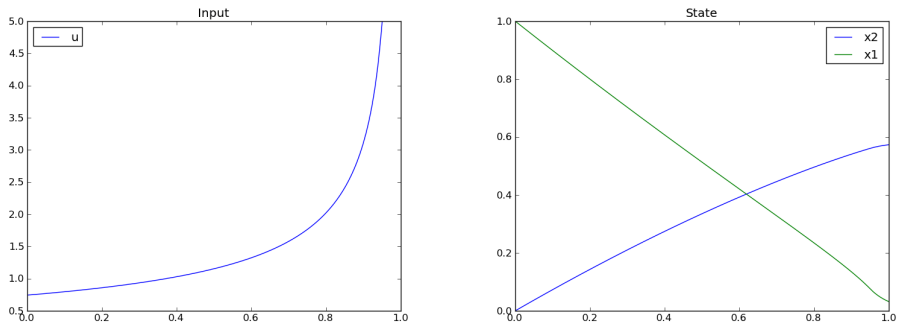
a. Go to omc path from working directory and run the following command

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

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

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

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



18.4 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.

18.4.1 Preparing the Model

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

Parameters

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

For all intended parameters, please note that:

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

Constraints

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

For instance, if mechanical stress must be less than 5 $N.m^{-2}$, one should write in the model:

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

If during simulation, the variable *mechanicalStress* exceeds 5 $N.m^{-2}$, the simulation will stop and be considered as a failure.

Objectives

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

18.4.2 Set problem in OMOptim

Launch OMOptim

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

Create a new project

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

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

Load models

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

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

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

You can load as many models as you want.

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

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

Dependencies

OMOptim should detect dependencies and load corresponding files. However, 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 18.3.

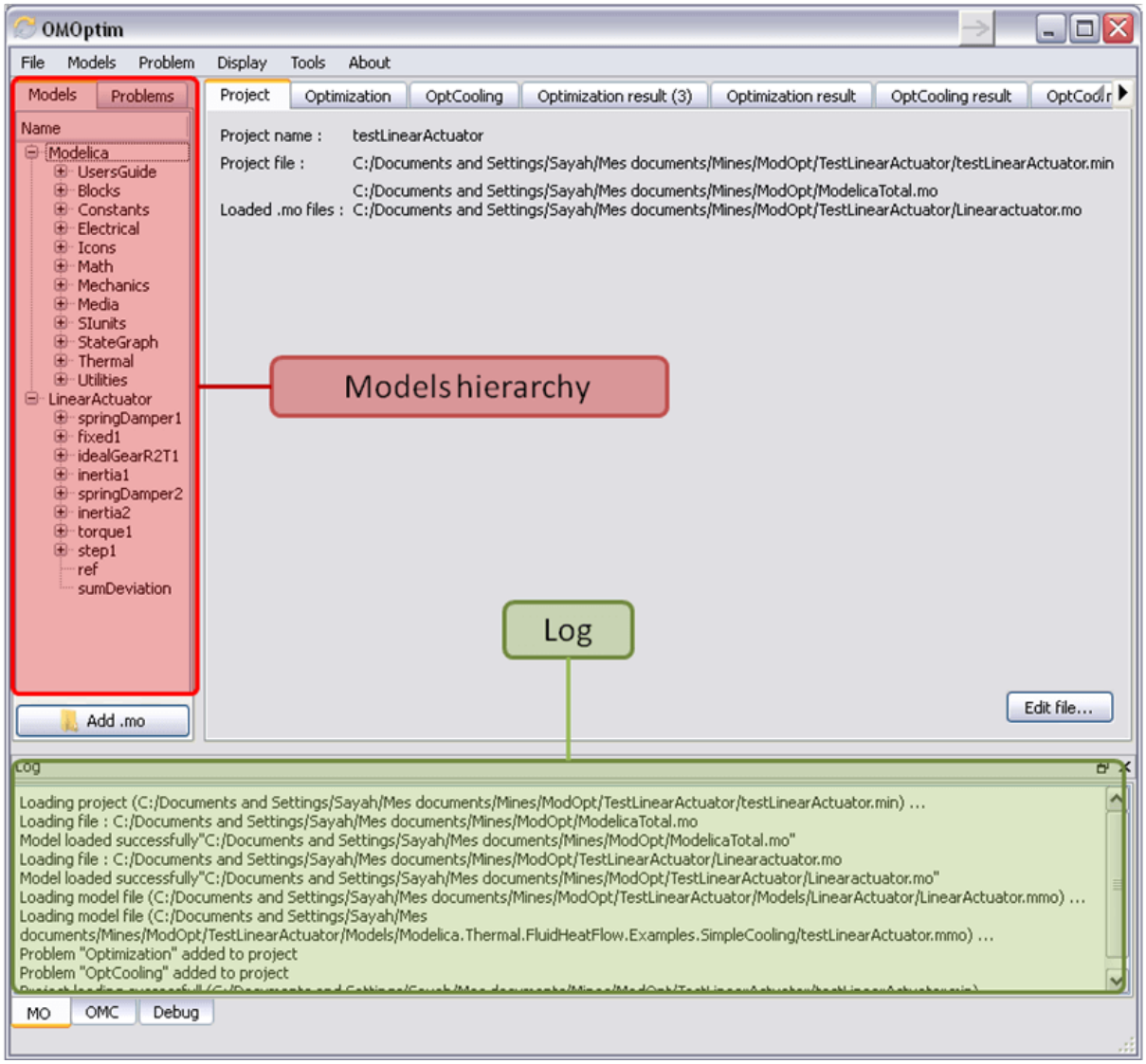


Figure 18.3: OMOptim window after having loaded model.

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.

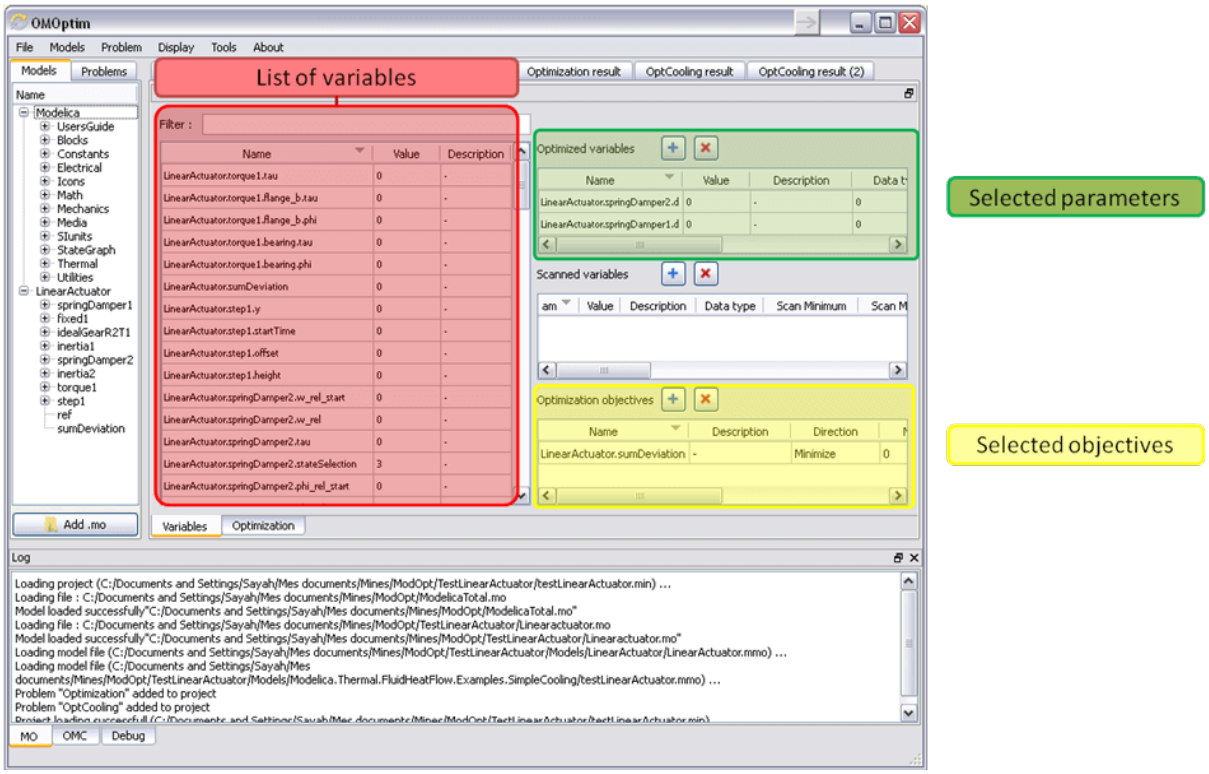


Figure 18.4: Forms for defining a new optimization problem.

List of Variables is Empty

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

Select Optimized Variables

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

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

Select objectives

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

For each objective, you must:

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

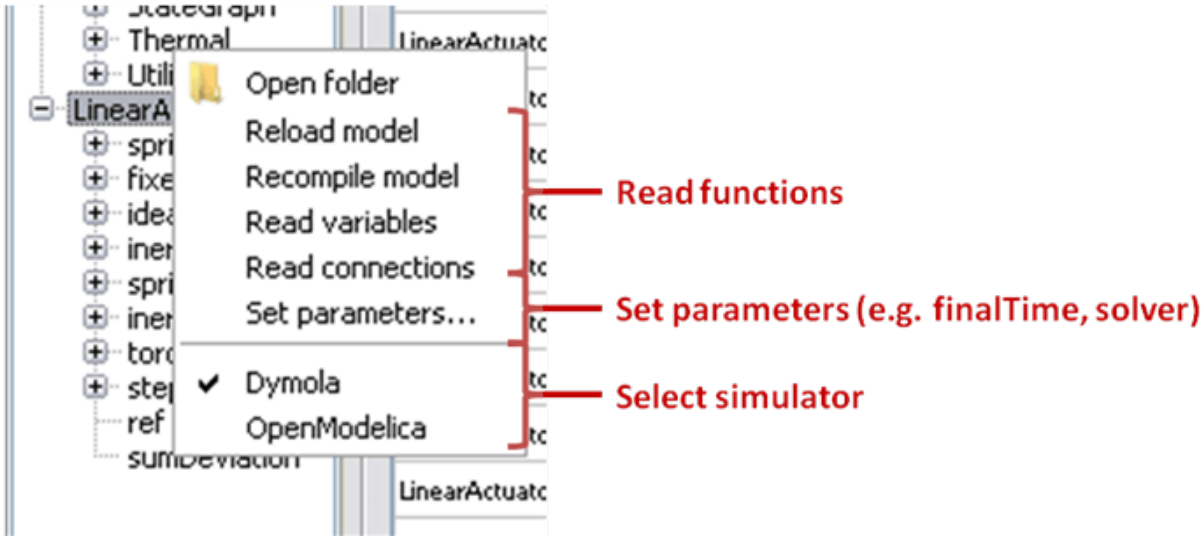


Figure 18.5: Selecting read variables, set parameters, and selecting simulator.

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

Select and configure algorithm

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

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

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

- ***Population size*: this is the number of configurations kept after a**
  generation. If it is set to 50, your final result can't contain more than 50 different points.
- ***Off spring rate*: this is the number of children per adult obtained**
  after combination process. If it is set to 3, each generation will contain 150 individual (considering population size is 50).
- ***Max generations*: this number defines the number of generations**
  after which optimization should stop. In our case, each generation corresponds to 150 simulations. Note that you can still stop optimization while it is running by clicking on *stop* button (which will appear once optimization is launched). Therefore, you can set a really high number and still stop optimization when you want without losing results obtained until there.
- ***Save frequency*: during optimization, best configurations can be**
  regularly saved. It allows to analyze evolution of best configurations but also to restart an optimization from previously obtained results. A Save Frequency parameter set to 3 means that after three generations, a file is automatically created containing best configurations. These files are named iteraion1.sav, iteration2.sav and are store in *Temp* directory, and moved to *SolvedProblems* directory when optimization is finished.
- ***ReinitStdDev*: this is a specific parameter of EAAdapt1. It defines**
  whether standard deviation of variables should be reinitialized. It is used only if you start optimization from previously obtained configurations (using *Use start file* option). Setting it to yes (1) will, in most of cases, lead to a spread research of optimized configurations, forgetting parameters' variations' reduction obtained in previous optimization.

Use start file

As indicated before, it is possible to pursue an optimization finished or stopped. To do this, you must enable *Use start file* option and select file from which optimization should be started. This file is an *iteration_.sav* file created in previous optimization. It is stored in corresponding *SolvedProblems* folder (*iteration10.sav* corresponds to the tenth generation of previous optimization).

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

Launch

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

Stopping Optimization

Optimization will be stopped when the generation counter will reach the generation number defined in parameters. However, you can still stop the optimization while it is running without 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.

18.4.3 Results

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

Obtaining all Variable Values

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

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

Window Regions in OMOptim GUI

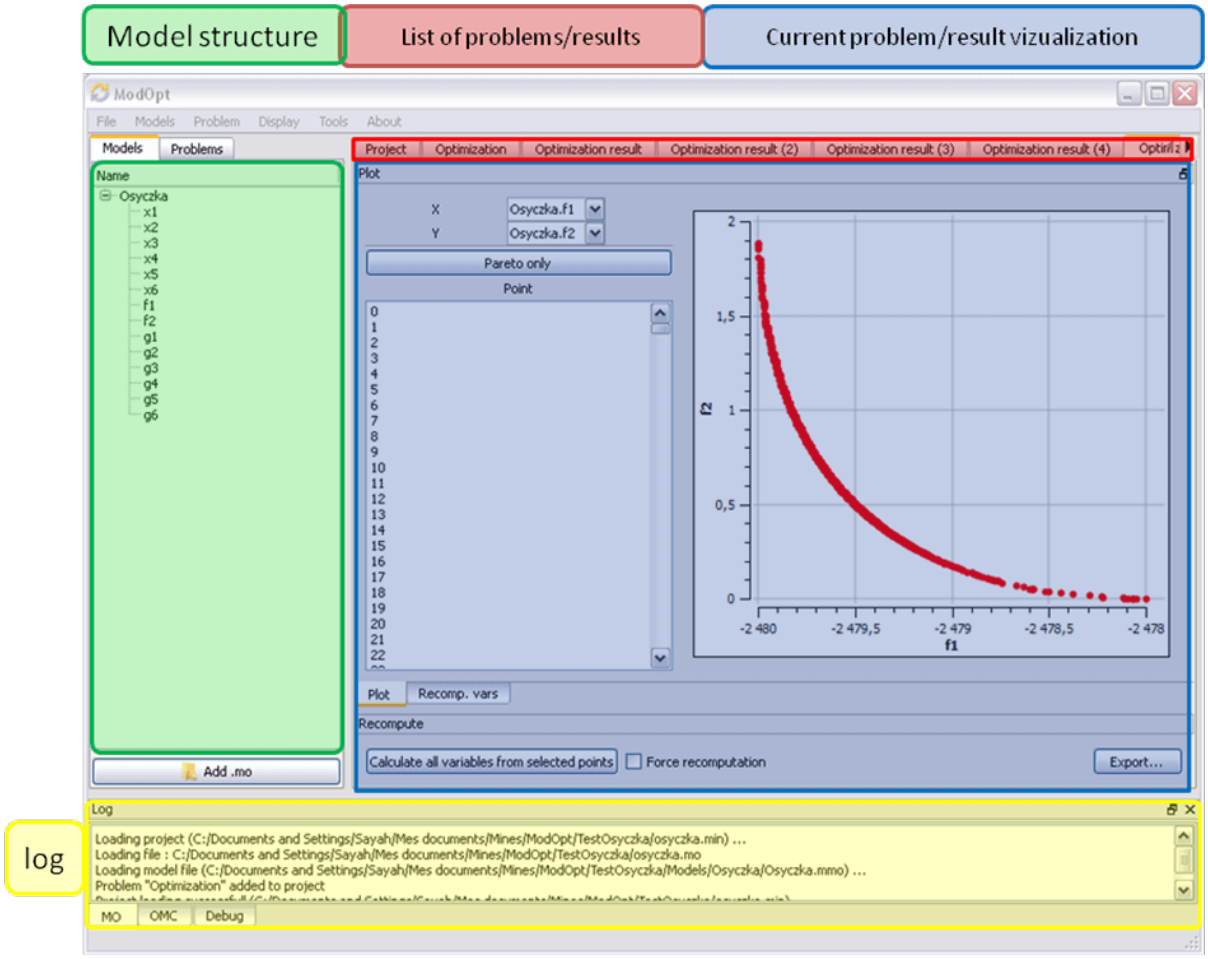


Figure 18.6: Window regions in OMOptim GUI.

CHAPTER

NINETEEN

PARAMETER SENSITIVITIES WITH OPENMODELICA

This section describes the use of OpenModelica to compute parameter sensitivities using forward sensitivity analysis together with the Sundials/IDA solver.

19.1 Single Parameter sensitivities with IDA/Sundials

19.1.1 Background

Parameter sensitivity analysis aims at analyzing the behavior of the corresponding model states w.r.t. model parameters.

Formally, consider a Modelica model as a DAE system:

$$F(x, \dot{x}, y, p, t) = 0 \; x(t_0) = x_0(p)$$

where $x(t) \in \mathbf{R}^n$ represent state variables, $\dot{x}(t) \in \mathbf{R}^n$ represent state derivatives, $y(t) \in \mathbf{R}^k$ represent algebraic variables, $p \in \mathbf{R}^m$ model parameters.

For parameter sensitivity analysis the derivatives

$$\frac{\partial x}{\partial p}$$

are required which quantify, according to their mathematical definition, the impact of parameters p on states x. In the Sundials/IDA implementation the derivatives are used to evolve the solution over the time by:

$$\dot{s}_i = \frac{\partial x}{\partial p_i}$$

19.1.2 An Example

This section demonstrates the usage of the sensitivities analysis in OpenModelica on an example. This module is enabled by the following OpenModelica compiler flag:

```
>>> setCommandLineOptions("--calculateSensitivities")
true
```

Listing 19.1: LotkaVolterra.mo

```
model LotkaVolterra
  Real x(start=5, fixed=true),y(start=3, fixed=true);
  parameter Real mu1=5,mu2=2;
  parameter Real lambda1=3,lambda2=1;
equation
  0 = x*(mu1-lambda1*y) - der(x);
  0 = -y* (mu2 -lambda2*x) - der(y);
end LotkaVolterra;
```

Also for the simulation it is needed to set `IDA` as solver integration method and add a further simulation flag `-idaSensitivity` to calculate the parameter sensitivities during the normal simulation.

```
>>> simulate(LotkaVolterra, method="ida", simflags="-idaSensitivity")
record SimulationResult
    resultFile = "«DOCHOME»/LotkaVolterra_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'ida', fileNamePrefix = 'LotkaVolterra', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = '-
↪idaSensitivity'",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0016262870000000001,
    timeBackend = 0.004616898,
    timeSimCode = 0.00119701,
    timeTemplates = 0.002024826,
    timeCompile = 0.160668522,
    timeSimulation = 0.006341019000000001,
    timeTotal = 0.17656481200000002
end SimulationResult;
```

Now all calculated sensitivities are stored into the results mat file under the $Sensitivities block, where all currently every **top-level** parameter of the Real type is used to calculate the sensitivities w.r.t. **every state**.

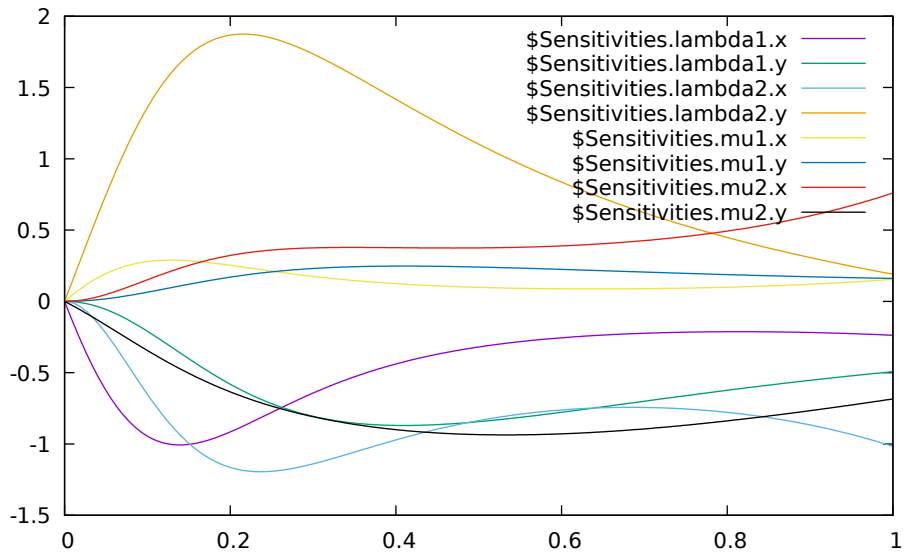


Figure 19.1: Results of the sensitivities calculated by IDA solver.

19.2 Single and Multi-parameter sensitivities with OMSens

OMSens is an OpenModelica sensitivity analysis and optimization module.

19.2.1 Installation

Follow the install instructions described on the OMSens github page.

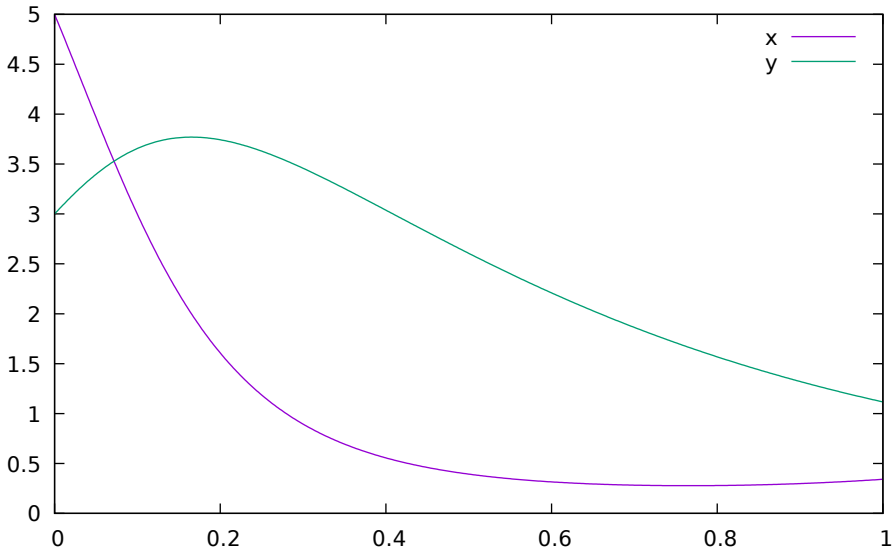


Figure 19.2: Results of the LotkaVolterra equations.

19.2.2 Usage

OMSens offers 3 flavors for parameter sensitivity analysis.

- Individual Sensitivity Analysis
  - Used to analyze how a parameter affects a variable when perturbed on its own
- Multi-parameter Sweep
  - Exploratory experimentation that sweeps the space of a set of parameters
- Vectorial Sensitivity Analysis
  - Used to find the combination of parameters that maximizes/minimizes a state variable

As an example, we choose the Lotka-Volterra model that consists of a second-order nonlinear set of ordinary differential equations. The system models the relationship between the populations of predators and preys in a closed ecosystem.

```
model LotkaVolterra "This is the typical equation-oriented model"
  parameter Real alpha=0.1 "Reproduction rate of prey";
  parameter Real beta=0.02 "Mortality rate of predator per prey";
  parameter Real gamma=0.4 "Mortality rate of predator";
  parameter Real delta=0.02 "Reproduction rate of predator per prey";
  parameter Real prey_pop_init=10 "Initial prey population";
  parameter Real pred_pop_init=10 "Initial predator population";
  Real prey_pop(start=prey_pop_init) "Prey population";
  Real pred_pop(start=pred_pop_init) "Predator population";
initial equation
  prey_pop = prey_pop_init;
  pred_pop = pred_pop_init;
equation
  der(prey_pop) = prey_pop*(alpha-beta*pred_pop);
  der(pred_pop) = pred_pop*(delta*prey_pop-gamma);
end LotkaVolterra;
```

Let's say we need to investigate the influence of model parameters on the predator population at 40 units of time. We assume a +/-5% uncertainty on model parameters.

We can use OMSens to study the sensitivity model to each parameter, one at a time.

Open the Lotka-Volterra model using OMEdit.

Individual Sensitivity Analysis

- Select *Sensitivity Optimization* > *Run Sensitivity Analysis and Optimization* from the menu. A window like the one below should appear. Windows users should use the default python executable that comes with OpenModelica installation i.e., they don't need to change the proposed python executable path. If you want to use some other python installation then make sure that all the python dependencies are installed for that python installation.

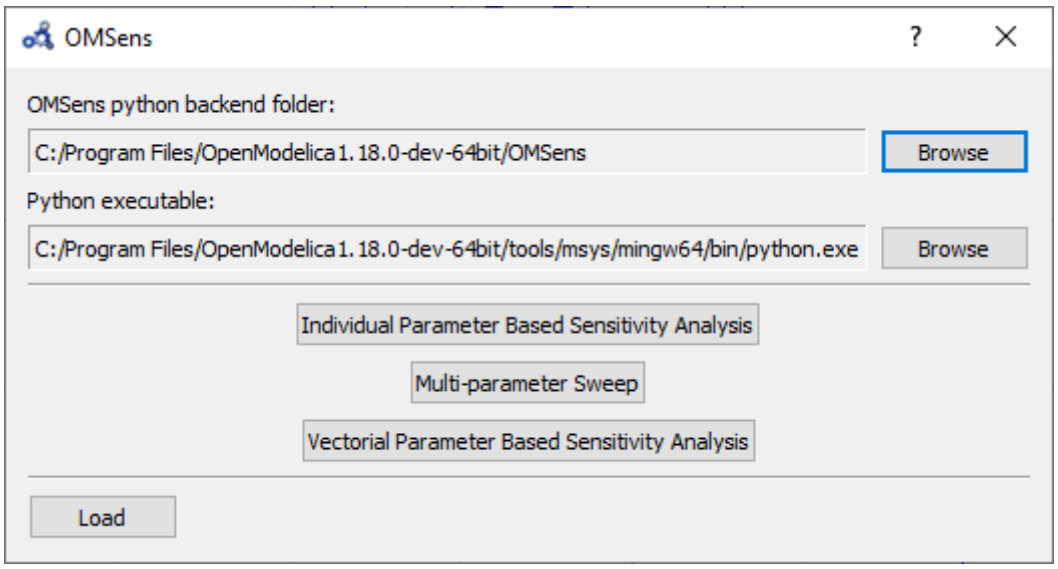


Figure 19.3: OMSens window.

- Choose **Individual Parameter Based Sensitivity Analysis** and set up the simulation settings.

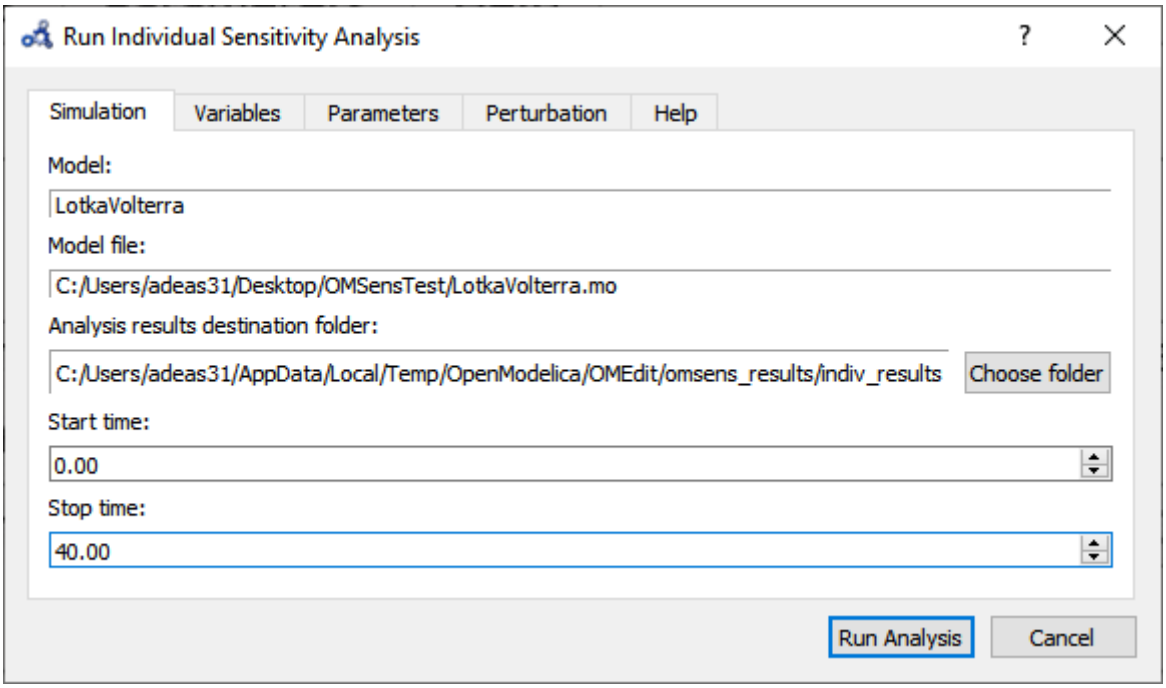


Figure 19.4: Run individual sensitivity analysis.

- Select variables.
- Select parameters.

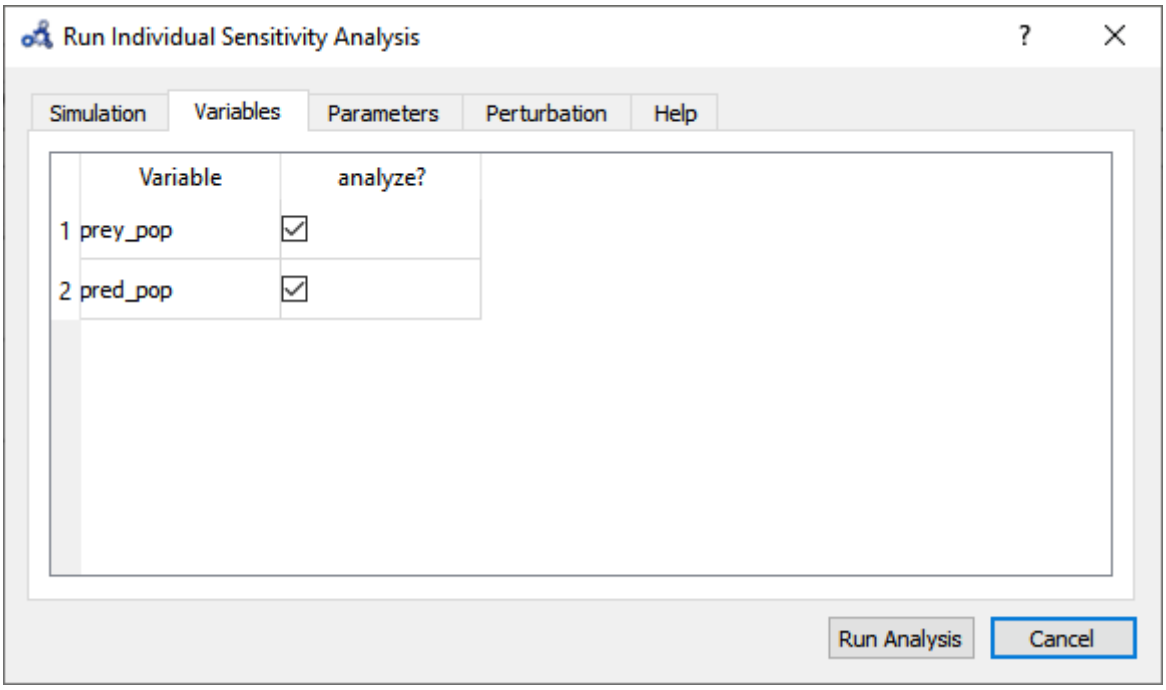


Figure 19.5: Individual sensitivity analysis variables.

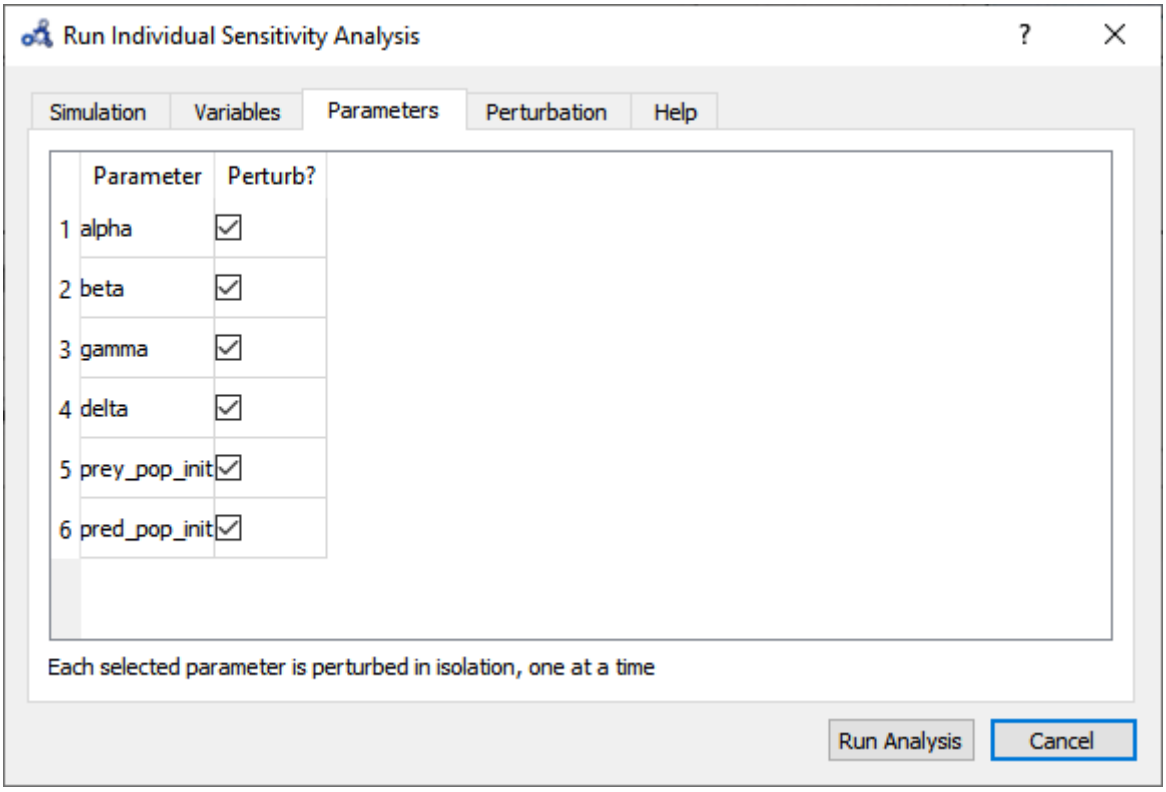


Figure 19.6: Individual sensitivity analysis parameters.

- Choose the perturbation percentage and direction. Run the analysis.

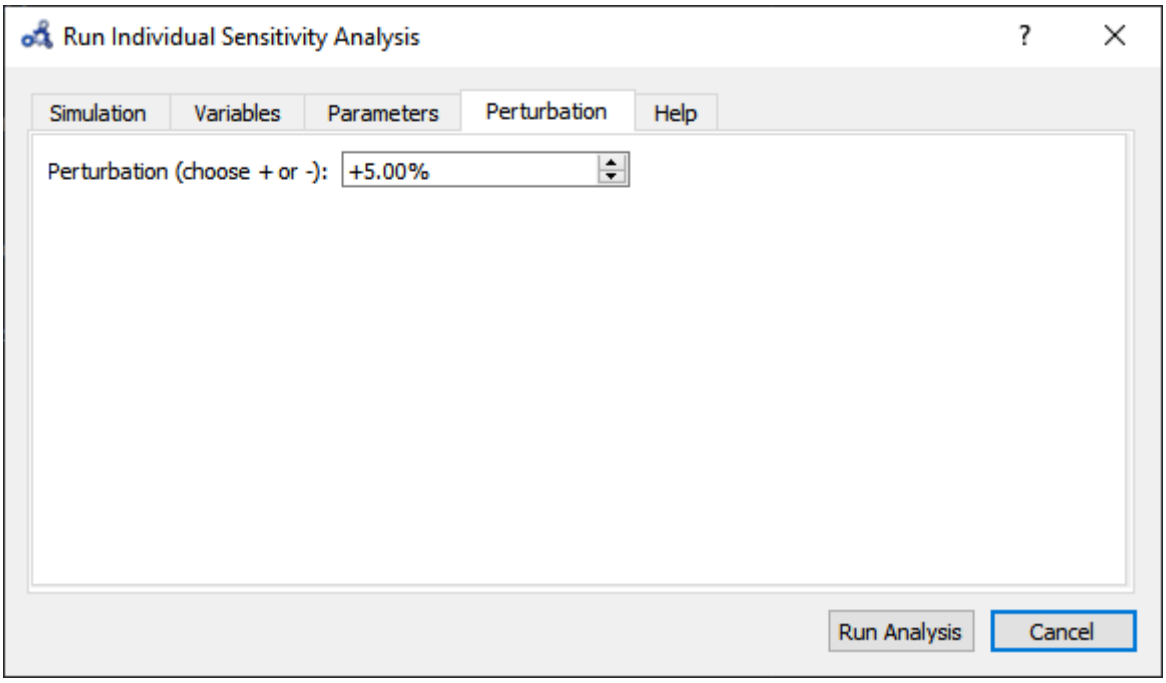


Figure 19.7: Individual sensitivity analysis perturbation.

- After the analysis a dialog with results is shown. Open the heatmap corresponding to the relative sensitivity index.

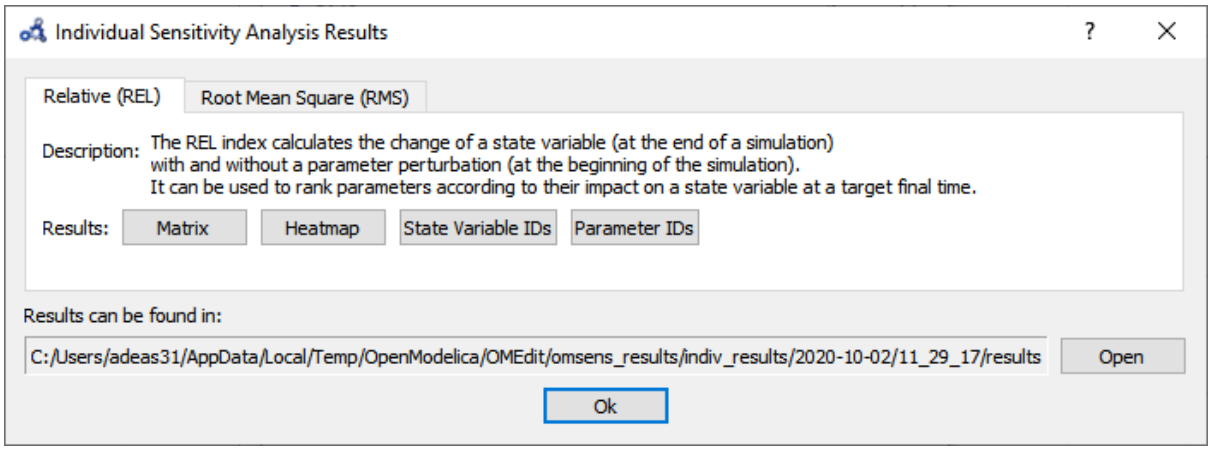


Figure 19.8: Individual sensitivity analysis results.

- The heatmap shows the effect of each parameter on each variable in the form of (parameter,variable) cells. As we can see, pred_pop was affected by the perturbation on every parameter but prey_pop presents a negligible sensitivity to delta (P.3). Recall that this heatmap shows the effect on the variables at time 40 for each perturbation imposed at time 0.

Multi-parameter Sweep

Now we would like to see what happens to pred_pop when the top 3 most influencing parameters are perturbed at the same time. Repeat the first three steps from *Individual Sensitivity Analysis* but this time select **Multi-parameter Sweep**.

- Choose to sweep alpha, gamma and pred_pop_init in a range of $\pm 5\%$ from its default value and with 3 iterations (#iter) distributed equidistantly within that range. Run the sweep analysis.

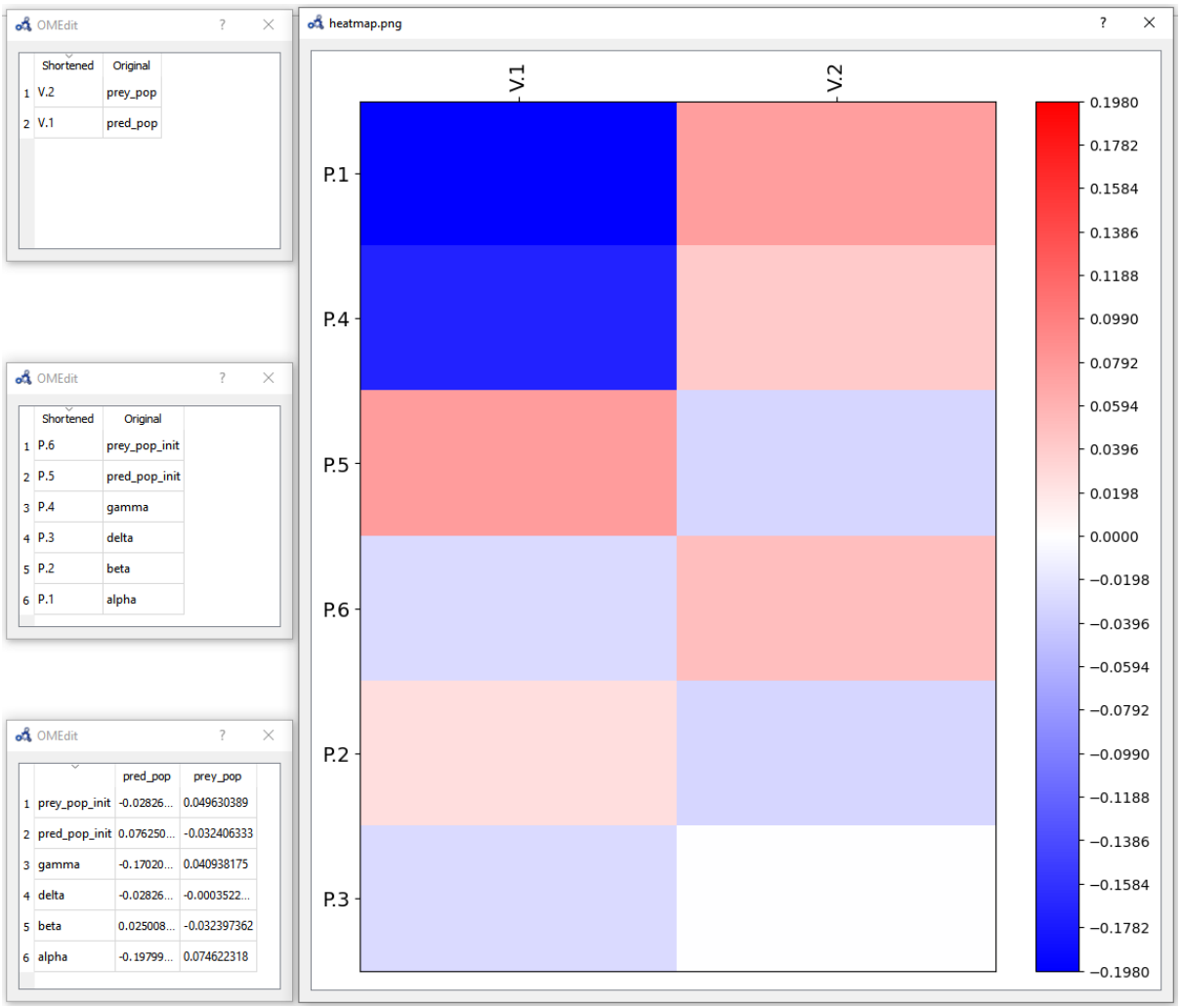


Figure 19.9: Individual sensitivity analysis heatmap.

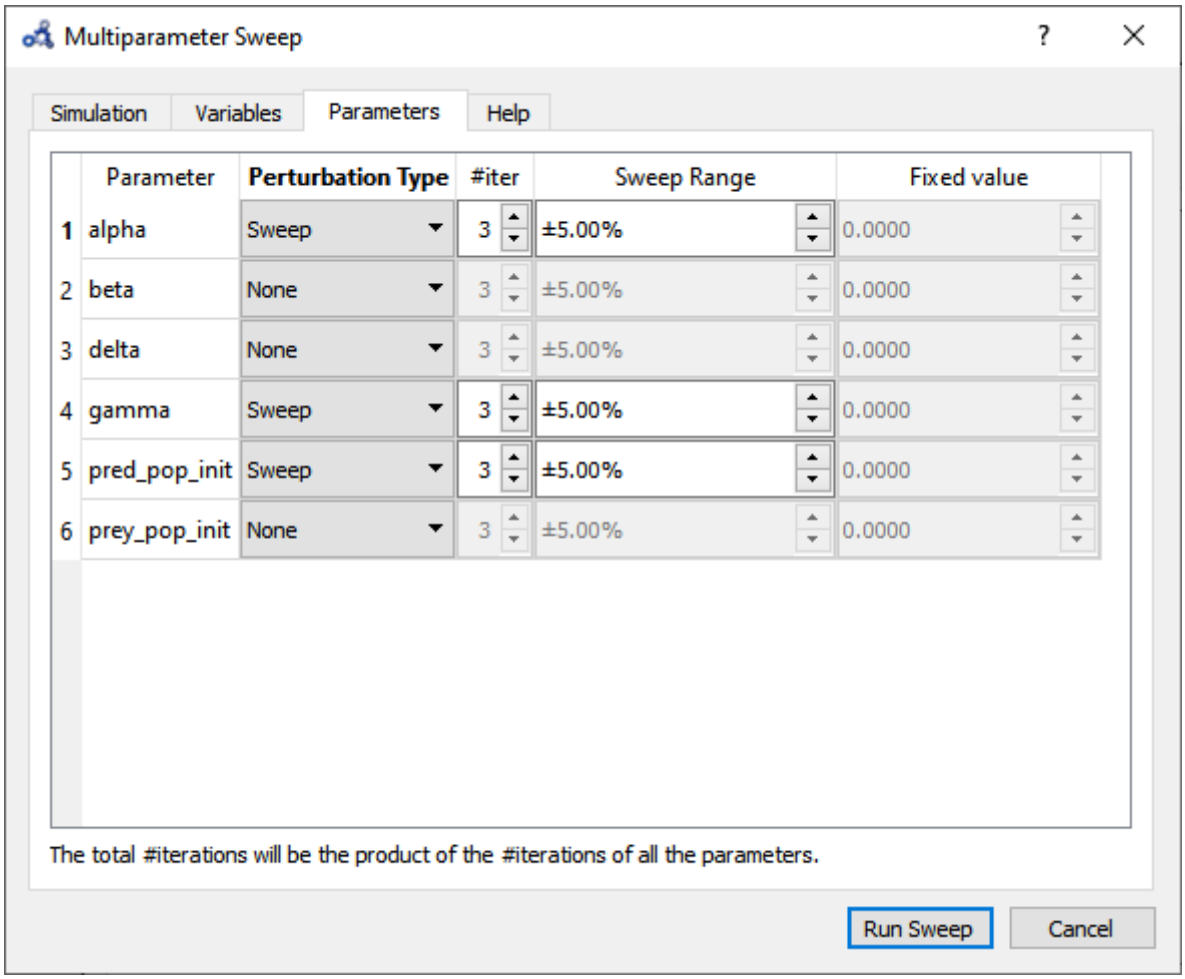


Figure 19.10: Multi-parameter sweep parameters.

- The backend is invoked and when it completes the analysis the following results dialog is shown. Open the plot for pred_pop.

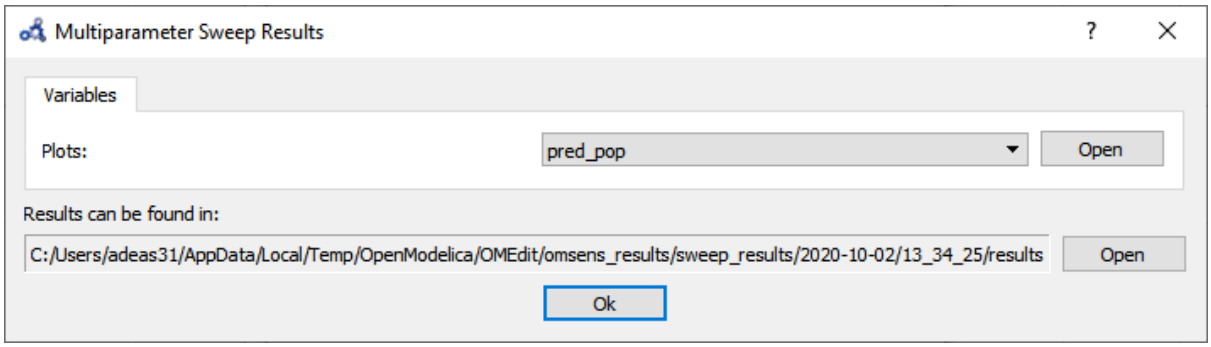


Figure 19.11: Multi-parameter sweep results.

- At time 40 the parameters perturbations with a higher predator population are all blue, but it's not clear which one. We need something more precise.

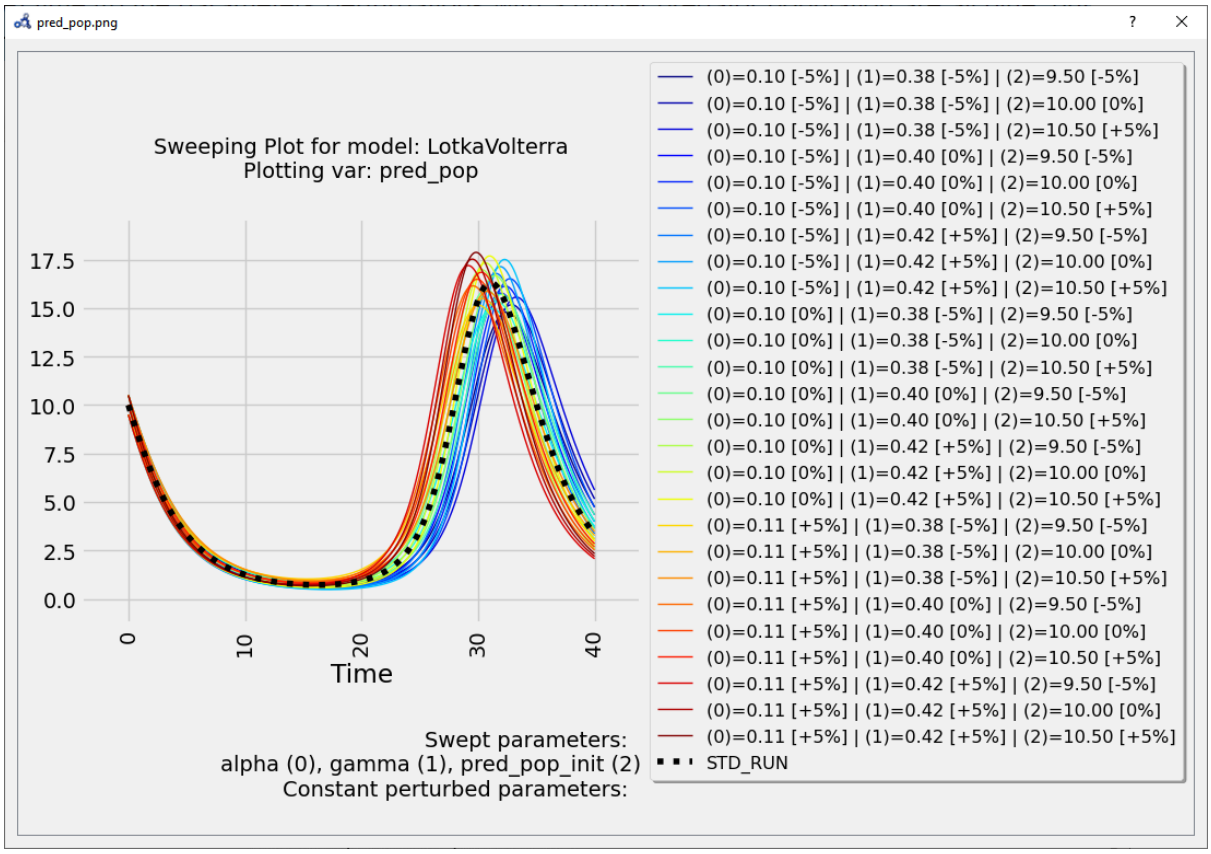


Figure 19.12: Multi-parameter sweep plot.

These results can be very informative but clearly the exhaustive exploration approach doesn't scale for more parameters (#p) and more perturbation values (#v) (#v^#p simulations required).

Vectorial Sensitivity Analysis

Using the Vectorial optimization-based analysis (see below) we can request OMSens to find a combination of parameters that perturbs the most (i.e. minimize or maximize) the value of the target variable at a desired simulation time.

For **Vectorial Sensitivity Analysis** repeat the first two steps from *Individual Sensitivity Analysis* but choose **Vectorial Parameter Based Sensitivity Analysis**.

- Choose only alpha, delta and pred_pop_init to perturb.

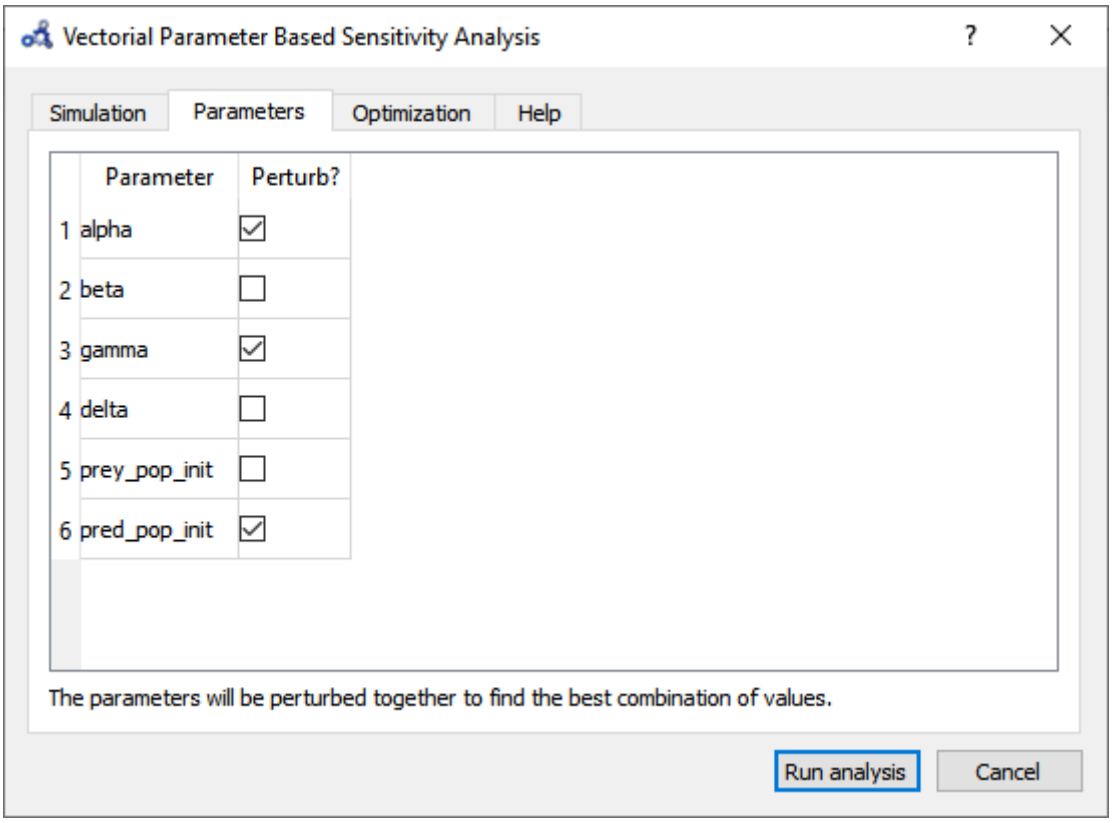


Figure 19.13: Vectorial sensitivity analysis parameters.

- Setup the optimization settings and run the analysis.
- The **Parameters** tab in the results window shows the values found by the optimization routine that maximize pred_pop at t=40 s.
- The **State Variable** tab shows the comparison between the values of the variable in the standard run vs the perturbed run at simulation time 40s.
- If we simulate using the optimum values and compare it to the standard (unperturbed) run, we see that it **delays the bell** described by the variable.
- So far, we have only perturbed the top 3 parameters detected by the **Individual Sensitivity** method. Maybe we can find a greater effect on the variable if we perturb all 6 parameters. Running a Sweep is not an option as perturbing 6 parameters with 3 iterations each results in 3^6=729 simulations. We run another Vectorial Sensitivity Analysis instead but now choose to perturb all 6 parameters.
- The **parameters tab** shows that the optimum value is found by perturbing all of the parameters to their boundaries.
- The **State Variable** tab shows that pred_pop can be increased by 98% when perturbing the 6 parameters as opposed to 68% when perturbing the top 3 influencing parameters.
- The plot shows again that the parameters found delay the bell-shaped curve, but with a stronger impact than before.

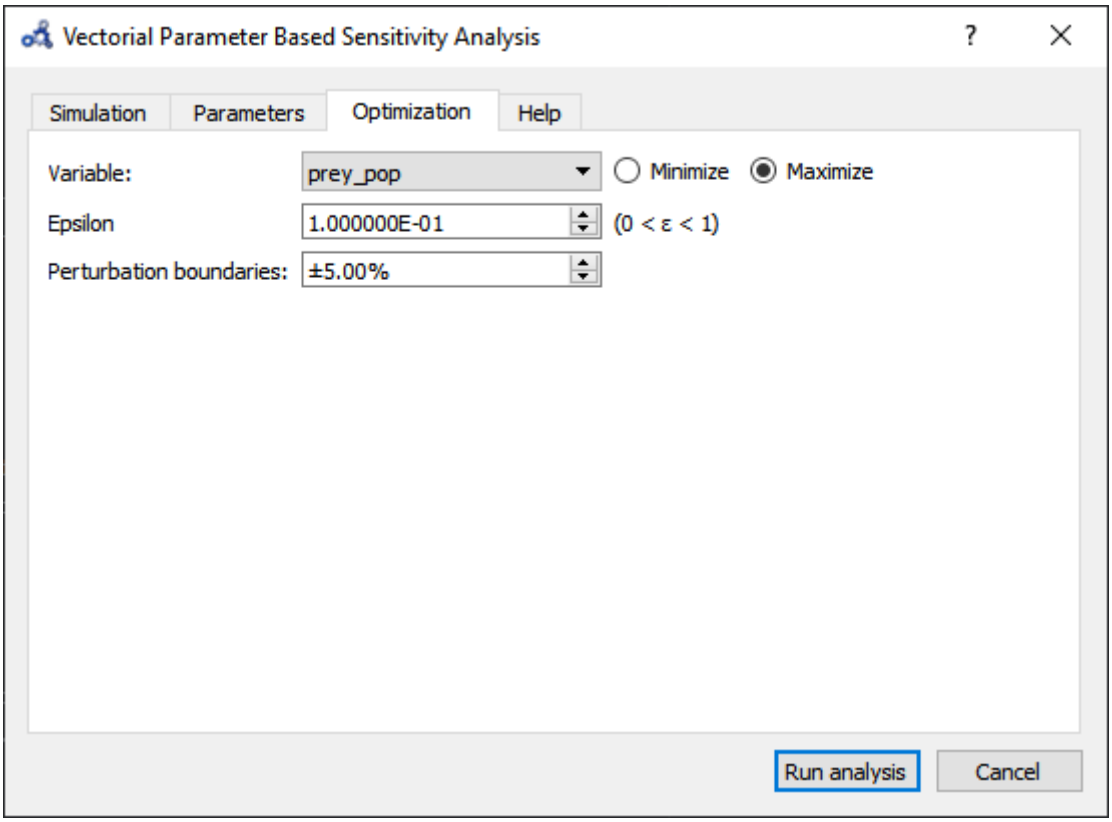


Figure 19.14: Vectorial sensitivity analysis optimization.

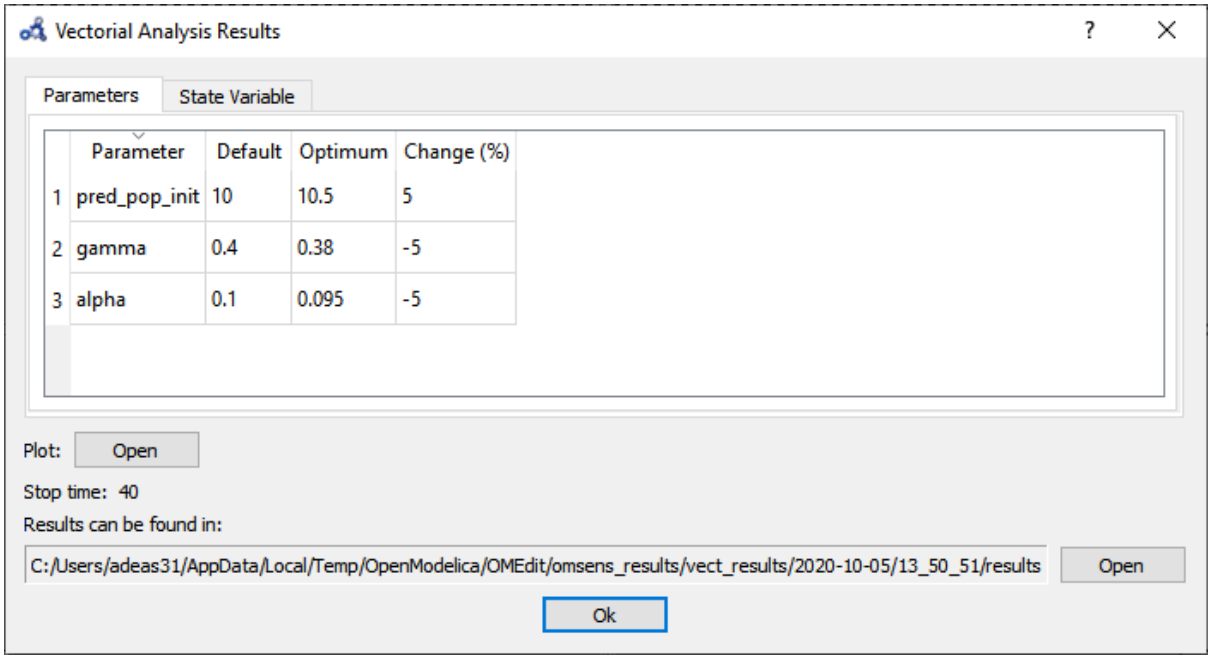


Figure 19.15: Vectorial sensitivity analysis parameters result.

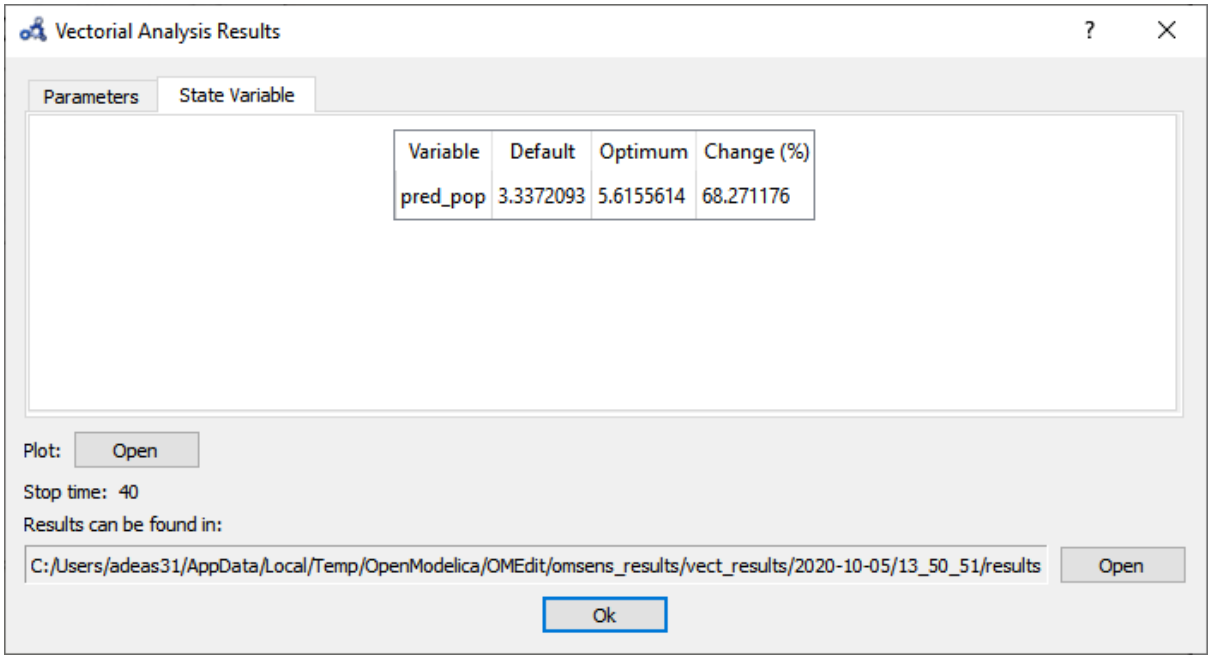


Figure 19.16: Vectorial sensitivity analysis state variables.

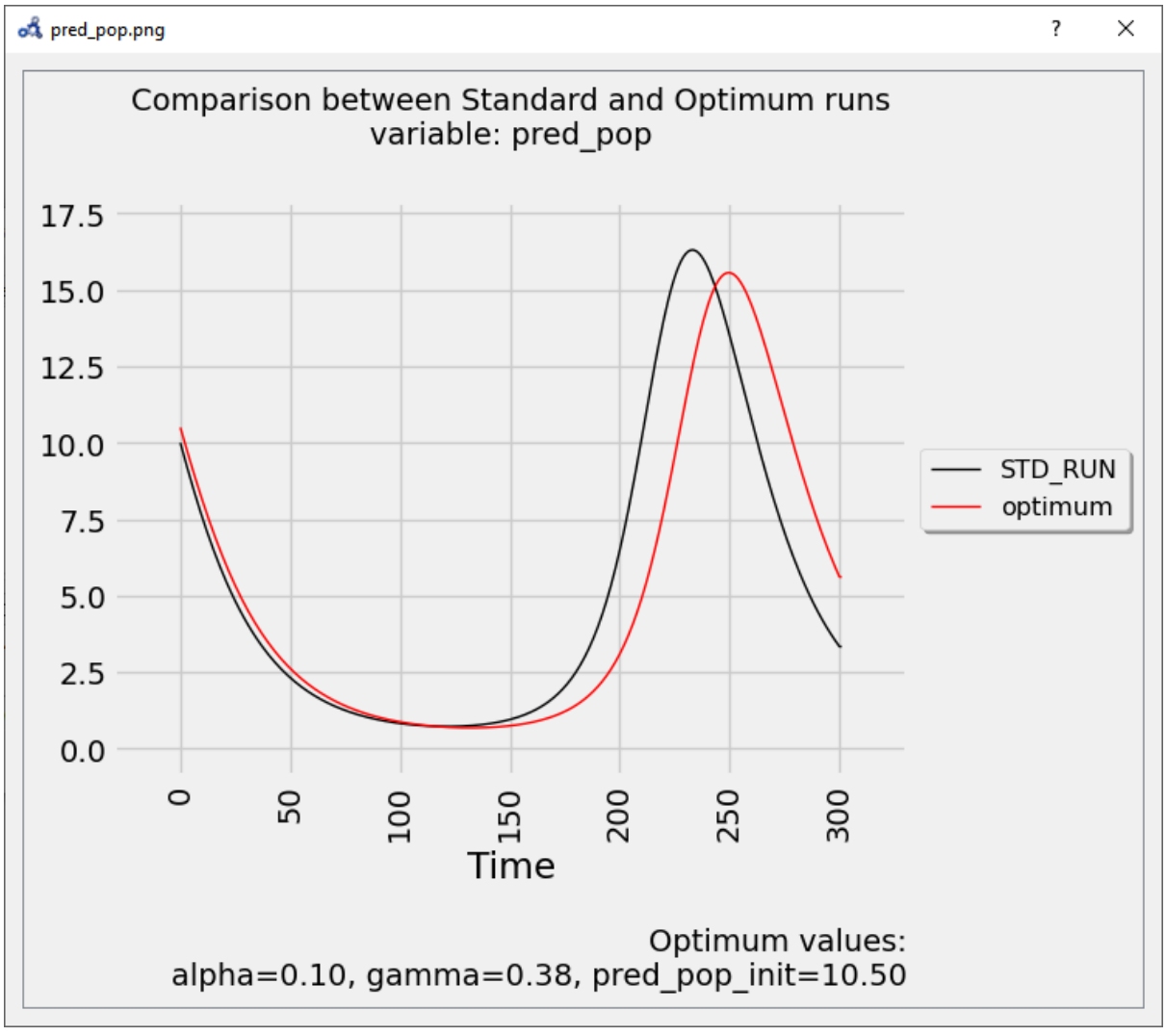


Figure 19.17: Vectorial sensitivity analysis plot.

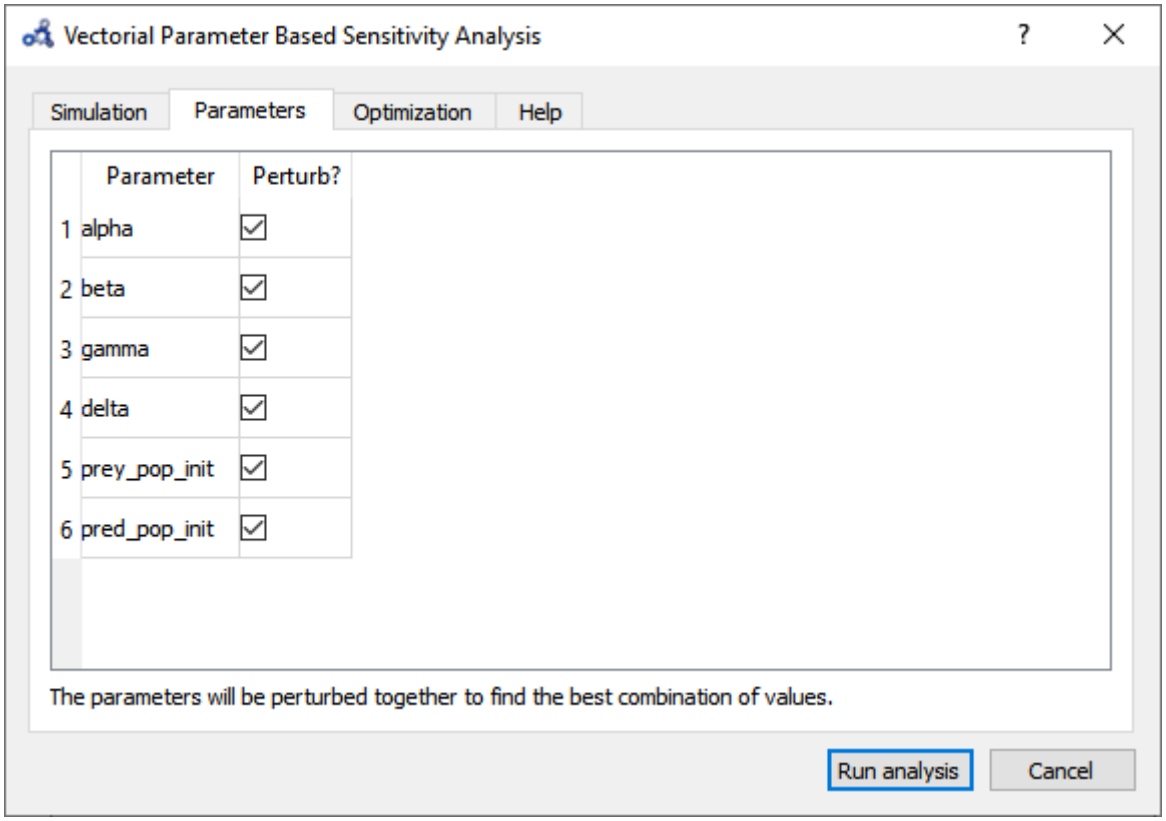


Figure 19.18: Vectorial sensitivity analysis parameters.

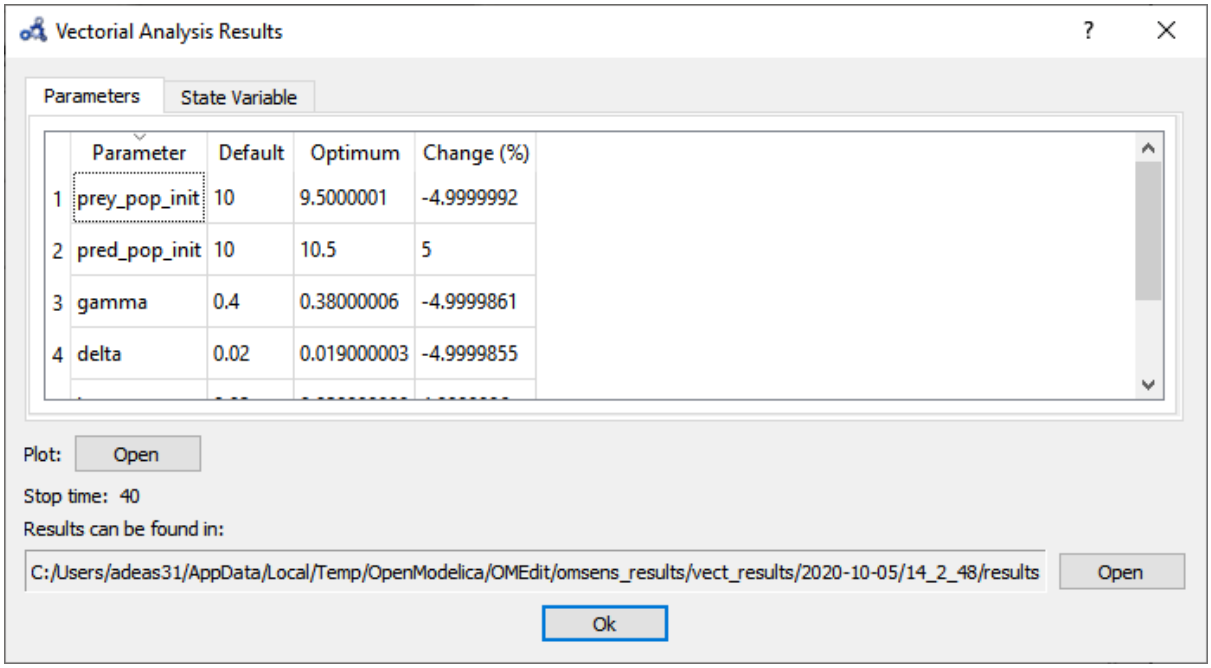


Figure 19.19: Vectorial sensitivity analysis parameters result.

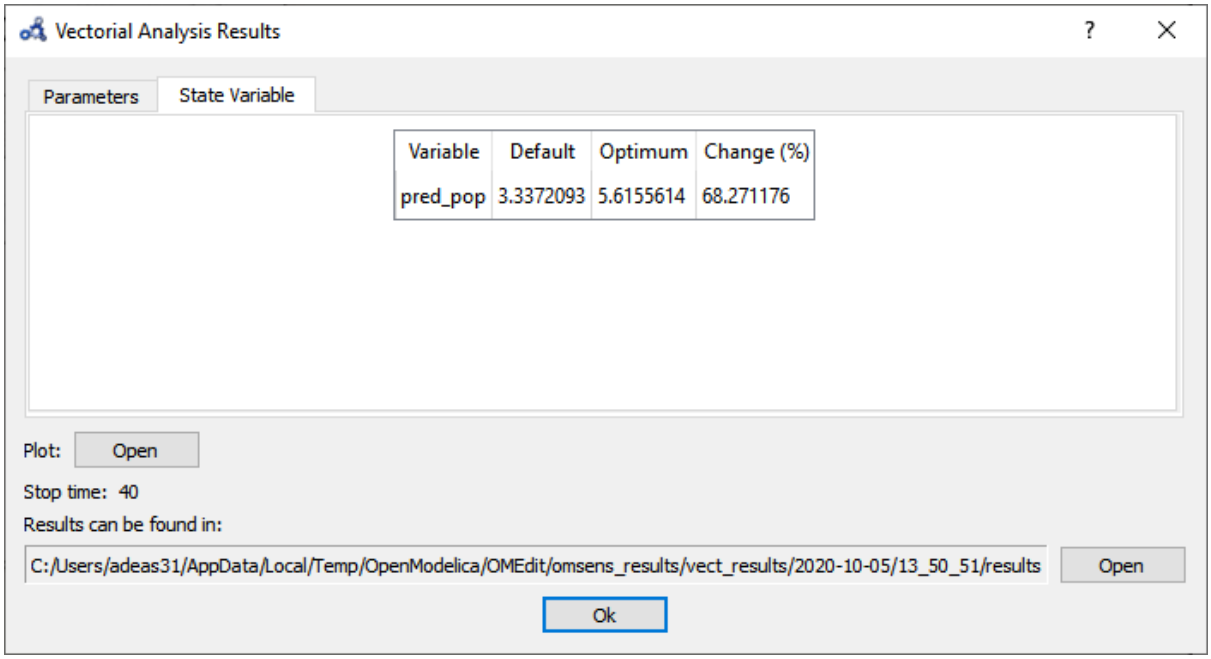


Figure 19.20: Vectorial sensitivity analysis state variables.

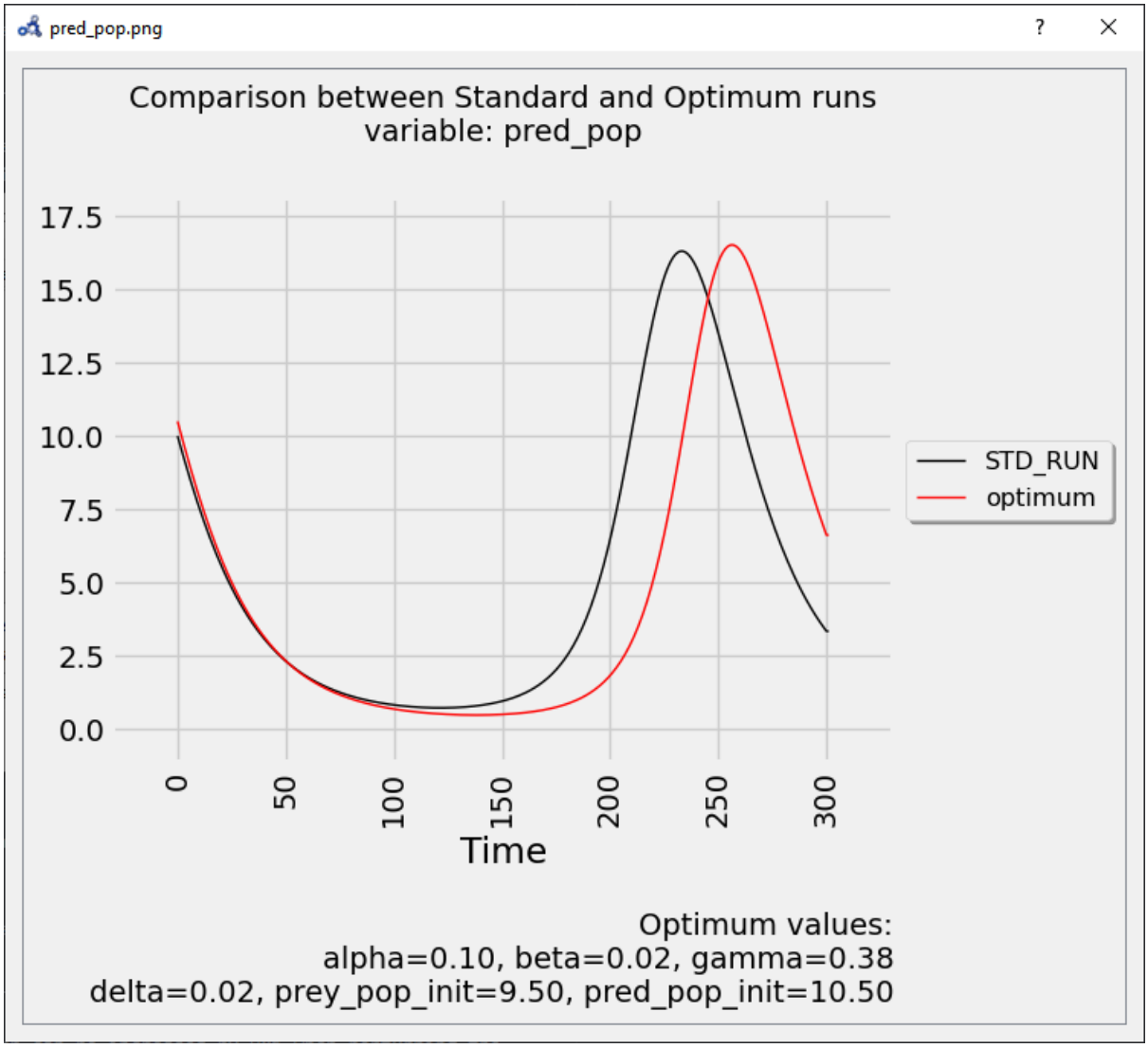


Figure 19.21: Vectorial sensitivity analysis plot.

CHAPTER

TWENTY

PDEMODELICA1

PDEModelica1 is nonstandardised experimental Modelica language extension for 1-dimensional partial differential extensions (PDE).

It is enabled using compiler flag `--grammar=PDEModelica`. Compiler flags may be set e.g. in OMEdit (globally in Tools->Options->Simulation->Translation Flags or in Simulation Setup->Translation Flags for specific models) or in an OpenModelica script using `setCommandLineOptions`. Note that PDEModelica does now work yet with the current frontend so you need to also use the flag `-d=-newInst` or check "Enable old frontend for code generation" in OMEdit under Translation Flags.

```
>>> setCommandLineOptions("--grammar=PDEModelica -d=-newInst")
true
```

20.1 PDEModelica1 language elements

Let us introduce new PDEModelica1 language elements by an advection equation example model:

```
model Advection "advection equation"
  parameter Real pi = Modelica.Constants.pi;
  parameter DomainLineSegment1D omega(L = 1, N = 100)  "domain";
  field Real u(domain = omega)                         "field";
initial equation
  u = sin(2*pi*omega.x)                                "IC";
equation
  der(u) + pder(u,x) = 0   indomain omega              "PDE";
  u = 0                    indomain omega.left         "BC";
  u = extrapolateField(u)  indomain omega.right        "extrapolation";
end Advection;
```

The domain `omega` represents the geometrical domain where the PDE holds. The domain is defined using the built-in record `DomainLineSegment1D`. This record contains among others `L` - the length of the domain, `N` - the number of grid points, `x` - the coordinate variable and the regions `left`, `right` and `interior`, representing the left and right boundaries and the interior of the domain.

The field variable `u` is defined using a new keyword `field`. The `domain` is a mandatory attribute to specify the domain of the field.

The `indomain` operator specifies where the equation containing the field variable holds. It is utilised in the initial conditions (IC) of the fields, in the PDE and in the boundary conditions (BC). The syntax is

```
anEquation indomain aDomain.aRegion;
```

If the region is omitted, `interior` is the default (e.g. the PDE in the example above).

The IC of the field variable u is written using an expression containing the coordinate variable `omega.x`.

The PDE contains a partial space derivative written using the `pder` operator. Also the second derivative is allowed (not in this example), the syntax is e.g. `pder(u,x,x)`. It is not necessary to specify the domain of coordinate in pder (to write e.g. `pder(u,omega.x)`, even though `x` is a member of `omega`.

20.2 Limitations

BCs may be written only in terms of variables that are spatially differentiated currently.

All fields that are spatially differentiated must have either BC or extrapolation at each boundary. This extrapolation should be done automatically by the compiler, but this has not been implemented yet. The current workaround is the usage of the `extrapolateField()` operator directly in the model.

If-equations are not spported yet, if-expressions must be used instead.

20.3 Viewing results

During translation field variables are replaced with arrays. These arrays may be plotted using *Array Plot* or even better using *Array Parametric Plot* (to plot x-coordinate versus a field).

CHAPTER

TWENTYONE

MDT - THE OPENMODELICA DEVELOPMENT TOOLING ECLIPSE PLUGIN

21.1 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.

21.2 Installation

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

The latest installation instructions are available through the OpenModelica Trac.

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

21.3 Getting Started

21.3.1 Configuring the OpenModelica Compiler

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

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

21.3.2 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..

21.3.3 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 21.1.

21.3.4 Creating one or more Modelica Projects

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

You need to disable automatic build for the project(s) (Figure 21.3).

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

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

21.3.5 Building and Running a Project

After having created a project, you eventually need to build the project (Figure 21.4).

The build options are the same as the make targets: you can build, build from scratch (clean), or run simulations depending on how the project is setup. See Figure 21.5 for an example of how omc can be compiled (`make omc` builds OMC).

21.3.6 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 21.7.

21.3.7 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.

21.3.8 Creating a Class

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

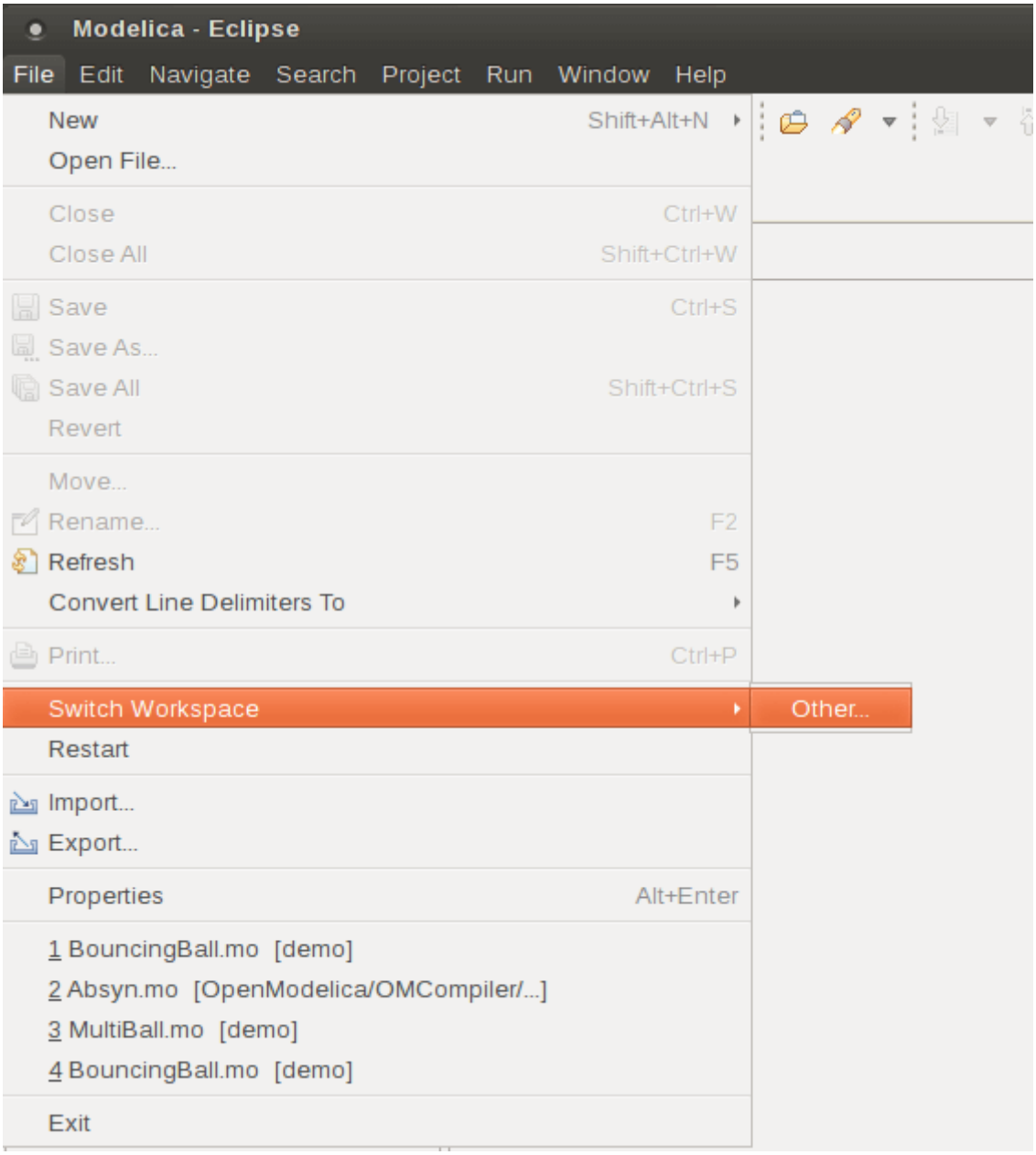


Figure 21.1: Eclipse Setup - Switching Workspace.

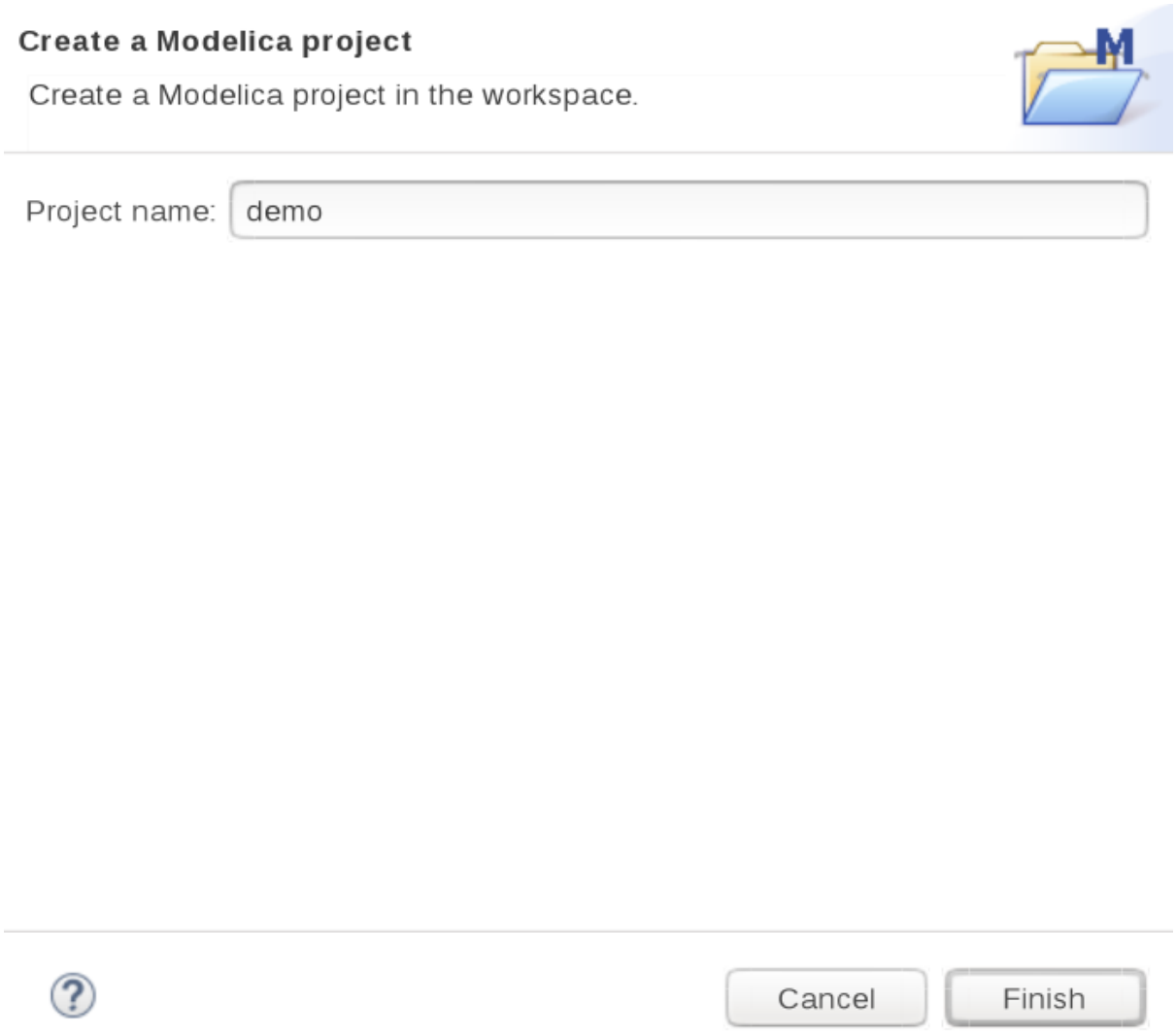


Figure 21.2: Eclipse Setup - creating a Modelica project in the workspace.

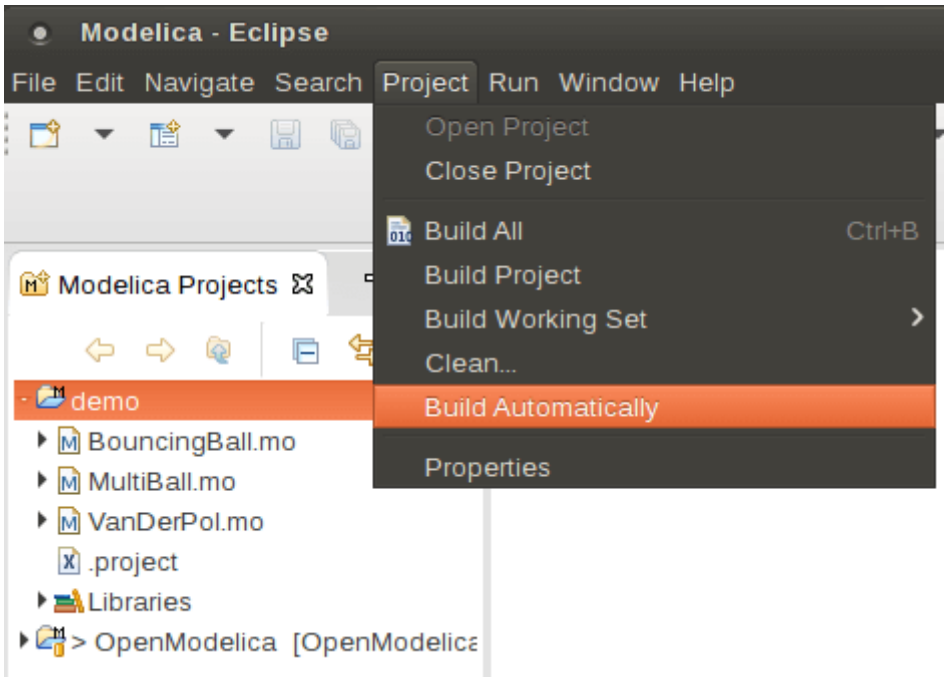


Figure 21.3: Eclipse Setup - disable automatic build for the projects.

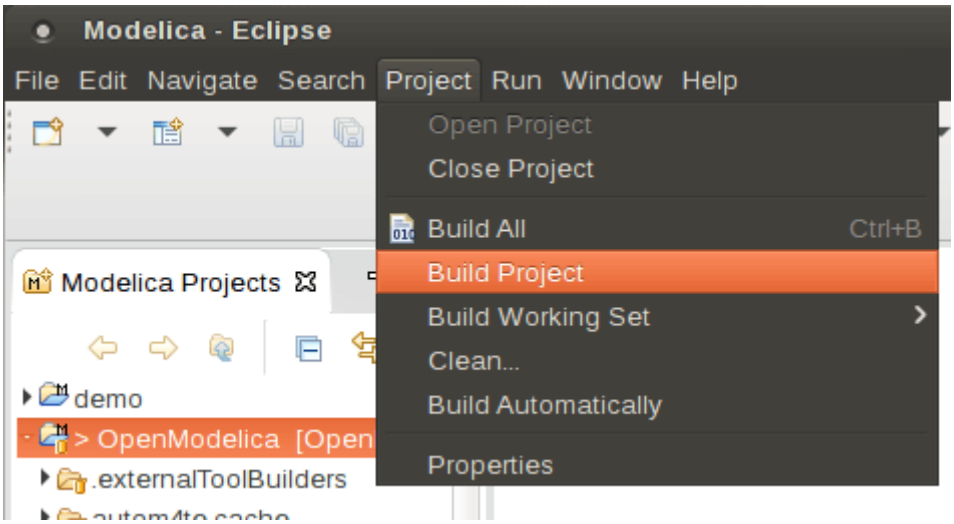


Figure 21.4: Eclipse MDT - Building a project.

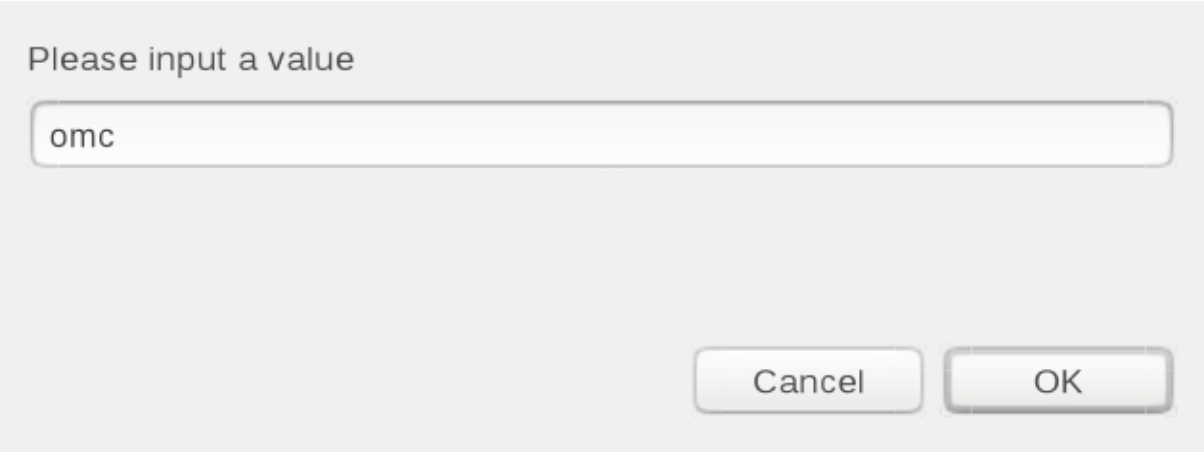


Figure 21.5: Eclipse - building a project.

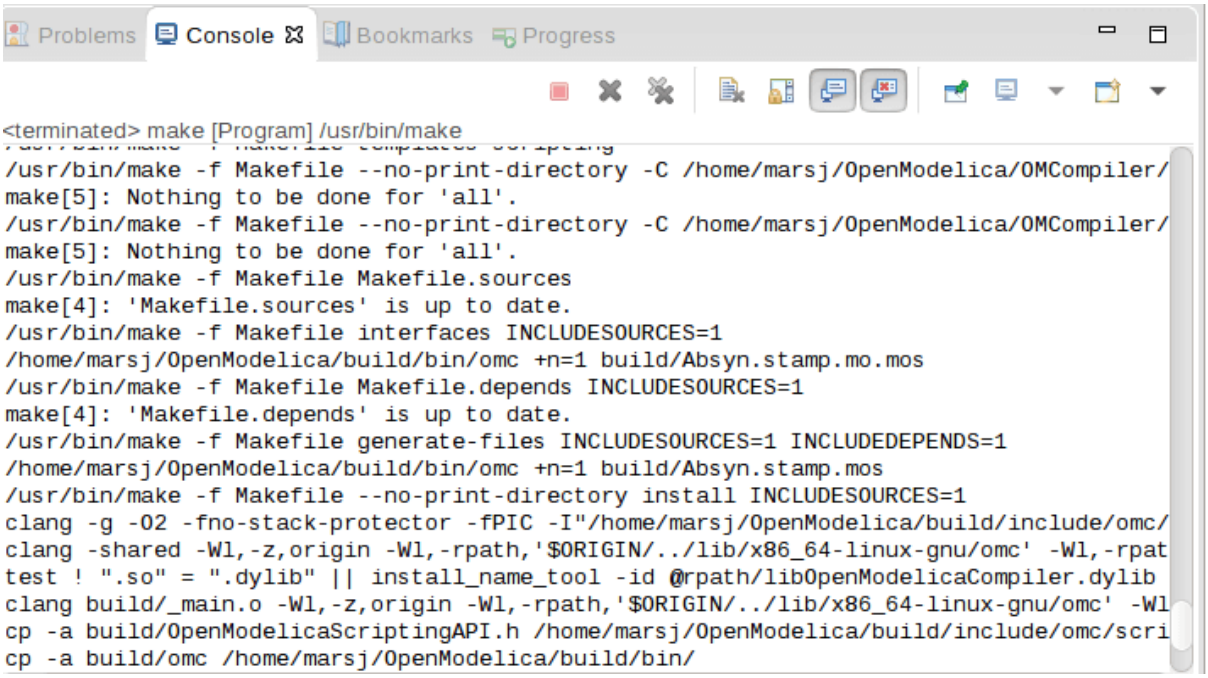


Figure 21.6: Eclipse - building a project, resulting log.

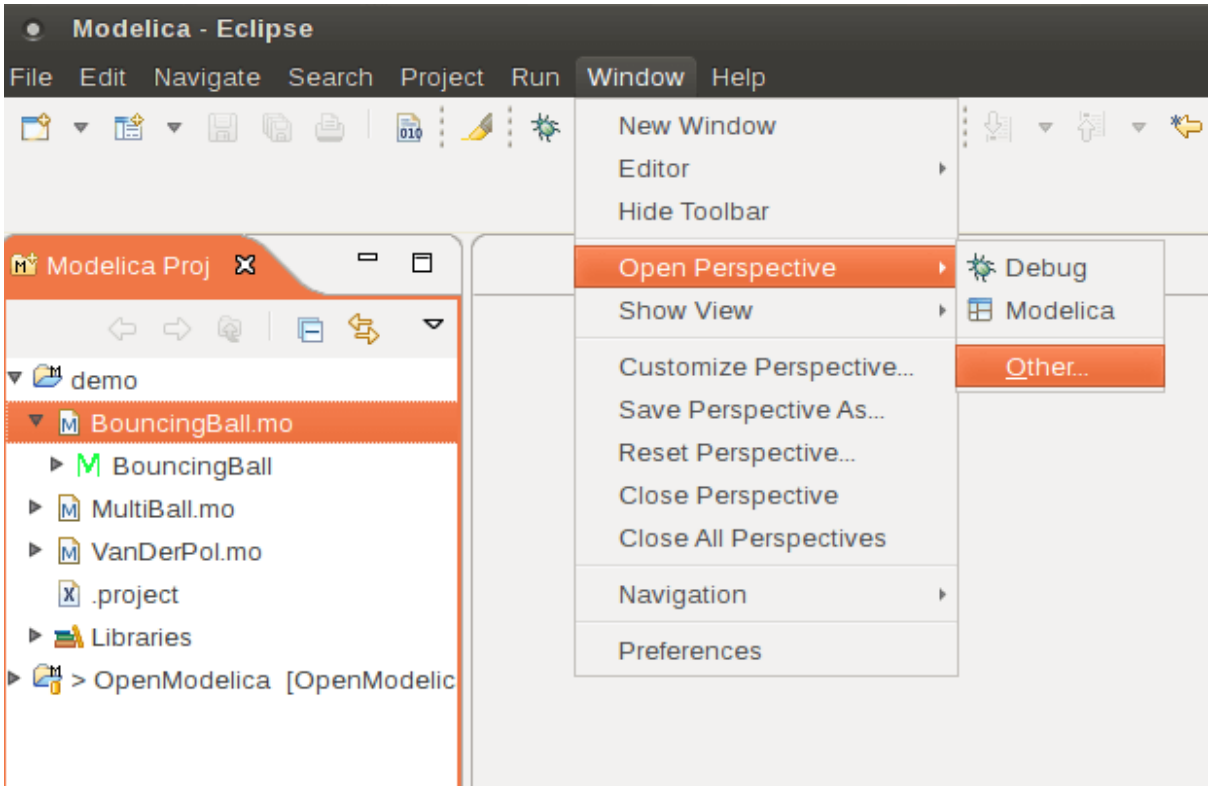


Figure 21.7: Eclipse - Switching to another perspective - e.g. the Java Perspective.

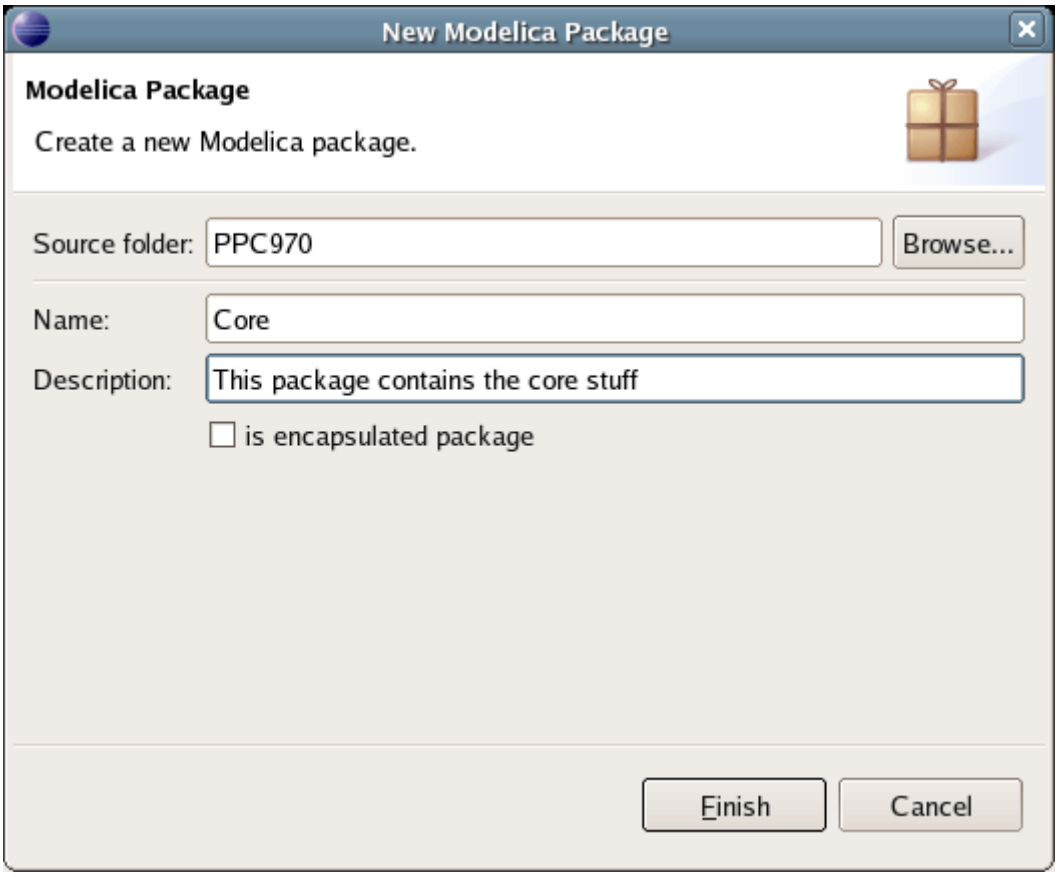


Figure 21.8: Creating a new Modelica package.

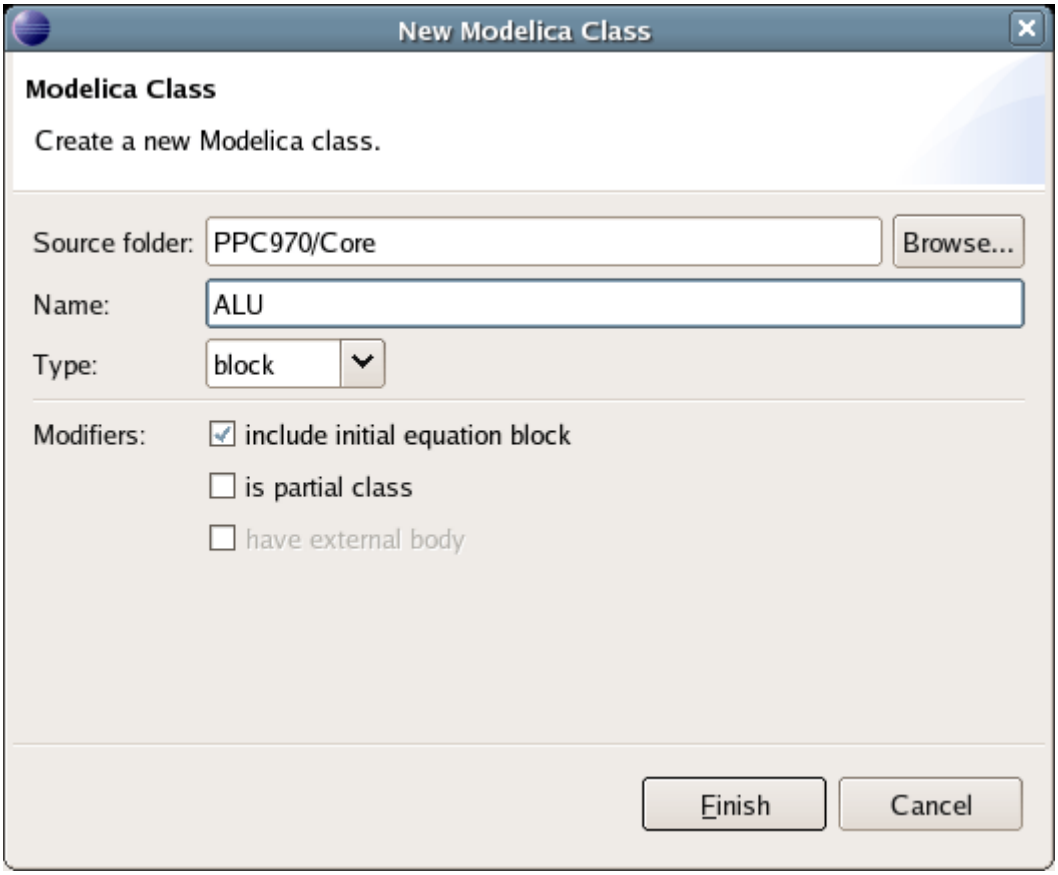


Figure 21.9: Creating a new Modelica class.

21.3.9 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.

21.3.10 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.

21.3.11 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.

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

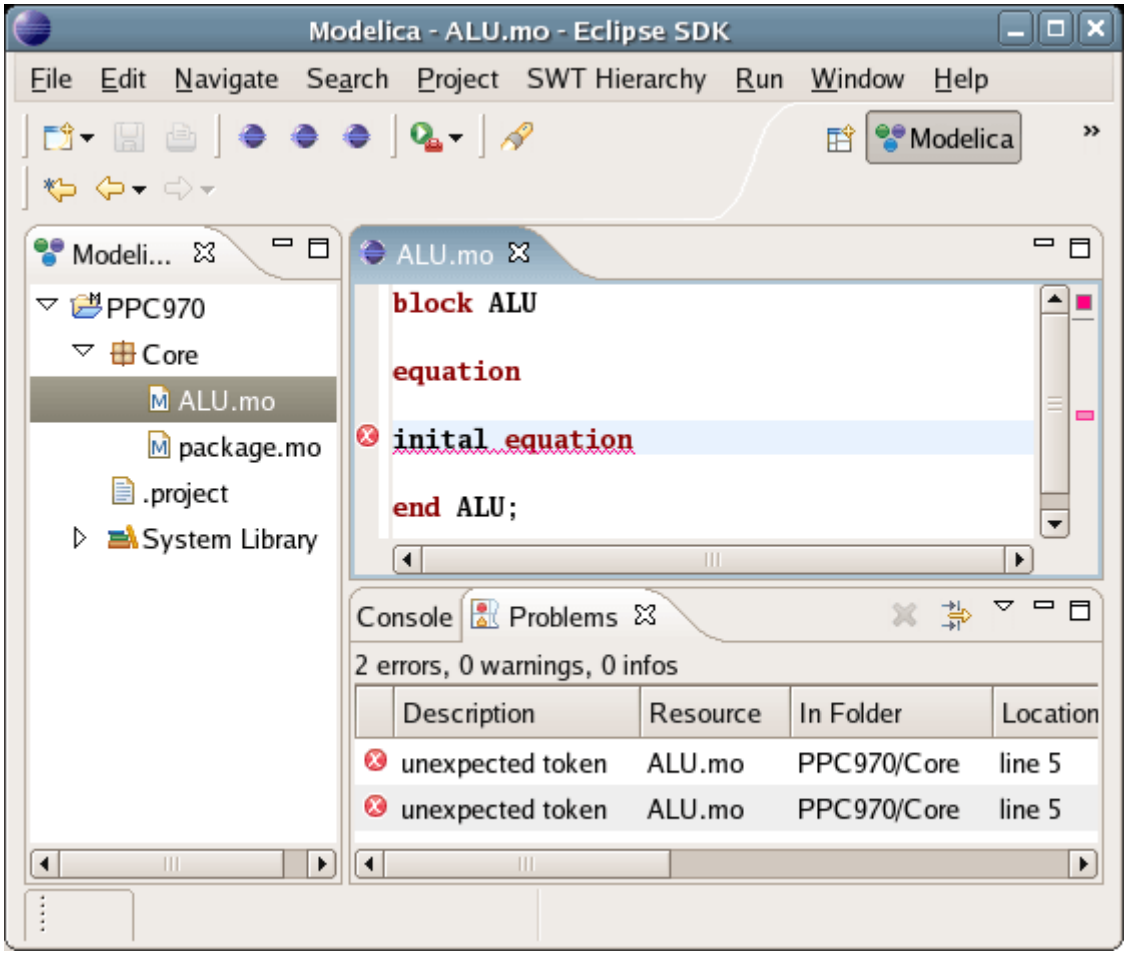


Figure 21.10: Syntax checking.

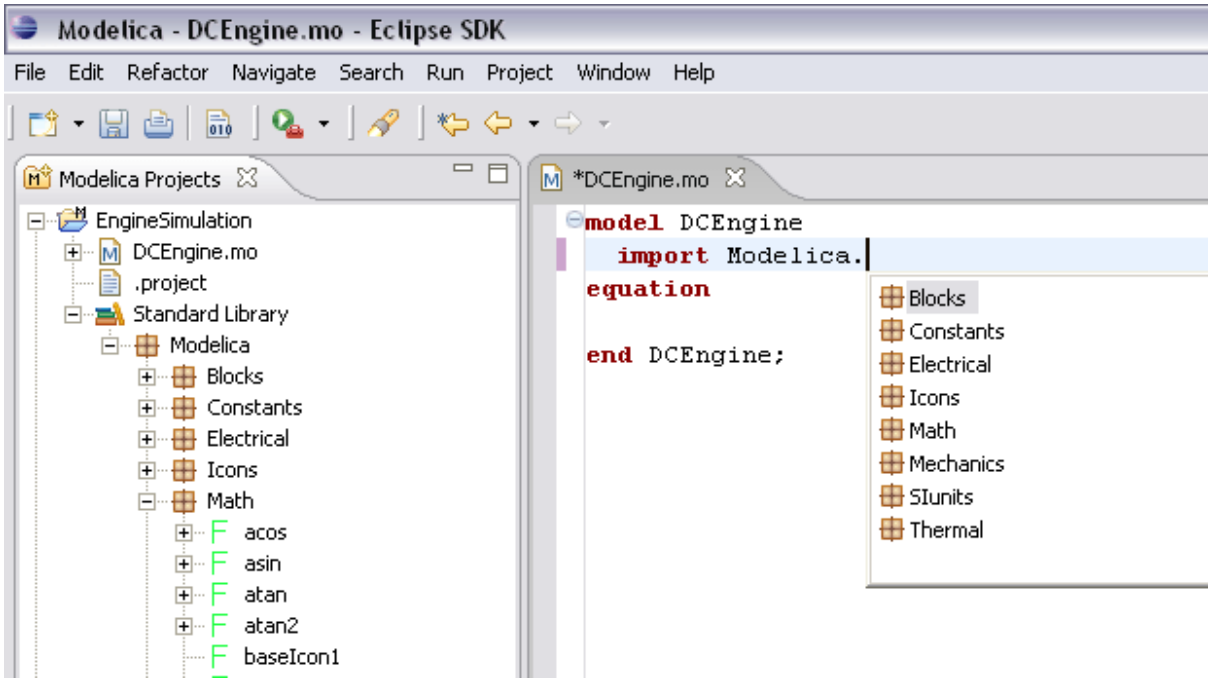


Figure 21.11: Code completion when typing a dot.

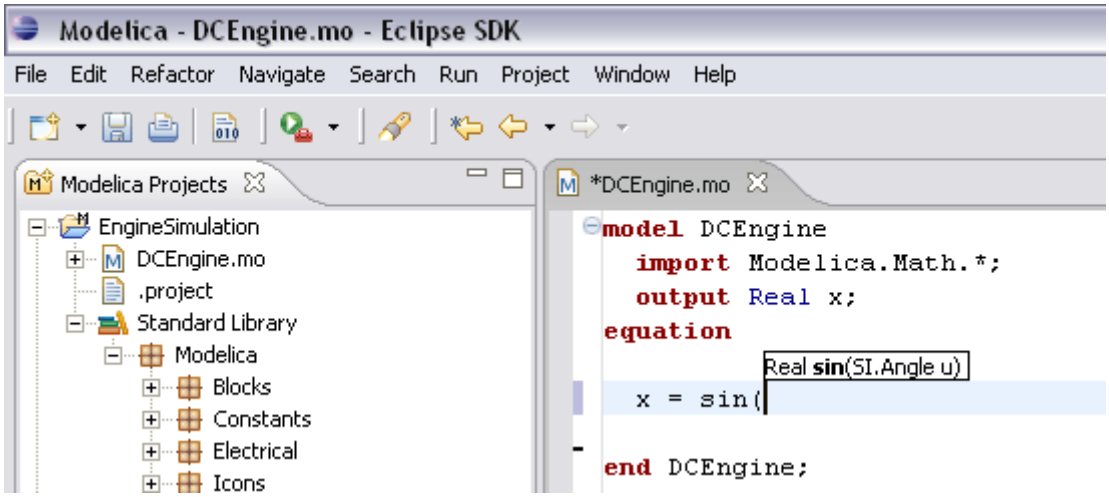


Figure 21.12: Code completion at a function call when typing left parenthesis.

21.3.12 Code Assistance on Identifiers when Hovering

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

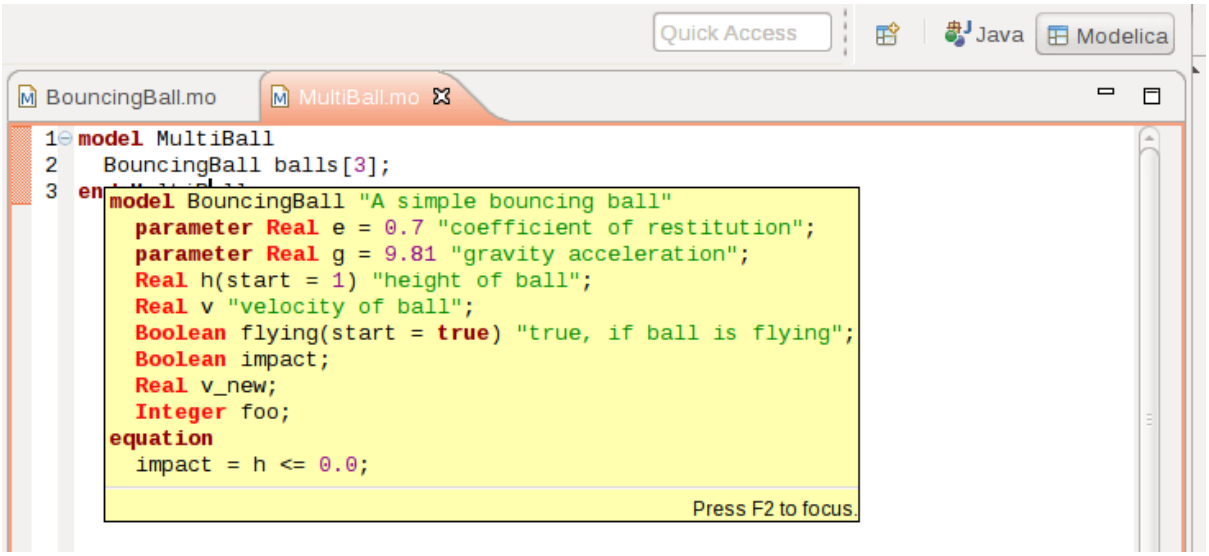


Figure 21.13: Displaying information for identifiers on hovering.

21.3.13 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.

21.3.14 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.

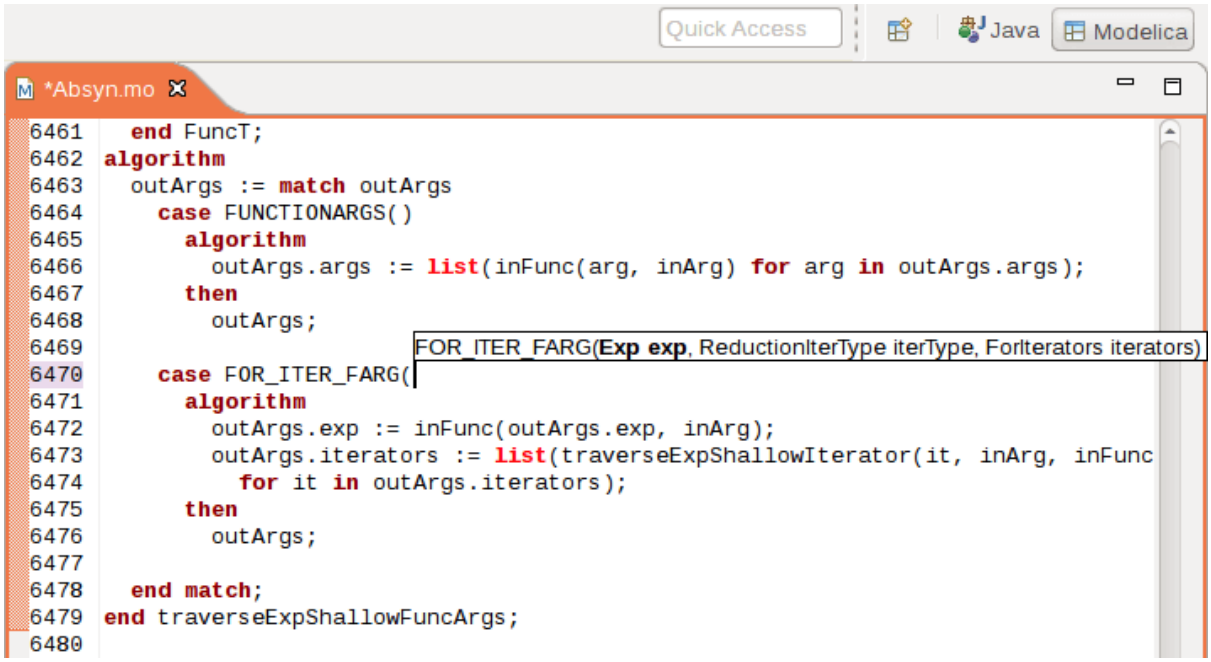


Figure 21.14: Code assistance when writing cases with records in MetaModelica.

21.3.15 Using the MDT Console for Plotting

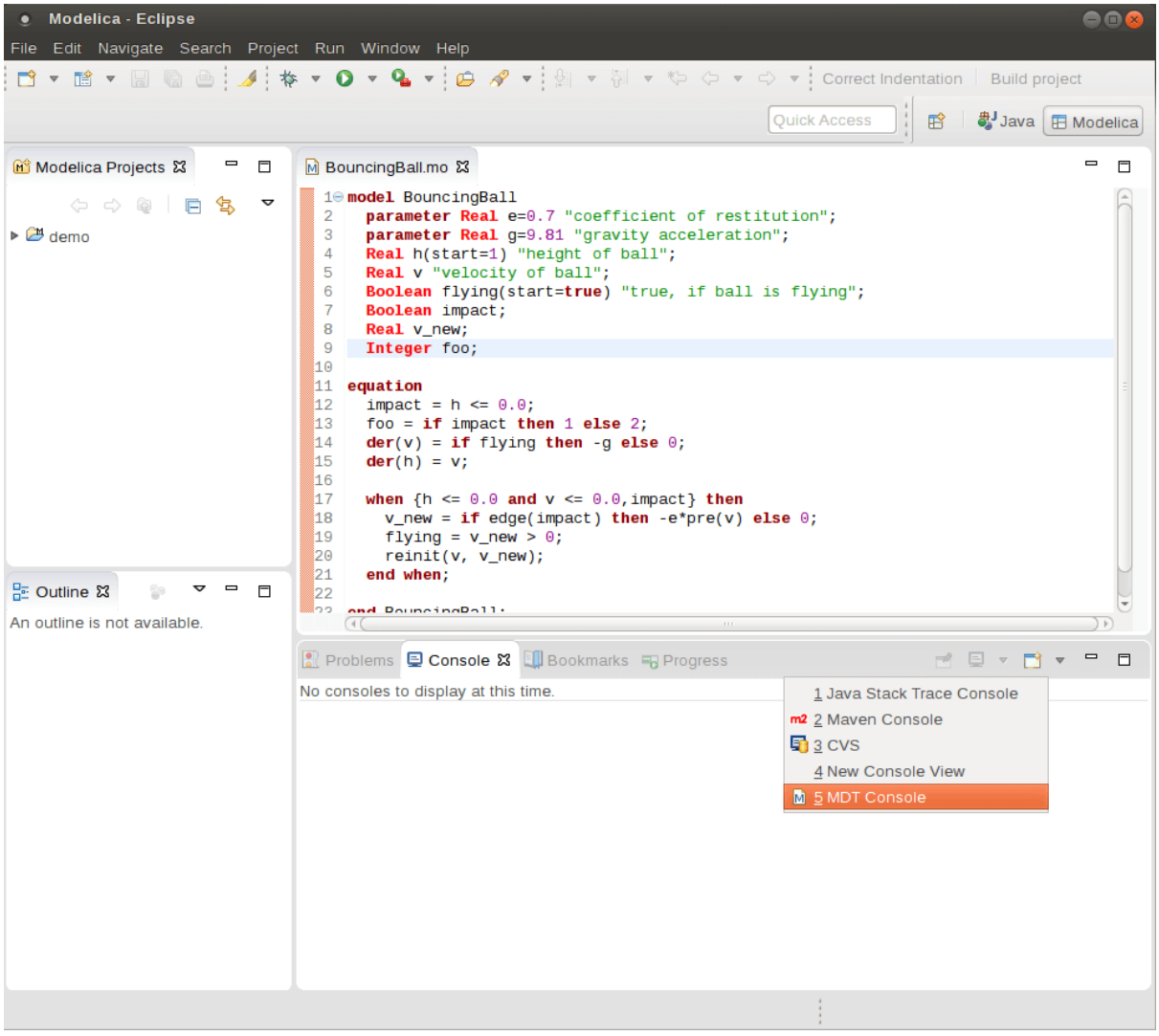


Figure 21.15: Activate the MDT Console.

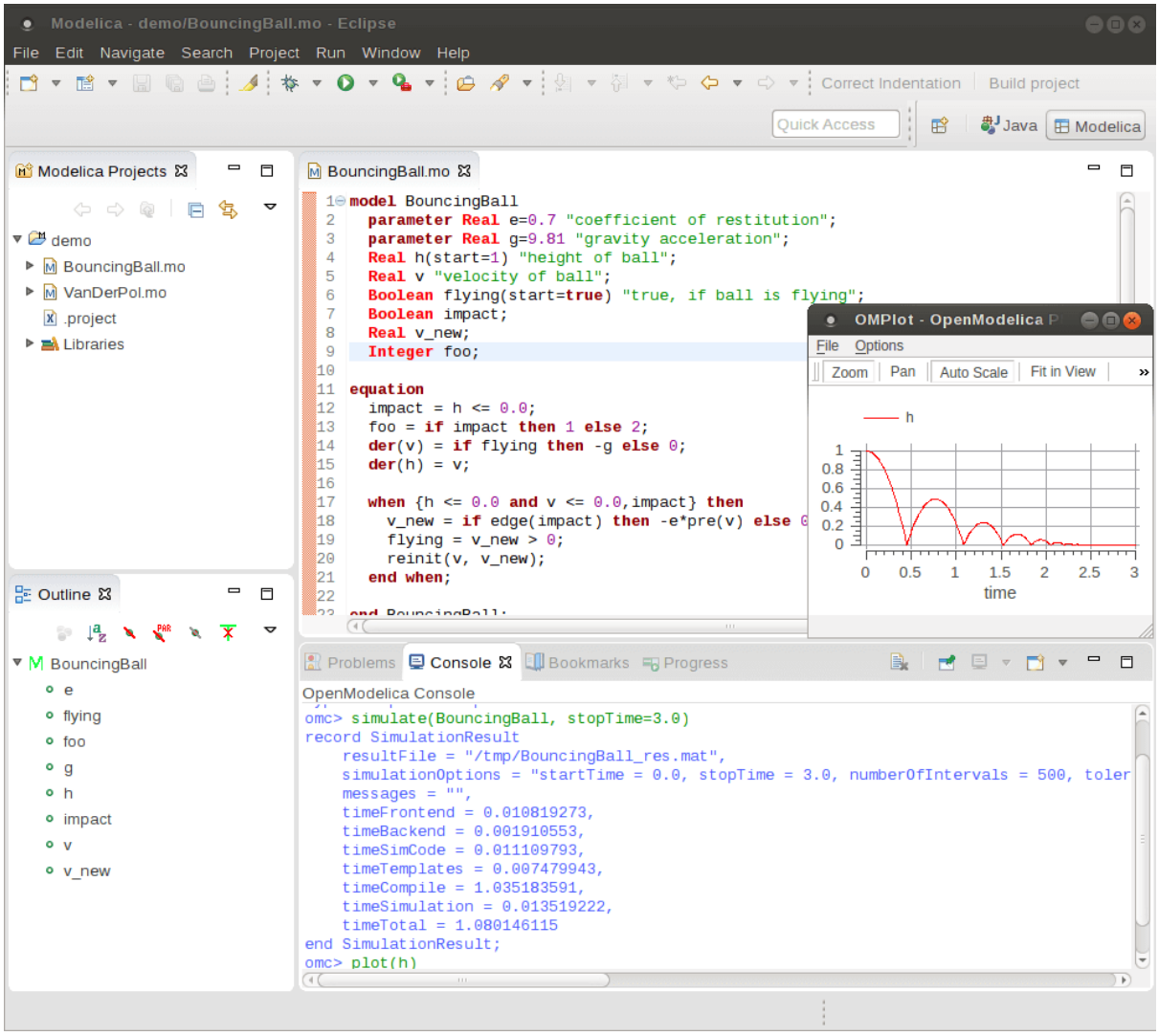


Figure 21.16: Simulation from MDT Console.

CHAPTER

TWENTYTWO

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

22.1 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.

22.1.1 Starting the Modelica Debugging Perspective

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

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

All these steps are presented below using images.

Create mos file

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

```
function HelloWorld
  input Real r;
  output Real o;
algorithm
```

(continues on next page)

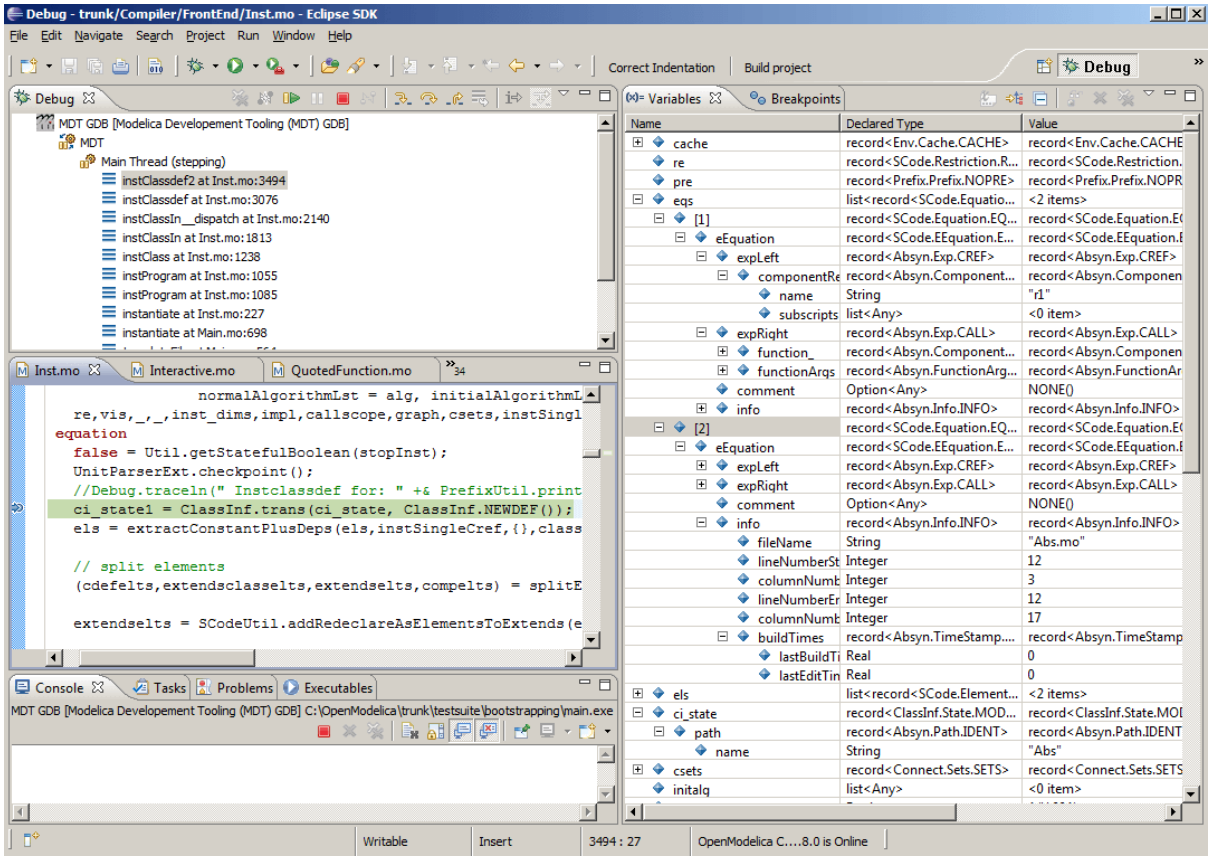


Figure 22.1: Debugging functionality.

(continued from previous page)

```
  o := 2 * r;
end HelloWorld;
```

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

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

Setting the debug configuration

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

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

Note that we require Gnu Debugger (GDB) for debugging session. We must specify the GDB location, also we must pass our script file as an argument to OMC.

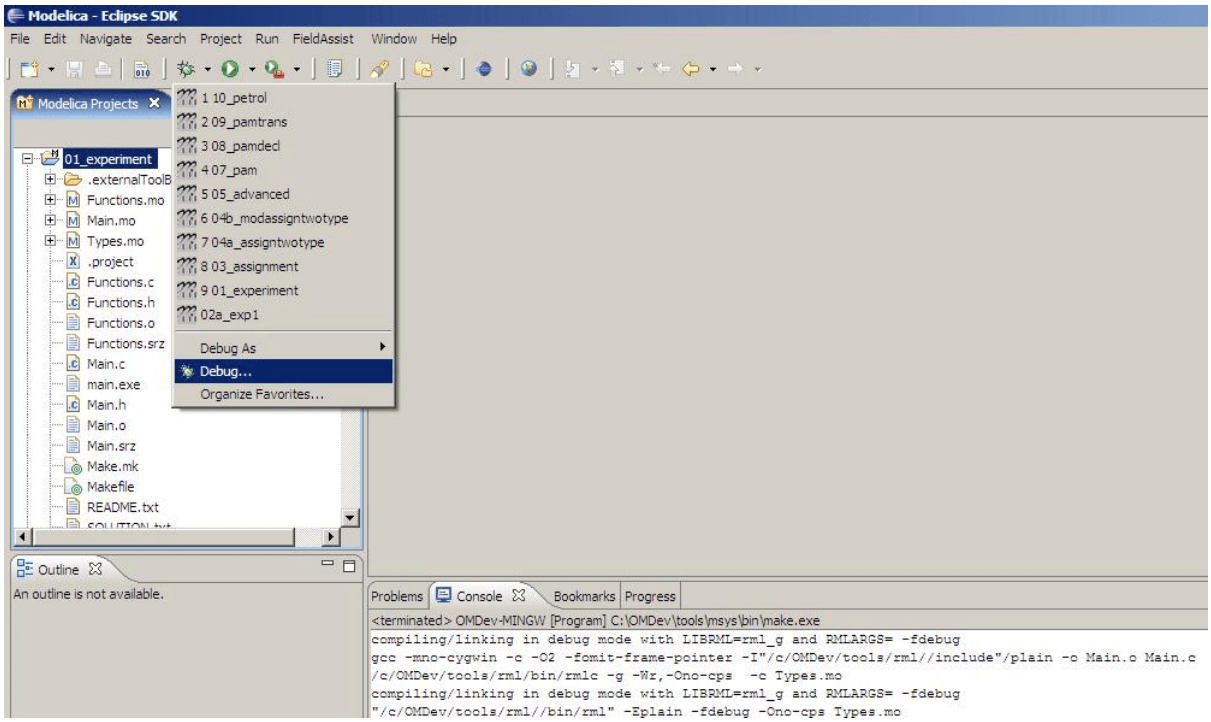


Figure 22.2: Accessing the debug configuration dialog.

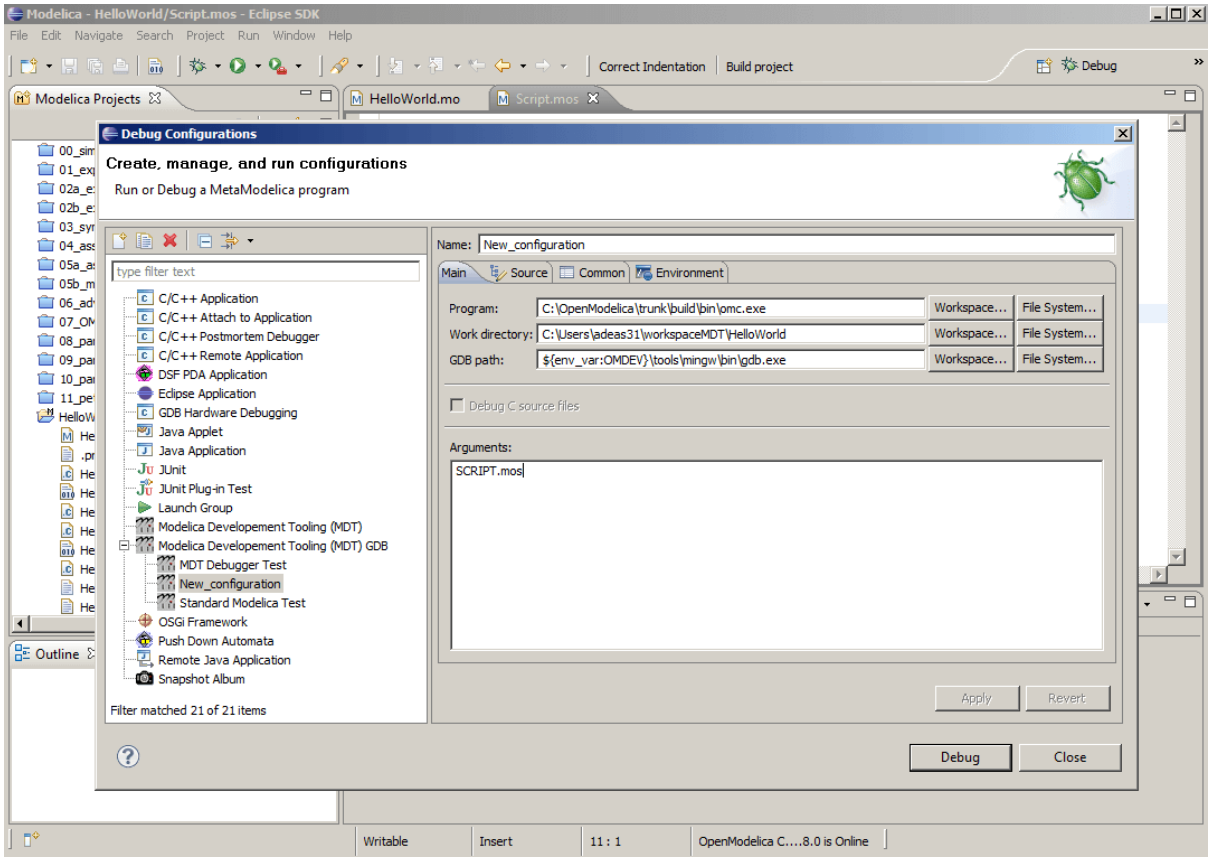


Figure 22.3: Creating the Debug Configuration.

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.

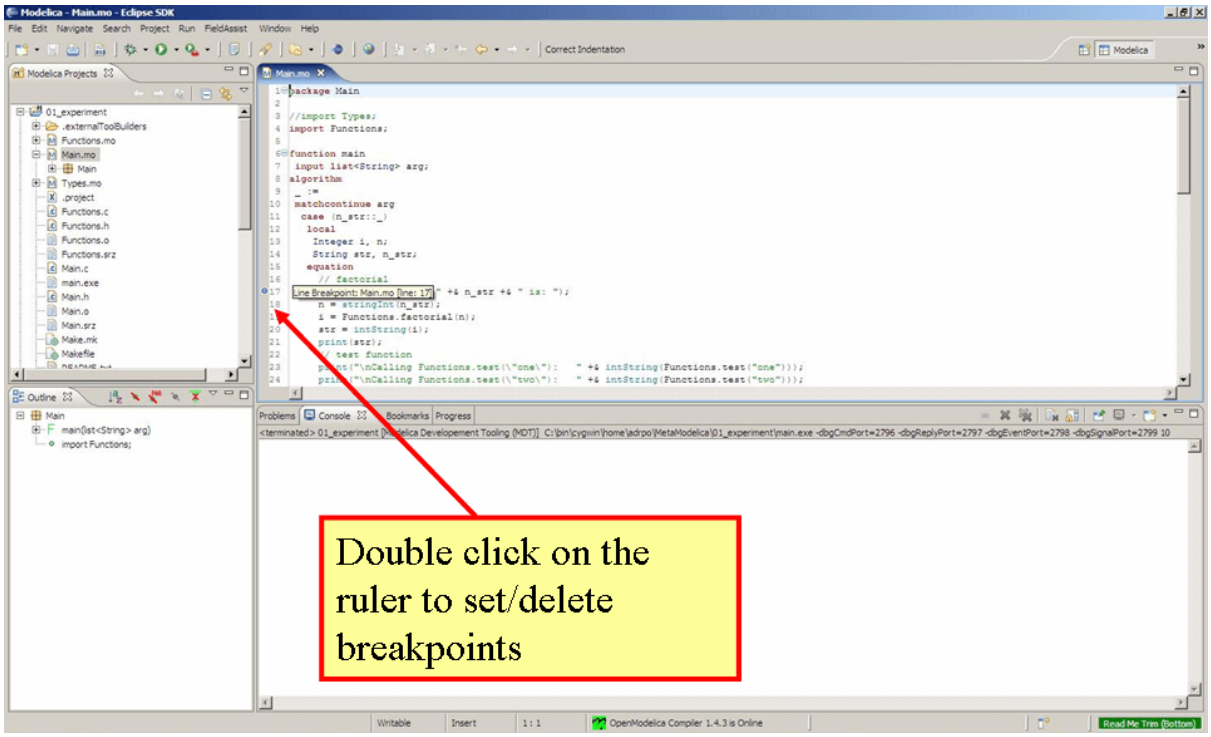


Figure 22.4: Setting/deleting breakpoints.

Starting the debugging session and enabling the debug perspective

22.1.2 The Debugging Perspective

The debug view primarily consists of two main views:

- Stack Frames View
- Variables View

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

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

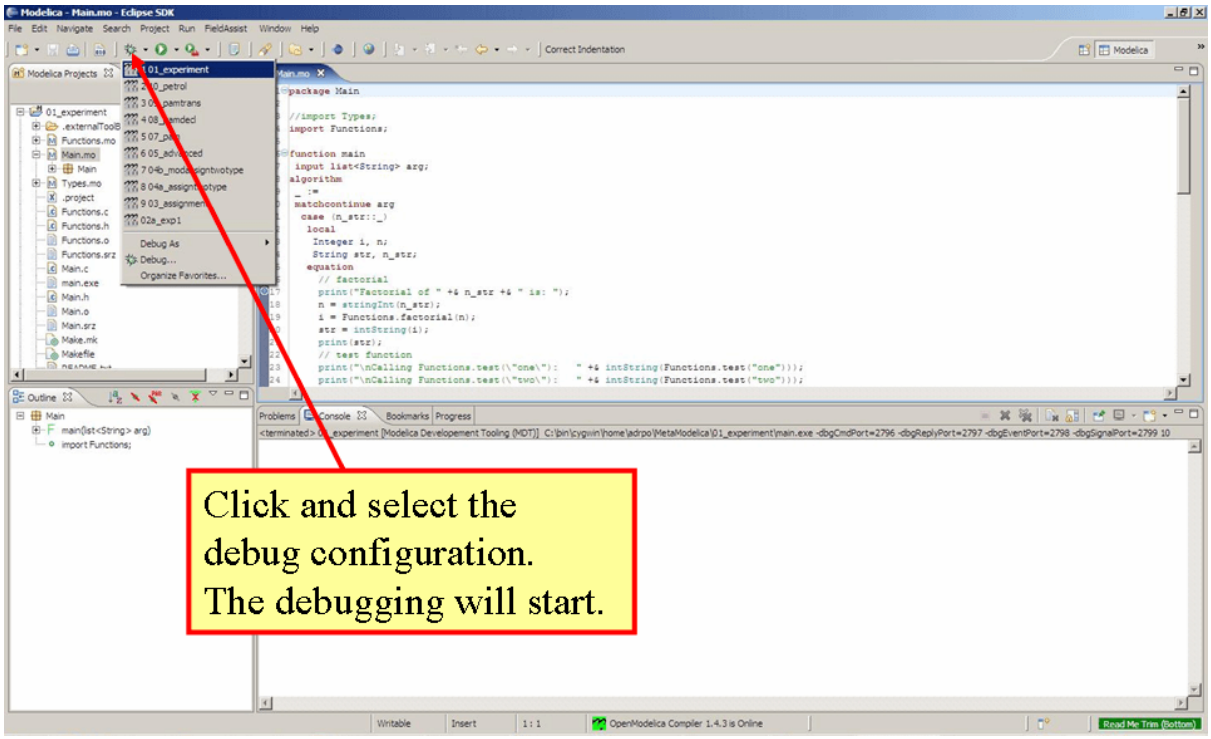


Figure 22.5: Starting the debugging session.

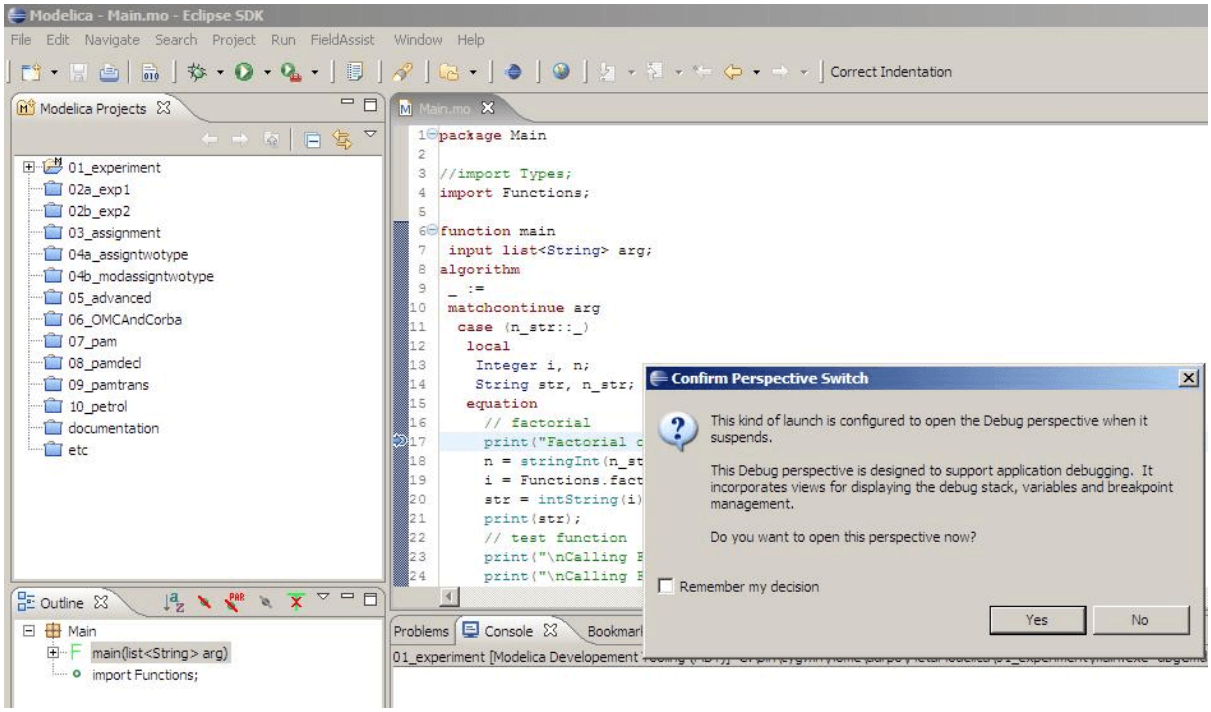


Figure 22.6: Eclipse will ask if the user wants to switch to the debugging perspective.

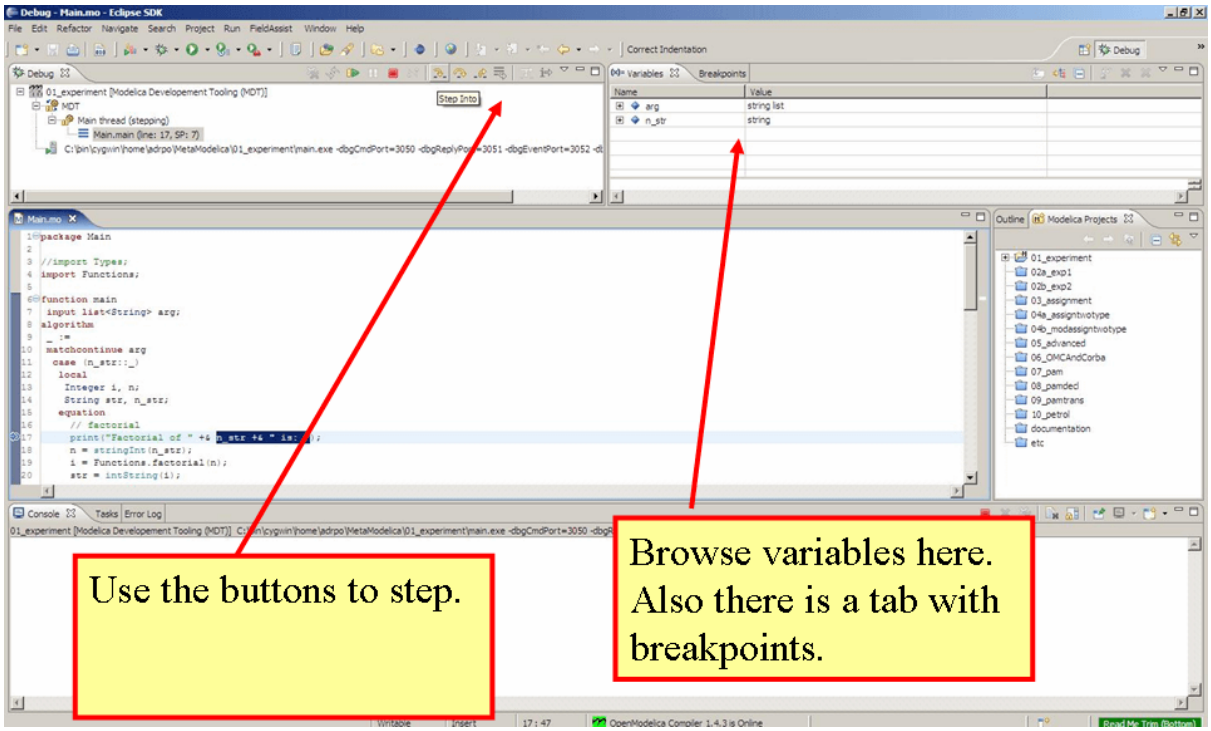


Figure 22.7: The debugging perspective.

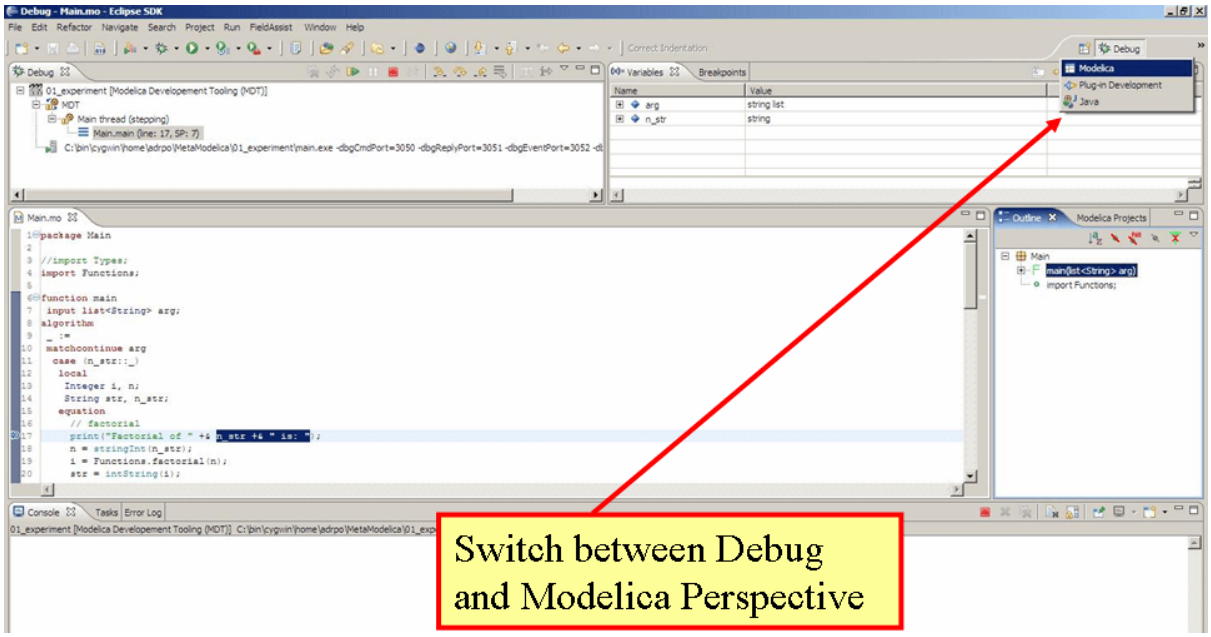


Figure 22.8: Switching between perspectives.

CHAPTER

TWENTYTHREE

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, turn on profiling with the following command line flag >>> setCommandLineOptions("--profiling=all")

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

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

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

We simulate as usual, after setting the profiling flag:

```
>>> setCommandLineOptions("--profiling=blocks+html")
true
>>> simulate(ProfilingTest)
record SimulationResult
    resultFile = "«DOCHOME»/ProfilingTest_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'ProfilingTest', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
```

(continues on next page)

(continued from previous page)

```
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
LOG_STDOUT        | info    | Time measurements are stored in ProfilingTest_prof.html␣
↪(human-readable) and ProfilingTest_prof.xml (for XSL transforms or more details)
",
    timeFrontend = 0.0011654710000000002,
    timeBackend = 0.011533901000000001,
    timeSimCode = 0.002320803,
    timeTemplates = 0.002549202,
    timeCompile = 0.165308343,
    timeSimulation = 0.017781134,
    timeTotal = 0.200704568
end SimulationResult;
"Warning: There are nonlinear iteration variables with default zero start attribute␣
↪found in NLSJac0. For more information set -d=initialization. In OMEdit Tools->
↪Options->Simulation->Show additional information from the initialization process,␣
↪in OMNotebook call setCommandLineOptions("-d=initialization").
Warning: The initial conditions are not fully specified. For more information set -
↪d=initialization. In OMEdit Tools->Options->Simulation->Show additional information␣
↪from the initialization process, in OMNotebook call setCommandLineOptions("-
↪d=initialization")."
```

23.1 Profiling information for ProfilingTest

23.1.1 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 ProfilingTest_prof.xml.

23.1.2 Settings

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

23.1.3 Summary

Task	Time	Fraction
Pre-Initialization	0.000062	3.12%
Initialization	0.000216	10.86%
Event-handling	0.000010	0.50%
Creating output file	0.000276	13.88%
Linearization		NaN%
Time steps	0.001209	60.78%
Overhead	0.000086	4.32%
Unknown	NaN	NaN%
Total simulation time	0.001989	100.00%

23.1.4 Global Steps

	Steps	Total Time	Fraction	Average Time	Max Time	Deviation
|Graph th umbnail 999|	499	0 .001209	60.78%	2.4228 4569138 277e-06	0.00 0052008	20.47x

23.1.5 Measured Function Calls

	Name	Calls	Time	Fraction	Max Time	Deviation
|Graph th umbnail function fun0| |Graph th umbnail count function fun0|	`ProfilingTest.f <#line=0>`__	506	0.00 0004349	0.22%	0.00 0000140	15.29x

23.1.6 Measured Blocks

	Name	Calls	Time	F raction	Max Time	De viation
|Graph th umbnail eq0| |Graph th umbnail count eq0|	` <# eq0>`__	8	0.00 0054663	2.75%	0.00 0054803	7.02x
|Graph th umbnail eq11| |Graph th umbnail count eq11|	` <#e q11>`__	2	0.00 0000902	0.05%	0.00 0000922	1.04x
|Graph th umbnail eq19| |Graph th umbnail count eq19|	` <#e q19>`__	504	0.00 0415407	20.89%	0.00 0021642	25.26x
|Graph th umbnail eq21| |Graph th umbnail count eq21|	` <#e q21>`__	504	0.00 0430250	21.63%	0.00 0019235	21.53x

Equations

Name	Variables
eq0	
eq1	*y*
eq2	*s*
eq3	*z1*
eq4	
eq5	` <#var0>`__
eq6	` <#var0>`__
eq7	` <#var0>`__
eq8	` <#var0>`__
eq9	*z2*
eq10	
eq11	*x*
eq12	
eq13	*z2*
eq14	
eq15	` <#var0>`__
eq16	` <#var0>`__
eq17	` <#var0>`__
eq18	` <#var0>`__
eq19	*z1*
eq20	
eq21	*x*
eq22	*der(y)*
eq23	

Variables

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

This report was generated by OpenModelica on 2026-09-22 08:25:26.

23.2 Genenerated JSON for the Example

Listing 23.1: ProfilingTest_prof.json


```
{
"name":"ProfilingTest",
"prefix":"ProfilingTest",
"date":"2026-09-22 08:25:26",
"method":"dassl",
"outputFormat":"mat",
"outputFilename":"ProfilingTest_res.mat",
"outputFilesize":24581,
```


(continues on next page)

(continued from previous page)


```
"overheadTime":8.6219e-05,
"preinitTime":6.1626e-05,
"initTime":0.000216457,
"eventTime":9.954e-06,
"outputTime":0.000276412,
"jacobianTime":9.21e-06,
"totalTime":0.00198915,
"totalStepsTime":2.264e-06,
"totalTimeProfileBlocks":0.000901222,
"numStep":499,
"maxTime":5.2008e-05,
"functions":[
{"name":"ProfilingTest.f","ncall":506,"time":0.000004349,"maxTime":0.000000140}
],
"profileBlocks":[
{"id":0,"ncall":8,"time":0.000054663,"maxTime":0.000054803},
{"id":11,"ncall":2,"time":0.000000902,"maxTime":0.000000922},
{"id":19,"ncall":504,"time":0.000415407,"maxTime":0.000021642},
{"id":21,"ncall":504,"time":0.000430250,"maxTime":0.000019235}
]
}
```


23.3 Using the Profiler from OMEdit

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

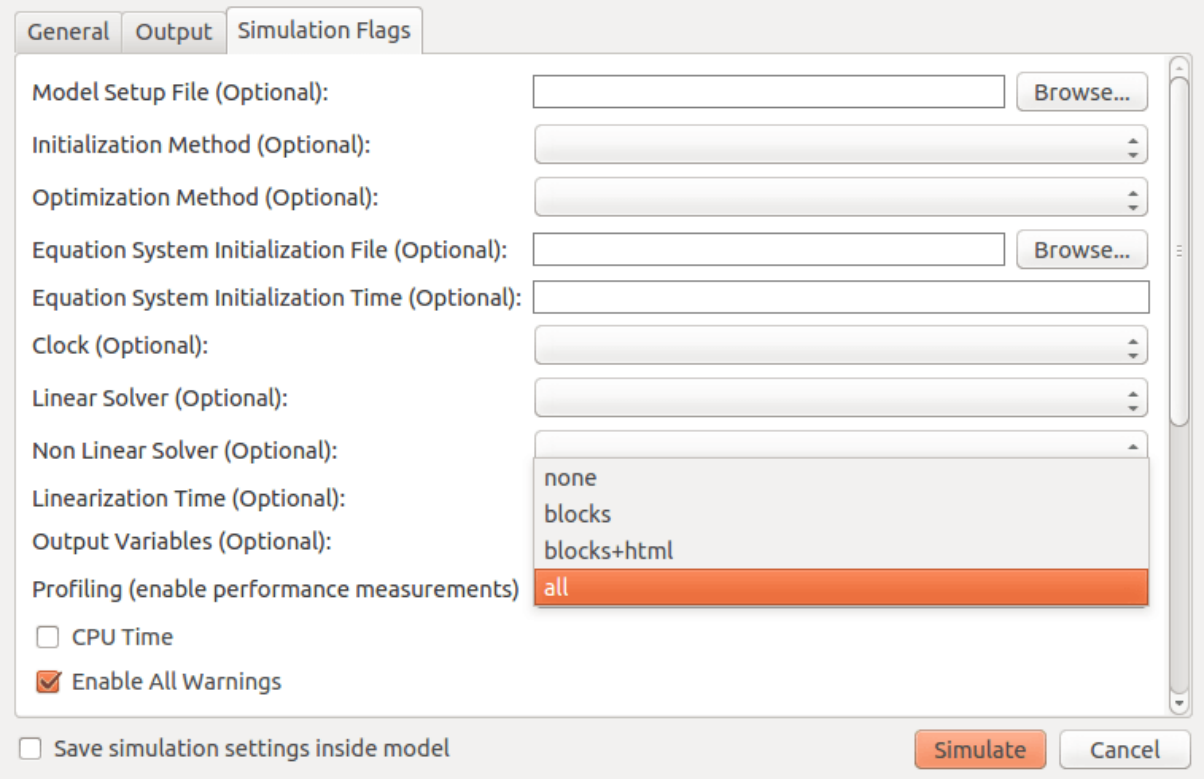


Figure 23.1: Setting up the profiler from OMEdit.

When profiling the DoublePendulum example from MSL, the following output in Figure 23.2 is a typical result. This information clearly shows which system takes longest to simulate (a linear system, where most of the time

overhead probably comes from initializing LAPACK over and over).

Equations Browser

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

Defines

Variable
damper.a_rel
revolute2.frame_b.f[2]

Figure 23.2: Profiling results of the Modelica standard library DoublePendulum example, sorted by execution time.

CHAPTER

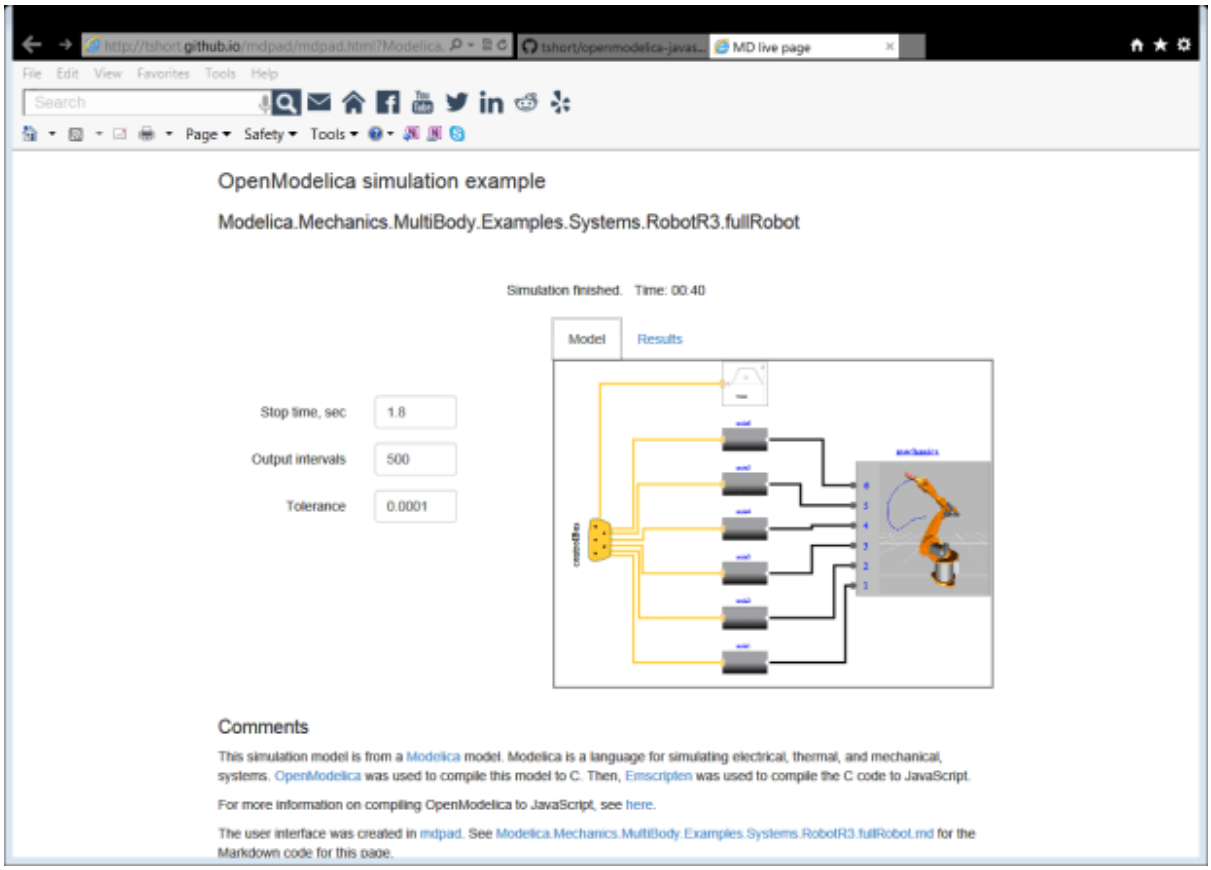
TWENTYFOUR

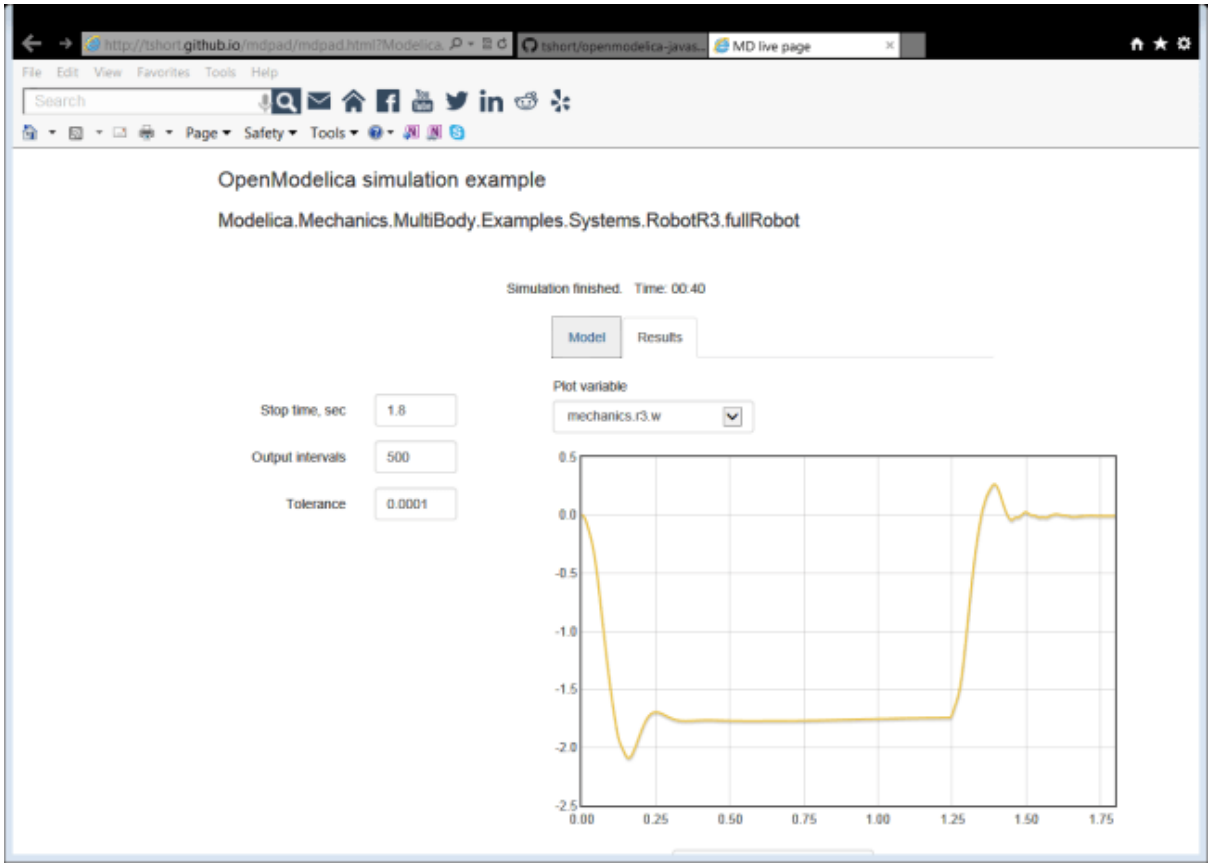
SIMULATION IN WEB BROWSER

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.




MD live page
File Edit View Favorites Tools Help
Search
Page Safety Tools
OpenModelica simulation example
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot
Simulation finished. Time: 00:40
Model
Results
Stop time, sec
1.8
Output intervals
500
Tolerance
0.0001
Plot variable
mechanics.r3.w
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

CHAPTER

TWENTYFIVE

INTEROPERABILITY - C AND PYTHON

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

25.1 Calling External C functions

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

```
model ExternalLibraries

  function ExternalFunc1
    input Real x;
    output Real y;
  external y=ExternalFunc1_ext(x) annotation(Library="ExternalFunc1.o",␣
↪LibraryDirectory="modelica://ExternalLibraries", Include="#include \"ExternalFunc1.
↪h\"");
  end ExternalFunc1;

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

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

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

Listing 25.1: ExternalFunc1.c

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

Listing 25.2: ExternalFunc1.h

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

Listing 25.3: ExternalFunc2.c

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

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

```
>>> system(getCompiler() + " -c -o ExternalFunc1.o ExternalFunc1.c")
0
>>> system(getCompiler() + " -c -o ExternalFunc2.o ExternalFunc2.c")
0
>>> system("ar rcs libExternalFunc2.a ExternalFunc2.o")
0
>>> simulate(ExternalLibraries)
record SimulationResult
    resultFile = "«DOCHOME»/ExternalLibraries_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'ExternalLibraries', options =
↪'', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.001173947,
    timeBackend = 0.003796316,
    timeSimCode = 9.736500000000001e-4,
    timeTemplates = 0.001912616,
    timeCompile = 0.171952031,
    timeSimulation = 0.005707819,
    timeTotal = 0.185586941
end SimulationResult;
```

Note

Notification: Model statistics after passing the front-end and creating the data structures used by the back-end:

* Number of equations: 2

* Number of variables: 2

Notification: Model statistics after passing the back-end for initialization:

* Number of independent subsystems: 2

* Number of states: 0 ()

* Number of discrete variables: 0 ()

* Number of discrete states: 0 ()

```
* Number of clocked states: 0 ()
* Top-level inputs: 0
Notification: Strong component statistics for initialization (4):
* Single equations (assignments): 4
* Array equations: 0
* Algorithm blocks: 0
* Record equations: 0
* When equations: 0
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0
Notification: Model statistics after passing the back-end for simulation:
* Number of independent subsystems: 2
* Number of states: 2 (y,x)
* Number of discrete variables: 0 ()
* Number of discrete states: 0 ()
* Number of clocked states: 0 ()
* Top-level inputs: 0
Notification: Strong component statistics for simulation (2):
* Single equations (assignments): 2
* Array equations: 0
* Algorithm blocks: 0
* Record equations: 0
* When equations: 0
* If-equations: 0
* Equation systems (not torn): 0
* Torn equation systems: 0
* Mixed (continuous/discrete) equation systems: 0
```

And plot the results:

25.2 Calling external Python Code from a Modelica model

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

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

(continues on next page)

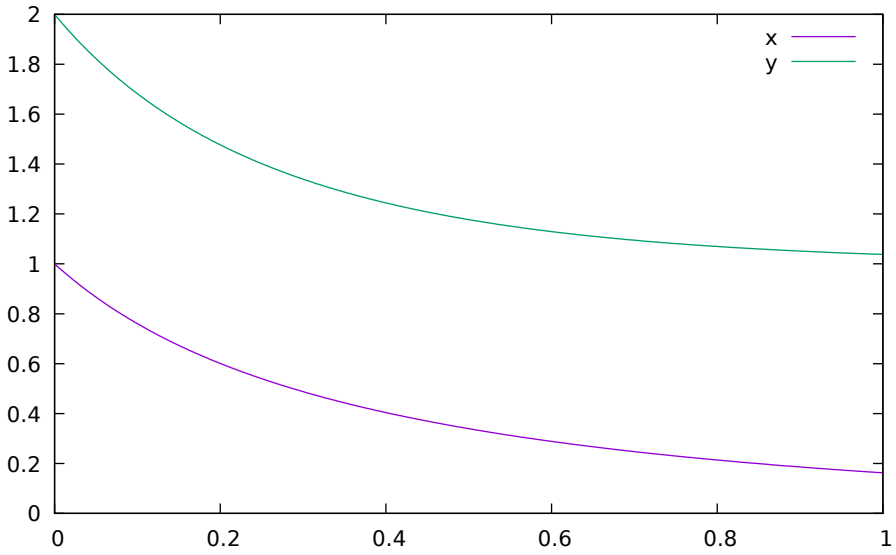


Figure 25.1: Plot generated by OpenModelica+gnuplot

(continued from previous page)

```
void pyRunString(const char *str)
{
  Py_Initialize();
  PyRun_SimpleString(str);
  Py_Finalize();
}
");
end pyRunString;

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

```
>>> system("python3-config --cflags > pycflags")
0
>>> system("python3-config --ldflags --embed > pyldflags")
0
>>> pycflags := stringReplace(readFile("pycflags"),"\n","");
>>> pyldflags := stringReplace(readFile("pyldflags"),"\n","");
>>> setCFlags(getCFlags()+pycflags)
true
>>> setLinkerFlags(getLinkerFlags()+pyldflags)
true
>>> simulate(CallExternalPython, stopTime=2)
record SimulationResult
    resultFile = "«DOCHOME»/CallExternalPython_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 2.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'CallExternalPython', options␣
↪= '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "Python says: simulation time 0
```

(continues on next page)

(continued from previous page)

```
Python says: simulation time 0
LOG_SUCCESS       | info    | The initialization finished successfully without␣
↪homotopy method.
Python says: simulation time 2
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.00106387,
    timeBackend = 0.010063987,
    timeSimCode = 0.0015922730000000002,
    timeTemplates = 0.0015402550000000001,
    timeCompile = 0.21307071300000002,
    timeSimulation = 0.024617664,
    timeTotal = 0.25201656
end SimulationResult;
```

Note

Notification: Model statistics after passing the front-end and creating the data structures used by the back-end:

* Number of equations: 0

* Number of variables: 0

Notification: Model statistics after passing the back-end for initialization:

* Number of independent subsystems: 1

* Number of states: 0 ()

* Number of discrete variables: 0 ()

* Number of discrete states: 0 ()

* Number of clocked states: 0 ()

* Top-level inputs: 0

Notification: Strong component statistics for initialization (0):

* Single equations (assignments): 0

* Array equations: 0

* Algorithm blocks: 0

* Record equations: 0

* When equations: 0

* If-equations: 0

* Equation systems (not torn): 0

* Torn equation systems: 0

* Mixed (continuous/discrete) equation systems: 0

Notification: Model statistics after passing the back-end for simulation:

* Number of independent subsystems: 1

* Number of states: 0 ()

* Number of discrete variables: 0 ()

* Number of discrete states: 0 ()

* Number of clocked states: 0 ()

* Top-level inputs: 0

Notification: Strong component statistics for simulation (0):

* Single equations (assignments): 0

* Array equations: 0

* Algorithm blocks: 0

* Record equations: 0

* When equations: 0

* If-equations: 0

* Equation systems (not torn): 0

* Torn equation systems: 0

* Mixed (continuous/discrete) equation systems: 0

25.3 Calling OpenModelica from Python Code

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

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

Listing 25.4: CalledbyPython.mo

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

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

Listing 25.5: PythonCaller.py

```
#!/usr/bin/env python3
import os

newb = 0.5
exec(open('CreateMosFile.py').read())
os.popen(r"omc CalledbyPython.mos").read()
exec(open('RetrResult.py').read())
```

Listing 25.6: CreateMosFile.py

```
#!/usr/bin/env python3
mos_file = open('CalledbyPython.mos','w', 1)
mos_file.write('loadFile("CalledbyPython.mo");\n')
mos_file.write('setComponentModifierValue(CalledbyPython,b,$Code(="+str(newb)+"));\n')
mos_file.write('simulate(CalledbyPython,stopTime=10);\n')
mos_file.close()
```

Listing 25.7: RetrResult.py

```
#!/usr/bin/env python3
def zeros(n):
  return [0.0] * int(n)
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 ZeroMQ 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

TWENTYSIX

OPENMODELICA PYTHON INTERFACE

This chapter describes the OpenModelica Python integration facilities.

- OMPython - the OpenModelica Python scripting interface, see *OMPython - OpenModelica Python Interface*.
- EnhancedOMPython - Enhanced OMPython scripting interface, see *Enhanced OMPython Features*.

26.1 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 is not a standalone package, it depends upon the OpenModelica installation.

OMPython is implemented in Python and depends on ZeroMQ - high performance asynchronous messaging library.

To install OMPython follow the instructions at https://github.com/OpenModelica/OMPython

26.1.1 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.

26.1.2 Test Commands

OMPython provides a OMCSessionZMQ class that uses ZeroMQ to communicate with OpenModelica.

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

To get started, create an OMCSessionZMQ object:

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

```
>>> omc.sendExpression("getVersion()")
v1.28.0-dev.838+g4c2a253e5a.cmake
>>> omc.sendExpression("cd()")
«DOCHOME»
>>> omc.sendExpression("loadModel(Modelica)")
True
>>> omc.sendExpression("loadFile(getInstallationDirectoryPath() + \"/share/doc/omc/
↪testmodels/BouncingBall.mo\")")
True
>>> omc.sendExpression("instantiateModel(BouncingBall)")
class BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(start = 1.0, fixed = true) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(start = true, fixed = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0.0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then -e * pre(v) else 0.0;
    flying = v_new > 0.0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

We get the name and other properties of a class:

```
>>> omc.sendExpression("getClassNames()")
('BouncingBall', 'ModelicaServices', 'Complex', 'Modelica')
>>> omc.sendExpression("isPartial(BouncingBall)")
False
>>> omc.sendExpression("isPackage(BouncingBall)")
False
>>> omc.sendExpression("isModel(BouncingBall)")
True
>>> omc.sendExpression("checkModel(BouncingBall)")
Check of BouncingBall completed successfully.
Class BouncingBall has 6 equation(s) and 6 variable(s).
1 of these are trivial equation(s).
>>> omc.sendExpression("getClassRestriction(BouncingBall)")
model
>>> omc.sendExpression("getClassInformation(BouncingBall)")
('model', '', False, False, False, '/var/lib/jenkins/ws/OpenModelica/build/share/doc/
↪omc/testmodels/BouncingBall.mo', False, 1, 1, 26, 17, (), False, False, '', '',␣
↪False, '', '', '', '', '')
>>> omc.sendExpression("getConnectionCount(BouncingBall)")
0
>>> omc.sendExpression("getInheritanceCount(BouncingBall)")
0
>>> omc.sendExpression("getComponentModifierValue(BouncingBall,e)")
0.7
```

(continues on next page)

(continued from previous page)

```
>>> omc.sendExpression("checkSettings()")
{'OPENMODELICAHOME': '«OPENMODELICAHOME»', 'OPENMODELICALIBRARY': '«OPENMODELICAHOME»/
↪lib/omlibrary', 'OMC_PATH': '«OPENMODELICAHOME»/bin/omc', 'SYSTEM_PATH': '/var/lib/
↪jenkins/ws/OpenModelica/build/bin:/var/lib/jenkins/ws/OpenModelica/build/bin:/usr/
↪local/cargo/bin:/opt/venv/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
↪sbin:/bin', 'OMDEV_PATH': '', 'OMC_FOUND': True, 'MODELICAUSERCFLAGS': '', 'WORKING_
↪DIRECTORY': '«DOCHOME»', 'CREATE_FILE_WORKS': True, 'REMOVE_FILE_WORKS': True, 'OS
↪': 'linux', 'SYSTEM_INFO': 'Linux 8aa239e33368 6.8.0-138-generic #138-Ubuntu SMP␣
↪PREEMPT_DYNAMIC Fri Jul 31 22:41:49 UTC 2026 x86_64 x86_64 x86_64 GNU/Linux\n',
↪'RTLIBS': ' -lOpenModelicaRuntimeC -lomcgc -llapack -lblas -lm -lpthread -rdynamic -
↪-coverage ', 'C_COMPILER': 'cc', 'C_COMPILER_VERSION': 'cc (Ubuntu 11.4.0-1ubuntu1~
↪22.04.3) 11.4.0\nCopyright (C) 2021 Free Software Foundation, Inc.\nThis is free␣
↪software; see the source for copying conditions.  There is NO\nwarranty; not even␣
↪for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.\n\n', 'C_COMPILER_
↪RESPONDING': True, 'CONFIGURE_CMDLINE': 'Configured  using arguments: '}
```

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

```
>>> omc.sendExpression("simulate(BouncingBall, stopTime=3.0)")
{'resultFile': '«DOCHOME»/BouncingBall_res.mat', 'simulationOptions': "startTime = 0.
↪0, stopTime = 3.0, numberOfIntervals = 500, tolerance = 1e-6, method = 'dassl',␣
↪fileNamePrefix = 'BouncingBall', options = '', outputFormat = 'mat', variableFilter␣
↪= '.*', cflags = '', simflags = ''", 'messages': 'LOG_SUCCESS       | info    | The␣
↪initialization finished successfully without homotopy method.\nLOG_SUCCESS       |␣
↪info    | The simulation finished successfully.\n', 'timeFrontend': 0.
↪0012992330000000001, 'timeBackend': 0.006498576000000001, 'timeSimCode': 0.
↪0019479520000000002, 'timeTemplates': 0.0026548310000000003, 'timeCompile': 0.
↪176783153, 'timeSimulation': 0.007225912, 'timeTotal': 0.196501717000000002}
>>> omc.sendExpression("val(h , 2.0)")
0.04239430772884106
```

Import As Library

To use the module from within another python program, simply import OMCSessionZMQ from within the using program.

For example:

```
# test.py
from OMPython import OMCSessionZMQ
omc = OMCSessionZMQ()
cmds = [
  'loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/BouncingBall.
↪mo")',
  "simulate(BouncingBall)",
  "plot(h)"
  ]
for cmd in cmds:
  answer = omc.sendExpression(cmd)
  print("\n{}:\n{}".format(cmd, answer))
```

26.1.3 Implementation

Client Implementation

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

OMPython is designed to,

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

26.2 Enhanced OMPython Features

Some more improvements are added to OMPython functionality for querying more information about the models and simulate them. A list of new user friendly API functionality allows user to extract information about models using python objects. A list of API functionality is described below.

To get started, create a ModelicaSystem object:

```
>>> from OMPython import OMCSessionZMQ
>>> omc = OMCSessionZMQ()
>>> model_path=omc.sendExpression("getInstallationDirectoryPath()") + "/share/doc/omc/
↪testmodels/"
>>> from OMPython import ModelicaSystem
>>> mod=ModelicaSystem(model_path + "BouncingBall.mo","BouncingBall")
```

The object constructor requires a minimum of 2 input arguments which are strings, and may need a third string input argument.

- The first input argument must be a string with the file name of the Modelica code, with Modelica file extension ".mo". If the Modelica file is not in the current directory of Python, then the file path must also be included.
- The second input argument must be a string with the name of the Modelica model including the namespace if the model is wrapped within a Modelica package.
- The third input argument (optional) is used to specify the list of dependent libraries or dependent Modelica files e.g.,

```
>>> mod=ModelicaSystem(model_path + "BouncingBall.mo","BouncingBall",["Modelica"])
```

- The fourth input argument (optional), is a keyword argument which is used to set the command line options e.g.,

```
>>> mod=ModelicaSystem(model_path + "BouncingBall.mo","BouncingBall",
↪commandLineOptions="-d=newInst")
```

26.2.1 BuildModel

The buildModel API can be used after ModelicaSystem(), in case the model needs to be updated or additional simulationflags needs to be set using sendExpression()

```
>>> mod.buildModel()
```

26.2.2 Standard get methods

- getQuantities()
- getContinuous()
- getInputs()
- getOutputs()
- getParameters()

- getSimulationOptions()
- getSolutions()

Three calling possibilities are accepted using getXXX() where "XXX" can be any of the above functions (eg:) getParameters().

- getXXX() without input argument, returns a dictionary with names as keys and values as values.
- getXXX(S), where S is a string of names.
- getXXX(["S1","S2"]) where S1 and S1 are list of string elements

26.2.3 Usage of getMethods

```
>>> mod.getQuantities() // method-1, list of all variables from xml file
[{'aliasvariable': None, 'Name': 'height', 'Variability': 'continuous', 'Value': '1.0
↪', 'alias': 'noAlias', 'Changeable': 'true', 'Description': None}, {'aliasvariable
↪': None, 'Name': 'c', 'Variability': 'parameter', 'Value': '0.9', 'alias': 'noAlias
↪', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getQuantities("height") // method-2, to query information about single␣
↪quantity
[{'aliasvariable': None, 'Name': 'height', 'Variability': 'continuous', 'Value': '1.0
↪', 'alias': 'noAlias', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getQuantities(["c","radius"]) // method-3, to query information about list of␣
↪quantity
[{'aliasvariable': None, 'Name': 'c', 'Variability': 'parameter', 'Value': '0.9',
↪'alias': 'noAlias', 'Changeable': 'true', 'Description': None}, {'aliasvariable':␣
↪None, 'Name': 'radius', 'Variability': 'parameter', 'Value': '0.1', 'alias':
↪'noAlias', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getContinuous() // method-1, list of continuous variable
{'velocity': -1.825929609047952, 'der(velocity)': -9.8100000000000005, 'der(height)':␣
↪-1.825929609047952, 'height': 0.65907039052943617}
```

```
>>> mod.getContinuous(["velocity","height"]) // method-2, get specific variable value␣
↪information
(-1.825929609047952, 0.65907039052943617)
```

```
>>> mod.getInputs()
{}
```

```
>>>  mod.getOutputs()
{}
```

```
>>> mod.getParameters()  // method-1
{'c': 0.9, 'radius': 0.1}
```

```
>>> mod.getParameters(["c","radius"]) // method-2
[0.9, 0.1]
```

```
>>> mod.getSimulationOptions()  // method-1
{'stepSize': 0.002, 'stopTime': 1.0, 'tolerance': 1e-06, 'startTime': 0.0, 'solver':
↪'dassl'}
```

```
>>> mod.getSimulationOptions(["stepSize","tolerance"]) // method-2
[0.002, 1e-06]
```

The getSolution method can be used in two different ways.

1. using default result filename
2. use the result filenames provided by user

This provides a way to compare simulation results and perform regression testing

```
>>> mod.getSolutions() // method-1 returns list of simulation variables for which␣
↪results are available
['time', 'height', 'velocity', 'der(height)', 'der(velocity)', 'c', 'radius']
```

```
>>> mod.getSolutions(["time","height"])  // return list of numpy arrays
```

```
>>> mod.getSolutions(resultfile="c:/tmpbouncingBall.mat") // method-2 returns list of␣
↪simulation variables for which results are available , the resulfile location is␣
↪provided by user
```

```
>>> mod.getSolutions(["time","height"],resultfile="c:/tmpbouncingBall.mat") // return␣
↪list of array
```

26.2.4 Standard set methods

- setInputs()
- setParameters()
- setSimulationOptions()

Two setting possibilities are accepted using setXXXs(),where "XXX" can be any of above functions.

- setXXX("Name=value") string of keyword assignments
- setXXX(["Name1=value1","Name2=value2","Name3=value3"]) list of string of keyword assignments

26.2.5 Usage of setMethods

```
>>> mod.setInputs(["cAi=1","Ti=2"]) // method-2
```

```
>>> mod.setParameters("radius=14") // method-1 setting parameter value
```

```
>>> mod.setParameters(["radius=14","c=0.5"]) // method-2 setting parameter value␣
↪using second option
```

```
>>> mod.setSimulationOptions(["stopTime=2.0","tolerance=1e-08"]) // method-2
```

26.2.6 Simulation

An example of how to get parameter names and change the value of parameters using set methods and finally simulate the "BouncingBall.mo" model is given below.

```
>>>  mod.getParameters()
{'c': 0.9, 'radius': 0.1}
```

```
>>>  mod.setParameters(["radius=14","c=0.5"]) //setting parameter value
```

To check whether new values are updated to model , we can again query the getParameters().

```
>>> mod.getParameters()
{'c': 0.5, 'radius': 14}
```

The model can be simulated using the *simulate* API in the following ways,

1. without any arguments
2. resultfile (keyword argument) - (only filename is allowed and not the location)
3. simargs (keyword argument) - runtime simulationflags supported by OpenModelica

```
>>> mod.simulate() // method-1 default result file name will be used
>>> mod.simulate(resultfile="tmpbouncingBall.mat")  // method-2 resultfile name␣
↪provided by users
>>> mod.simulate(simargs={"noEventEmit": None, "noRestart": None, "override": {"e": 0.
↪3, "g": 10}}) // method-3 simulationflags provided by users
```

26.2.7 Linearization

The following methods are proposed for linearization.

- linearize()
- getLinearizationOptions()
- setLinearizationOptions()
- getLinearInputs()
- getLinearOutputs()
- getLinearStates()

26.2.8 Usage of Linearization methods

```
>>> mod.getLinearizationOptions()  // method-1
{'simflags': ' ', 'stepSize': 0.002, 'stopTime': 1.0, 'startTime': 0.0,
↪'numberOfIntervals': 500.0, 'tolerance': 1e-08}
```

```
>>> mod.getLinearizationOptions("startTime","stopTime") // method-2
[0.0, 1.0]
```

```
>>> mod.setLinearizationOptions(["stopTime=2.0","tolerance=1e-06"])
```

```
>>> mod.linearize()  //returns a tuple of 2D numpy arrays (matrices) A, B, C and D.
```

```
>>> mod.getLinearInputs()  //returns a list of strings of names of inputs used when␣
↪forming matrices.
```

```
>>> mod.getLinearOutputs() //returns a list of strings of names of outputs used when␣
↪forming matrices
```

```
>>> mod.getLinearStates() // returns a list of strings of names of states used when␣
↪forming matrices.
```

CHAPTER

TWENTYSEVEN

OMMATLAB - OPENMODELICA MATLAB INTERFACE

OMMatlab - the OpenModelica Matlab API is a free, open source, highly portable Matlab-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. OMMatlab 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. OMMatlab is not a standalone package, it depends upon the OpenModelica installation.

OMMatlab is implemented in Matlab and depends on ZeroMQ - high performance asynchronous messaging library and it supports the Modelica Standard Library version 3.2 that is included in starting with OpenModelica 1.9.2.

To install OMMatlab follow the instructions at https://github.com/OpenModelica/OMMatlab

27.1 Features of OMMatlab

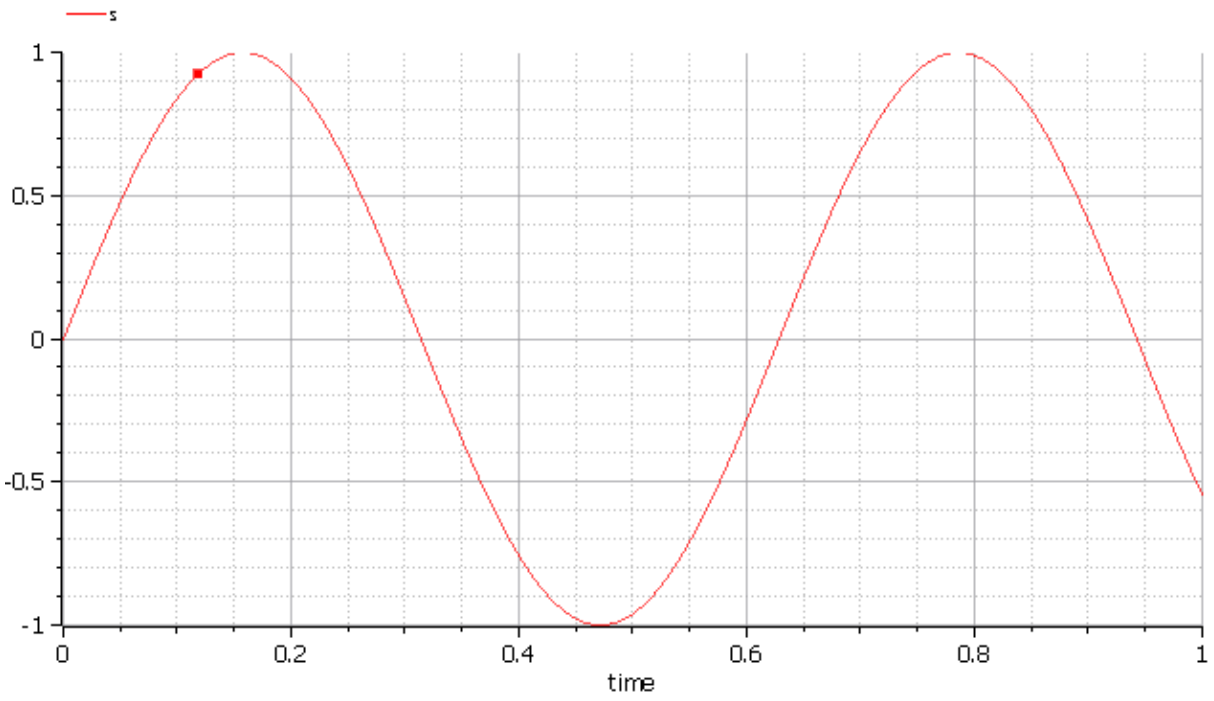
The OMMatlab package contains the following features:

- Import the OMMatlab package in Matlab
- Connect with the OpenModelica compiler through zmq sockets
- Able to interact with the OpenModelica compiler through the *available API*
- All the API calls are communicated with the help of the sendExpression method implemented in a Matlab pacakge
- The results are returned as strings

27.2 Test Commands

To get started, create a OMMatlab session object:

```
>>> import OMMatlab.*
>>> omc= OMMatlab()
>>> omc.sendExpression("getVersion()")
'v1.13.0-dev-531-gde26b558a (64-bit)'
>>> omc.sendExpression("loadModel(Modelica)")
'true'
>>> omc.sendExpression("model a Real s; equation s=sin(10*time); end a;")
'{a}'
>>> omc.sendExpression("simulate(a)")
>>> omc.sendExpression("plot(s)")
'true'
```



27.2.1 Advanced OMMatlab Features

OMMatlab package has advanced functionality for querying more information about the models and simulate them. A list of new user friendly API functionality allows user to extract information about models using matlab objects. A list of API functionality is described below.

To get started, create a ModelicaSystem object:

```
>>> import OMMatlab.*
>>> omc= OMMatlab()
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall")
```

The object constructor requires a minimum of 2 input arguments which are strings, and third input argument which is optional .

- The first input argument must be a string with the file name of the Modelica code, with Modelica file extension ".mo". If the Modelica file is not in the current directory, then the file path must also be included.
- The second input argument must be a string with the name of the Modelica model including the namespace if the model is wrapped within a Modelica package.
- The third input argument (optional) is used to specify the list of dependent libraries or dependent Modelica files The argument can be passed as a string or array of strings e.g.,

```
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall",["Modelica", "SystemDynamics",
↪ "dcmotor.mo"])
```

- The fourth input argument (optional), which is used to set the command line options e.g.,

```
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall",["Modelica", "SystemDynamics",
↪ "dcmotor.mo"],"-d=newInst")
```

Matlab does not support keyword arguments, and hence inorder to skip an argument, empty list should be used "[]" e.g.,

```
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall",[],"-d=newInst")
```

27.3 WorkDirectory

For each Matlab session a temporary work directory is created and the results are published in that working directory, Inorder to get the workdirectory the users can use the following API

```
>>> omc.getWorkDirectory()
'C:/Users/arupa54/AppData/Local/Temp/tp7dd648e5_5de6_4f66_b3d6_90bce1fe1d58'
```

27.4 BuildModel

The buildModel API can be used after ModelicaSystem(), in case the model needs to be updated or additional simulationflags needs to be set using sendExpression()

```
>>> omc.buildModel()
```

27.5 Standard get methods

- getQuantities()
- showQuantities()
- getContinuous()
- getInputs()
- getOutputs()
- getParameters()
- getSimulationOptions()
- getSolutions()

Three calling possibilities are accepted using getXXX() where "XXX" can be any of the above functions (eg:) getParameters().

- getXXX() without input argument, returns a dictionary with names as keys and values as values.
- getXXX(S), where S is a string of names.
- getXXX(["S1","S2"]) where S1 and S1 are array of string elements

27.6 Usage of getMethods

```
>>> omc.getQuantities() // method-1, list of all variables from xml file
+----------+------------+--------------------------+--------------+------------+-------
↪----+---------------+-------+
| name     | changeable | description              | variability  | causality  | alias␣
↪    | aliasVariable | value |
+----------+------------+--------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'h'      | 'true'     | 'height of ball'         | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+--------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'v'      | 'true'     | 'velocity of ball'       | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+--------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'der(h)' | 'false'    | 'der(height of ball)'    | 'continuous' | 'internal' |
```

(continues on next page)

(continued from previous page)

```
↪'noAlias' | ''            | ''    |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'der(v)' | 'false'    | 'der(velocity of ball)' | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
```

```
>>> omc.getQuantities("h") // method-2, to query information about single quantity
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| name     | changeable | description             | variability  | causality  | alias␣
↪    | aliasVariable | value |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'h'      | 'true'     | 'height of ball'        | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
```

```
>>> omc.getQuantities(["h","v"]) // method-3, to query information about list of␣
↪quantity
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| name     | changeable | description             | variability  | causality  | alias␣
↪    | aliasVariable | value |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'h'      | 'true'     | 'height of ball'        | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
| 'v'      | 'true'     | 'velocity of ball'      | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+-------------------------+--------------+------------+-------
↪----+---------------+-------+
```

```
>>> omc.getContinuous() // method-1, returns struct of continuous variable
struct with fields:
  h     : '1.0'
  v     : ''
  der_h_: ''
  der_v_: ''
```

```
>>> omc.getContinuous(["h","v"])   // method-2, returns string array
"1.0"    ""
```

```
>>> omc.getInputs()
struct with no fields
```

```
>>> omc.getOutputs()
struct with no fields
```

```
>>> omc.getParameters()  // method-1
```

(continues on next page)

(continued from previous page)

```
struct with fields:
        e: '0.7'
        g: '9.810000000000001'
```

```
>>> omc.getParameters(["c","radius"]) // method-2
"0.7"  "9.810000000000001"
```

```
>>> omc.getSimulationOptions()  // method-1
struct with fields:
        startTime: '0'
         stopTime: '1'
         stepSize: '0.002'
        tolerance: '1e-006'
           solver: 'dassl'
```

```
>>> omc.getSimulationOptions(["stepSize","tolerance"]) // method-2
"0.002", "1e-006"
```

The getSolution method can be used in two different ways.

1. using default result filename
2. use the result filenames provided by user

This provides a way to compare simulation results and perform regression testing

```
>>> omc.getSolutions() // method-1 returns string arrays of simulation variables for␣
↪which results are available, the default result filename is taken
"time", "height", ""velocity", "der(height)", "der(velocity)", "c", "radius"
```

```
>>> omc.getSolutions(["time","h"])  // return list of cell arrays
1×2 cell array
{1×506 double}    {1×506 double}
```

```
>>> omc.getSolutions([],"c:/tmpbouncingBall.mat") // method-2 returns string arrays␣
↪of simulation variables for which results are available , the resulfile location is␣
↪provided by user
"time", "height", "velocity", "der(height)", "der(velocity)", "c", "radius"
```

```
>>> omc.getSolutions(["time","h"],"c:/tmpbouncingBall.mat") // return list of cell␣
↪arrays
1×2 cell array
{1×506 double}    {1×506 double}
```

27.7 Standard set methods

- setInputs()
- setParameters()
- setSimulationOptions()

Two setting possibilities are accepted using setXXXs(),where "XXX" can be any of above functions.

- setXXX("Name=value") string of keyword assignments
- setXXX(["Name1=value1","Name2=value2","Name3=value3"]) array of string of keyword assignments

27.8 Usage of setMethods

```
>>> omc.setInputs("cAi=1") // method-1
```

```
>>> omc.setInputs(["cAi=1","Ti=2"]) // method-2
```

```
>>> omc.setParameters("e=14") // method-1
```

```
>>> omc.setParameters(["e=14","g=10.8"]) // method-2 setting parameter value using␣
↪array of string
```

```
>>> omc.setSimulationOptions(["stopTime=2.0","tolerance=1e-08"])
```

27.9 Advanced Simulation

An example of how to do advanced simulation to set parameter values using set methods and finally simulate the "BouncingBall.mo" model is given below .

```
>>> omc.getParameters()
struct with fields:
        e: '0.7'
        g: '9.810000000000001'
```

```
>>> omc.setParameters(["e=0.9","g=9.83"])
```

To check whether new values are updated to model , we can again query the getParameters().

```
>>> omc.getParameters()
struct with fields:
    e: "0.9"
    g: "9.83"
```

Similary we can also use setInputs() to set a value for the inputs during various time interval can also be done using the following.

```
>>> omc.setInputs("cAi=1")
```

The model can be simulated using the *simulate* API in the following ways,

1. without any arguments
2. resultfile names provided by user (only filename is allowed and not the location)
3. simflags - runtime simulationflags supported by OpenModelica

```
>>> omc.simulate() // method-1 default result file name will be used
>>> omc.simulate("tmpbouncingBall.mat") // method-2 resultfile name provided by users
>>> omc.simulate([],"-noEventEmit -noRestart -override=e=0.3,g=9.71") // method-3␣
↪simulationflags provided by users, since matlab does not support keyword argument␣
↪we skip argument1 result file with empty list
```

27.10 Linearization

The following methods are available for linearization of a modelica model

- linearize()

- getLinearizationOptions()
- setLinearizationOptions()
- getLinearInputs()
- getLinearOutputs()
- getLinearStates()

27.11 Usage of Linearization methods

```
>>> omc.getLinearizationOptions()  // method-1
```

```
>>> omc.getLinearizationOptions(["startTime","stopTime"]) // method-2
"0.0", "1.0"
```

```
>>> omc.setLinearizationOptions(["stopTime=2.0","tolerance=1e-08"])
```

```
>>> omc.linearize()  //returns a list 2D arrays (matrices) A, B, C and D.
```

```
>>> omc.getLinearInputs()  //returns a list of strings of names of inputs used when␣
↪forming matrices.
```

```
>>> omc.getLinearOutputs() //returns a list of strings of names of outputs used when␣
↪forming matrices.
```

```
>>> omc.getLinearStates() // returns a list of strings of names of states used when␣
↪forming matrices.
```

CHAPTER

TWENTYEIGHT

OMJULIA - OPENMODELICA JULIA SCRIPTING

OMJulia - the OpenModelica Julia API is a free, open source, highly portable Julia based interactive session handler for Julia scripting of OpenModelica API functionality. It provides the modeler with components for creating a complete Julia-Modelica modeling, compilation and simulation environment based on the latest OpenModelica implementation and Modelica library standard available. OMJulia is architectured to combine both the solving strategy and model building. Thus, 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. OMJulia is not a standalone package, it depends upon the OpenModelica installation.

OMJulia.jl is implemented in Julia and depends on ZeroMQ - high performance asynchronous messaging library.

The latest OMJulia.jl documentation can be found online: OMJulia.jl documentation

CHAPTER

TWENTYNINE

JUPYTER-OPENMODELICA

An OpenModelica Kernel for Jupyter Notebook. All commands are interpreted by OMPython which communicates with OpenModelica Compiler and the results are presented to user.

The project is available at https://github.com/OpenModelica/jupyter-openmodelica.

Follow the Readme file to install and start running modelica models directly in Jupyter Notebook

CHAPTER

THIRTY

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.

30.1 OpenModelica Scripting Commands

The following are brief descriptions of the scripting commands available in the OpenModelica environment. See also https://build.openmodelica.org/Documentation/OpenModelica.Scripting.html. All commands are shown in alphabetical order:

30.1.1 interactiveDumpAbsynToJL

Dumps the AST into a Julia representation.

```
function interactiveDumpAbsynToJL
  output String res;
end interactiveDumpAbsynToJL;
```

30.1.2 relocateFunctions

Update symbols in the running program to ones defined in the given shared object.

```
function relocateFunctions
  input String fileName;
  input String names[:, 2];
  output Boolean success;
end relocateFunctions;
```

30.1.3 toJulia

Translates Absyn to Julia.

```
function toJulia
  output String res;
end toJulia;
```

30.1.4 GC_expand_hp

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

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

30.1.5 GC_gcollect_and_unmap

Forces the GC to collect and unmap memory.

30.1.6 GC_get_prof_stats

Returns a record with the GC statistics.

```
function GC_get_prof_stats
  output GC_PROFSTATS gcStats;
end GC_get_prof_stats;
```

30.1.7 GC_set_max_heap_size

Forces the GC to limit the maximum heap size.

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

30.1.8 addClassAnnotation

Adds an annotation to a class.

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

30.1.9 addComponent

Adds a component to the given class.

```
function addComponent
  input TypeName componentName;
  input TypeName typeName;
  input TypeName classPath;
  input Expression binding = $Code(());
  input ExpressionOrModification modification = $Code();
  input Expression comment = $Code(());
  input Expression annotate = $Code(());
```

(continues on next page)

(continued from previous page)

```
  output Boolean success;
end addComponent;
```

30.1.10 addConnection

Adds a connection to the given class.

```
function addConnection
  input VariableName connector1;
  input VariableName connector2;
  input TypeName className;
  input Expression comment = $Code(());
  input Expression annotate = $Code(());
  output Boolean success;
end addConnection;
```

30.1.11 addEquation

Adds an equation to a class.

```
function addEquation
  input TypeName className "The name of the class to add the equation to";
  input String eq "The equation given as a string";
  input Boolean isInitial = false "Whether the equation should be added as an initial␣
↪or normal equation";
  output Boolean success "true if the equation could be added, otherwise false";
end addEquation;
```

30.1.12 addInitialState

Adds an initial state to a class.

```
function addInitialState
  input TypeName cl;
  input String state;
  input ExpressionOrModification annotate;
  output Boolean bool;
end addInitialState;
```

30.1.13 addTransition

Adds a transition to a class.

```
function addTransition
  input TypeName cl;
  input String from;
  input String to;
  input String condition;
  input Boolean immediate = true;
  input Boolean reset = true;
  input Boolean synchronize = false;
  input Integer priority = 1;
  input ExpressionOrModification annotate;
  output Boolean bool;
end addTransition;
```

30.1.14 alarm

Schedules an alarm signal for the process.

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

30.1.15 appendEnvironmentVar

Appends a variable to the environment variables list.

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

30.1.16 basename

Returns the base name (file part) of a file path.

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

30.1.17 buildEncryptedPackage

Builds an encrypted package for a class.

```
function buildEncryptedPackage
  input TypeName className "the class that should encrypted";
  input Boolean encrypt = true;
  output Boolean success;
end buildEncryptedPackage;
```

30.1.18 buildLabel

Calls buildModel with the --generateLabeledSimCode flag enabled.

```
function buildLabel
  input TypeName className "the class that should be built";
  input Real startTime = 0.0 "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "dassl" "integration method used for simulation. <default> =␣
↪dassl";
  input String fileNamePrefix = "" "fileNamePrefix. <default> = \"\"";
  input String options = "" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
```

(continues on next page)

(continued from previous page)

```
  input String cflags = "" "cflags. <default> = \"\"";
  input String simflags = "" "simflags. <default> = \"\"";
  output String[2] buildModelResults;
end buildLabel;
```

30.1.19 buildModel

Translates a Modelica model into C code and builds a simulation executable.

```
function buildModel
  input TypeName className "the class that should be built";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.
↪0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "<default>" "integration method used for simulation. <default>
↪ = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String[2] buildModelResults;
end buildModel;
```

30.1.20 buildModelFMU

Translates a Modelica model into a Functional Mockup Unit.

```
function buildModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation), me_
↪cs (both model exchange and co-simulation)";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"className\"
↪";
  input String platforms[:] = {"static"} "The list of platforms to generate code for.
                                          \"dynamic\"=current platform, dynamically␣
↪link the runtime.
                                          \"static\"=current platform, statically␣
↪link everything.
                                          \"<cpu>-<vendor>-<os>\", host tripple, e.
↪g. \"x86_64-linux-gnu\" or \"x86_64-w64-mingw32\".
                                          \"<cpu>-<vendor>-<os> docker run ghcr.io/
↪openmodelica/crossbuild:v1.27.0\" host triple with OpenModelica supplied Docker␣
↪image, e.g. \"x86_64-linux-gnu docker run ghcr.io/openmodelica/crossbuild:v1.27.0\".
                                          \"<cpu>-<vendor>-<os> docker run <image>\
↪" host triple with Docker image, e.g. \"x86_64-linux-gnu docker run --pull=never␣
↪multiarch/crossbuild\"";
  input Boolean includeResources = false "Depreacted and no effect";
  input String method = "<default>" "integration method embedded in a Co-Simulation␣
```

(continues on next page)

(continued from previous page)

```
↪FMU. <default> = dassl";
  output String generatedFileName "Returns the full path of the generated FMU.";
end buildModelFMU;
```

30.1.21 cd

Changes the working directory.

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

30.1.22 checkAllModelsRecursive

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

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

30.1.23 checkCodeGraph

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

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

30.1.24 checkInterfaceOfPackages

Checks the interfaces of MetaModelica packages.

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

30.1.25 checkModel

Checks a model and returns the number of variables and equations.

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

30.1.26 checkSettings

Display some diagnostics.

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

30.1.27 checkTaskGraph

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

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

30.1.28 classAnnotationExists

Checks if an annotation exists in a class.

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

30.1.29 clear

Clears loaded classes and user defined variables.

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

30.1.30 clearCommandLineOptions

Resets all command line options to their default values.

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

30.1.31 clearDebugFlags

Resets all debug flags to their default values.

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

30.1.32 clearMessages

Clears the error buffer.

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

30.1.33 clearProgram

Clears loaded classes.

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

30.1.34 clearVariables

Clears all user defined variables.

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

30.1.35 closeSimulationResultFile

Closes the current simulation results file.

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

30.1.36 codeToString

Converts a $Code expression to a string.

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

30.1.37 compareFiles

Checks if two files are equal or not.

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

30.1.38 compareFilesAndMove

Overwrites a file with another if they differ.

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

30.1.39 convertPackageToLibrary

Runs the conversion script for a library on a selected package.

```
function convertPackageToLibrary
  input TypeName packageToConvert;
  input TypeName library;
  input String libraryVersion;
  output Boolean success;
end convertPackageToLibrary;
```

30.1.40 convertUnits

Gets conversion factors for two units.

```
function convertUnits
  input String s1;
  input String s2;
  output Boolean unitsCompatible;
  output Real scaleFactor;
  output Real offset;
end convertUnits;
```

30.1.41 copy

Copies a file.

```
function copy
  input String source;
  input String destination;
  output Boolean success;
end copy;
```

30.1.42 copyClass

Copies a class within the same level.

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

30.1.43 countMessages

Returns the number of buffered messages.

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

30.1.44 createModel

Creates a new empty model.

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

30.1.45 deleteClass

Unloads a class.

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

30.1.46 deleteComponent

Deletes a component from the given class.

```
function deleteComponent
  input TypeName componentName;
  input TypeName classPath;
  output Boolean success;
end deleteComponent;
```

30.1.47 deleteConnection

Deletes a connection in the given class.

```
function deleteConnection
  input VariableName connector1;
  input VariableName connector2;
  input TypeName className;
  output Boolean success;
end deleteConnection;
```

30.1.48 deleteEquation

Deletes an equation in a class.

30.1.49 deleteFile

Deletes a file with the given name.

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

30.1.50 deleteInitialState

Deletes an initial state in a class.

```
function deleteInitialState
  input TypeName cl;
  input String state;
  output Boolean bool;
end deleteInitialState;
```

30.1.51 deleteTransition

Deletes a transition from a class.

```
function deleteTransition
  input TypeName cl;
  input String from;
  input String to;
  input String condition;
  input Boolean immediate;
  input Boolean reset;
  input Boolean synchronize;
  input Integer priority;
  output Boolean bool;
end deleteTransition;
```

30.1.52 deltaSimulationResults

Calculates the sum of absolute errors.

```
function deltaSimulationResults
  input String filename;
  input String reffilename;
  input String method "method to compute then error. choose 1norm, 2norm, maxerr";
  input String[:] vars = fill("", 0);
  output Real result;
end deltaSimulationResults;
```

30.1.53 diffModelicaFileListings

Creates diffs of two strings corresponding to Modelica files

```
function diffModelicaFileListings
  input String before;
  input String after;
  input DiffFormat diffFormat = DiffFormat.color;
  input Boolean failOnSemanticsChange = false "Defaults to returning after instead of␣
↪hard fail";
  output String result;
end diffModelicaFileListings;
```

30.1.54 diffSimulationResults

Compares simulation results.

```
function diffSimulationResults
  input String actualFile;
  input String expectedFile;
  input String diffPrefix;
  input Real relTol = 1e-3 "y tolerance";
```

(continues on next page)

(continued from previous page)

```
  input Real relTolDiffMinMax = 1e-4 "y tolerance based on the difference between the␣
↪maximum and minimum of the signal";
  input Real rangeDelta = 0.002 "x tolerance";
  input String[:] vars = fill("", 0);
  input Boolean keepEqualResults = false;
  output Boolean success;
  output String[:] failVars;
end diffSimulationResults;
```

30.1.55 diffSimulationResultsHtml

Compares simulation results and generates an HTML report.

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

30.1.56 directoryExists

Returns whether the given directory exists or not.

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

30.1.57 dirname

Returns the directory name of a file path.

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

30.1.58 disableNewInstantiation

Disables the new (default) instantiation.

```
function disableNewInstantiation
  output Boolean success;
end disableNewInstantiation;
```

30.1.59 dumpXMLDAE

Outputs the DAE system corresponding to a specific model.

```
function dumpXMLDAE
  input TypeName className;
  input String translationLevel = "flat" "flat, optimiser, backEnd, or stateSpace";
  input Boolean addOriginalAdjacencyMatrix = false;
  input Boolean addSolvingInfo = false;
  input Boolean addMathMLCode = false;
  input Boolean dumpResiduals = false;
  input String fileNamePrefix = "<default>" "this is the className in string form by␣
↪default";
  input String rewriteRulesFile = "" "the file from where the rewiteRules are read,␣
↪default is empty which means no rewrite rules";
  output Boolean success "if the function succeeded true/false";
  output String xmlfileName "the Xml file";
end dumpXMLDAE;
```

30.1.60 echo

Turns interactive output on or off.

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

30.1.61 enableNewInstantiation

Enables the new (default) instantiation.

```
function enableNewInstantiation
  output Boolean success;
end enableNewInstantiation;
```

30.1.62 escapeXML

Replaces characters in a string with XML escape characters.

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

30.1.63 existClass

Returns whether the given class exists or not.

```
function existClass
  input TypeName cl;
  output Boolean exists;
end existClass;
```

30.1.64 existModel

Returns whether the given model exists or not.

30.1.65 existPackage

Returns whether the given package exists or not.

30.1.66 exit

Forces omc to quit with the given exit status.

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

30.1.67 exportToFigaro

Exports a model to a Figaro database.

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

30.1.68 extendsFrom

Returns true if the given class extends from the given base class.

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

30.1.69 filterSimulationResults

Creates a simulation results file with selected variables.

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

30.1.70 generateCode

Generates code for a function.

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

30.1.71 generateEntryPoint

Generates an entry point for a MetaModelica program.

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

30.1.72 generateHeader

Generates header file for external MetaModelica functions.

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

30.1.73 generateJuliaHeader

Generates a Julia header file for external MetaModelica functions.

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

30.1.74 generateScriptingAPI

Generates the scripting API.

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

30.1.75 generateSeparateCode

Generates code for a MetaModelica package.

```
function generateSeparateCode
  input TypeName className;
  input Boolean cleanCache = false "If true, the cache is reset between each␣
↪generated package. This conserves memory at the cost of speed.";
  output Boolean success;
end generateSeparateCode;
```

30.1.76 generateSeparateCodeDependencies

Generates dependencies for a MetaModelica package.

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

30.1.77 generateSeparateCodeDependenciesMakefile

Generates dependencies Makefile for a MetaModelica package.

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

30.1.78 generateVerificationScenarios

Generate scenarios for a verification model.

```
function generateVerificationScenarios
  input TypeName path;
  output Boolean success;
end generateVerificationScenarios;
```

30.1.79 getAlgorithmCount

Counts the number of algorithm sections in a class.

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

30.1.80 getAlgorithmItemsCount

Counts the number of algorithm statements in a class.

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

30.1.81 getAllSubtypeOf

Returns the list of all classes that extend from className given a parentClass where the lookup for className should start

```
function getAllSubtypeOf
  input TypeName className;
  input TypeName parentClass = $Code(AllLoadedClasses);
  input Boolean qualified = false "Not implemented";
  input Boolean includePartial = false;
  input Boolean sort = false;
  output TypeName classNames[:];
end getAllSubtypeOf;
```

30.1.82 getAnnotationCount

Counts the number of annotation sections in a class.

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

30.1.83 getAnnotationModifierValue

Returns the value for a modifier in the given annotation.

```
function getAnnotationModifierValue
  input TypeName className;
  input String annotationName;
  input String modifierName;
  output String modifierValue;
end getAnnotationModifierValue;
```

30.1.84 getAnnotationNamedModifiers

Returns the names of the modifiers in the given annotation.

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

30.1.85 getAnnotationVersion

Returns the current annotation version.

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

30.1.86 getAvailableIndexReductionMethods

Returns the currently available index reduction methods.

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

30.1.87 getAvailableLibraries

Returns a list of all available libraries.

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

30.1.88 getAvailableLibraryVersions

Returns the installed versions of a library.

```
function getAvailableLibraryVersions
  input TypeName libraryName;
  output String[:] librariesAndVersions;
end getAvailableLibraryVersions;
```

30.1.89 getAvailableMatchingAlgorithms

Returns the available matching algorithms.

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

30.1.90 getAvailablePackageConversionsFrom

Returns the versions that provide conversion from the requested version of the library.

```
function getAvailablePackageConversionsFrom
  input TypeName pkg;
  input String version;
  output String[:] convertsTo;
end getAvailablePackageConversionsFrom;
```

30.1.91 getAvailablePackageConversionsTo

Returns the versions that provide conversion to the requested version of the library.

```
function getAvailablePackageConversionsTo
  input TypeName pkg;
  input String version;
  output String[:] convertsTo;
end getAvailablePackageConversionsTo;
```

30.1.92 getAvailablePackageVersions

Returns the versions that provide the requested version of the library.

```
function getAvailablePackageVersions
  input TypeName pkg;
  input String version;
  output String[:] withoutConversion;
end getAvailablePackageVersions;
```

30.1.93 getAvailableTearingMethods

Returns the available tearing methods.

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

30.1.94 getBooleanClassAnnotation

Checks if an annotation exists and returns its value

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

30.1.95 getBuiltinType

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

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

30.1.96 getCFlags

Returns the C compiler flags (CFLAGS) used for simulation code.

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

30.1.97 getCXXCompiler

Returns the C++ compiler (CXX) used for simulation code.

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

30.1.98 getClassComment

Returns a class's comment.

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

30.1.99 getClassInformation

Returns information about a class.

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

(continues on next page)

(continued from previous page)

```
  output Boolean isDocumentationClass;
  output String version;
  output String preferredView;
  output Boolean state;
  output String access;
  output String versionDate;
  output String versionBuild;
  output String dateModified;
  output String revisionId;
end getClassInformation;
```

30.1.100 getClassNames

Returns the list of class names defined in the class.

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

30.1.101 getClassRestriction

Returns the restriction of the given class.

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

30.1.102 getCommandLineOptions

Returns all command line options who have non-default values as a list of strings.

```
function getCommandLineOptions
  output String[:] flags;
end getCommandLineOptions;
```

30.1.103 getCompiler

Returns the C compiler (CC) used for simulation code.

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

30.1.104 getComponentAnnotations

Returns the annotations of the components in the given class.

```
function getComponentAnnotations
  input TypeName className;
  output Expression result;
end getComponentAnnotations;
```

30.1.105 getComponentComment

Returns the comment on a component.

```
function getComponentComment
  input TypeName className;
  input TypeName componentName;
  output String comment;
end getComponentComment;
```

30.1.106 getComponentCount

Returns the number of components in a class.

```
function getComponentCount
  input TypeName classPath;
  output Integer count;
end getComponentCount;
```

30.1.107 getComponentModifierNames

Returns the list of class component modifiers.

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

30.1.108 getComponentModifierValue

Returns the binding equation of a component.

```
function getComponentModifierValue
  input TypeName class_;
  input TypeName modifier;
  output String value;
end getComponentModifierValue;
```

30.1.109 getComponentModifierValues

Returns the modifier for a component.

```
function getComponentModifierValues
  input TypeName class_;
  input TypeName modifier;
  output String value;
end getComponentModifierValues;
```

30.1.110 getComponents

Returns information about the component in a given class.

```
function getComponents
  input TypeName className;
  input Boolean useQuotes = false;
end getComponents;
```

30.1.111 getComponentsTest

Returns an array of records with information about the components of the given class.

```
function getComponentsTest
  input TypeName name;
  output Component[:] components;
  record Component
    String className "the type of the component";
    String name "the name of the component";
    String comment "the comment of the component";
    Boolean isProtected "true if component is protected";
    Boolean isFinal "true if component is final";
    Boolean isFlow "true if component is flow";
    Boolean isStream "true if component is stream";
    Boolean isReplaceable "true if component is replaceable";
    String variability "'constant', 'parameter', 'discrete', ''";
    String innerOuter "'inner', 'outer', ''";
    String inputOutput "'input', 'output', ''";
    String dimensions[:] "array with the dimensions of the component";
  end Component;
end getComponentsTest;
```

30.1.112 getConfigFlagValidOptions

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

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

30.1.113 getConnectionCount

Counts the number of connect equation in a class.

```
function getConnectionCount
  input TypeName className;
  output Integer count;
end getConnectionCount;
```

30.1.114 getConnectionList

Returns an list of all connect equations including those within loops

```
function getConnectionList
  input TypeName className;
  output String[:, :] result;
end getConnectionList;
```

30.1.115 getConnectorCount

Returns the number of public connectors in the given class.

```
function getConnectorCount
  input TypeName className;
  output Integer count;
end getConnectorCount;
```

30.1.116 getConversionsFromVersions

Returns the versions this library can convert from with and without conversions.

```
function getConversionsFromVersions
  input TypeName pack;
  output String[:] withoutConversion;
  output String[:] withConversion;
end getConversionsFromVersions;
```

30.1.117 getCrefInfo

Deprecated; use getClassInformation instead.

```
function getCrefInfo
  input TypeName cl;
  output Expression[:] result;
end getCrefInfo;
```

30.1.118 getDefaultComponentName

Returns the default component name for a class.

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

30.1.119 getDefaultComponentPrefixes

Returns the default component prefixes for a class.

```
function getDefaultComponentPrefixes
  input TypeName cl;
  output String prefixes;
end getDefaultComponentPrefixes;
```

30.1.120 getDefaultOpenCLDevice

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

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

30.1.121 getDefinitions

Dumps the defined packages, classes, and optionally functions to a string.

```
function getDefinitions
  input Boolean addFunctions;
  output String result;
end getDefinitions;
```

30.1.122 getDerivedClassModifierNames

Returns a derived class's modifier names.

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

30.1.123 getDerivedClassModifierValue

Returns a derived class's modifier value.

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

30.1.124 getDerivedUnits

Returns the list of derived units for the specified base unit.

```
function getDerivedUnits
  input String baseUnit;
  output String[:] derivedUnits;
end getDerivedUnits;
```

30.1.125 getDiagramAnnotation

Returns the Diagram annotation for a given class.

```
function getDiagramAnnotation
  input TypeName className;
  output Expression result;
end getDiagramAnnotation;
```

30.1.126 getDocumentationAnnotation

Returns the Documentation annotation defined in the class.

```
function getDocumentationAnnotation
  input TypeName cl;
  output String out[3] "{info,revision,infoHeader}";
end getDocumentationAnnotation;
```

30.1.127 getElementAnnotation

Returns the annotation on a component or class element as a string.

```
function getElementAnnotation
  input TypeName elementName;
  output String annotationString;
end getElementAnnotation;
```

30.1.128 getElementAnnotations

Returns the annotations of the components and short class definitions in the given class.

```
function getElementAnnotations
  input TypeName className;
  output Expression result;
end getElementAnnotations;
```

30.1.129 getElementModifierNames

Returns the list of element (component or short class) modifiers in a class.

```
function getElementModifierNames
  input TypeName className;
  input String elementName;
  output String[:] modifiers;
end getElementModifierNames;
```

30.1.130 getElementModifierValue

Returns the binding equation for an element.

```
function getElementModifierValue
  input TypeName className;
  input TypeName modifier;
  output String value;
end getElementModifierValue;
```

30.1.131 getElementModifierValues

Returns the modifier for an element.

```
function getElementModifierValues
  input TypeName className;
  input TypeName modifier;
  output String value;
end getElementModifierValues;
```

30.1.132 getElements

Returns information about the elements in a given class.

```
function getElements
  input TypeName className;
  input Boolean useQuotes = false;
end getElements;
```

30.1.133 getElementsInfo

Returns information about the elements in a given class.

```
function getElementsInfo
  input TypeName className;
  output Expression result;
end getElementsInfo;
```

30.1.134 getEnumerationLiterals

Returns the literals for a given enumeration type.

```
function getEnumerationLiterals
  input TypeName className;
  output String[:] result;
end getEnumerationLiterals;
```

30.1.135 getEnvironmentVar

Returns the value of the given environment variable.

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

30.1.136 getEquationCount

Counts the number of equation sections in a class.

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

30.1.137 getEquationItemsCount

Counts the number of equations in a class.

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

30.1.138 getErrorString

Returns current error messages.

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

30.1.139 getExtendsModifierNames

Returns the names of the modifiers on an extends clause.

```
function getExtendsModifierNames
  input TypeName className;
  input TypeName extendsName;
  input Boolean useQuotes = false;
  output String modifiers;
end getExtendsModifierNames;
```

30.1.140 getExtendsModifierValue

Returns the modifier value for a modifier on an extends clause.

```
function getExtendsModifierValue
  input TypeName className;
  input TypeName extendsName;
  input TypeName modifierName;
  output Expression result;
end getExtendsModifierValue;
```

30.1.141 getExternalFunctionSpecification

Returns information about a function's external specification.

```
function getExternalFunctionSpecification
  input TypeName functionName;
  output Expression result;
end getExternalFunctionSpecification;
```

30.1.142 getHomeDirectoryPath

Returns the path to the current user's HOME directory.

```
function getHomeDirectoryPath
  output String homeDirectoryPath;
end getHomeDirectoryPath;
```

30.1.143 getIconAnnotation

Returns the Icon annotation for a given class.

```
function getIconAnnotation
  input TypeName className;
  output Expression result;
end getIconAnnotation;
```

30.1.144 getImportCount

Counts the number of import-clauses in a class.

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

30.1.145 getImportedNames

Returns the definition names of all import-clauses in a class.

```
function getImportedNames
  input TypeName class_;
  output String[:] out_public;
  output String[:] out_protected;
end getImportedNames;
```

30.1.146 getIndexReductionMethod

Returns the currently used index reduction method.

```
function getIndexReductionMethod
  output String selected;
end getIndexReductionMethod;
```

30.1.147 getInheritanceCount

Returns the numbers of extends clauses in the given class.

```
function getInheritanceCount
  input TypeName className;
  output Integer count;
end getInheritanceCount;
```

30.1.148 getInheritedClasses

Returns the list of inherited classes in a class.

```
function getInheritedClasses
  input TypeName name;
  output TypeName inheritedClasses[:];
end getInheritedClasses;
```

30.1.149 getInitialAlgorithmCount

Counts the number of initial algorithm sections in a class.

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

30.1.150 getInitialAlgorithmItemsCount

Counts the number of initial algorithm statements in a class.

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

30.1.151 getInitialEquationCount

Counts the number of initial equation sections in a class.

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

30.1.152 getInitialEquationItemsCount

Counts the number of initial equations in a class.

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

30.1.153 getInitialStates

Returns a list of initial states in a class.

```
function getInitialStates
  input TypeName cl;
  output String[:, :] initialStates;
end getInitialStates;
```

30.1.154 getInstallationDirectoryPath

Returns the installation directory path.

```
function getInstallationDirectoryPath
  output String installationDirectoryPath;
end getInstallationDirectoryPath;
```

30.1.155 getInstantiatedParametersAndValues

Returns the top-level parameter names and values from the DAE.

```
function getInstantiatedParametersAndValues
  input TypeName cls;
  output String[:] values;
end getInstantiatedParametersAndValues;
```

30.1.156 getLanguageStandard

Returns the current Modelica Language Standard in use.

```
function getLanguageStandard
  output String outVersion;
end getLanguageStandard;
```

30.1.157 getLinker

Returns the linker (LINK) used for simulation code.

```
function getLinker
  output String linker;
end getLinker;
```

30.1.158 getLinkerFlags

Returns the linker flags (LDFLAGS) used for simulation code.

```
function getLinkerFlags
  output String linkerFlags;
end getLinkerFlags;
```

30.1.159 getLoadedLibraries

Returns the loaded libraries.

```
function getLoadedLibraries
  output String[:, 2] libraries;
end getLoadedLibraries;
```

30.1.160 getMMfileTotalDependencies

Returns imports for a MetaModelica package.

```
function getMMfileTotalDependencies
  input String in_package_name;
  input String public_imports_dir;
  output String[:] total_pub_imports;
end getMMfileTotalDependencies;
```

30.1.161 getMatchingAlgorithm

Returns the currently used matching algorithm.

```
function getMatchingAlgorithm
  output String selected;
end getMatchingAlgorithm;
```

30.1.162 getMemorySize

Returns the amount of system memory.

```
function getMemorySize
  output Real memory(unit = "MiB");
end getMemorySize;
```

30.1.163 getModelFigures

Returns the figures defined in the class' figures sub-annotation of Documentation.

```
function getModelFigures
  input TypeName name;
  output Figure[:] figures;
end getModelFigures;
```

30.1.164 getModelInstance

Dumps a model instance as a JSON string.

```
function getModelInstance
  input TypeName className;
  input TypeName context = $Code(__NoContext);
  input String modifier = "";
  input Boolean prettyPrint = false;
  output String result;
end getModelInstance;
```

30.1.165 getModelInstanceAnnotation

Dumps the annotation of a model using the same JSON format as getModelInstance.

```
function getModelInstanceAnnotation
  input TypeName className;
  input String[:] filter = fill("", 0);
  input Boolean prettyPrint = false;
  output String result;
end getModelInstanceAnnotation;
```

30.1.166 getModelInstanceAnnotationReference

Like getModelInstanceAnnotation, but returns an integer handle to an in-memory structure instead of a JSON string (see getModelInstanceReference).

```
function getModelInstanceAnnotationReference
  input TypeName className;
  input String[:] filter = fill("", 0);
  output Integer handle;
end getModelInstanceAnnotationReference;
```

30.1.167 getModelInstanceReference

Like getModelInstance, but returns an integer handle to an in-memory model instance structure instead of a JSON string (see issue #15219). The handle is intended to be read directly by an in-process client such as OMEdit and must be released with releaseModelInstanceReference. Returns 0 on failure.

```
function getModelInstanceReference
  input TypeName className;
  input TypeName context = $Code(__NoContext);
  input String modifier = "";
  output Integer handle;
end getModelInstanceReference;
```

30.1.168 getModelicaPath

Returns the Modelica library path.

```
function getModelicaPath
  output String modelicaPath;
end getModelicaPath;
```

30.1.169 getNamedAnnotation

Returns the value of the annotation with the given name in the given class.

```
function getNamedAnnotation
  input TypeName className;
  input TypeName annotationName;
  output Expression result;
end getNamedAnnotation;
```

30.1.170 getNoSimplify

Returns true if noSimplify flag is set.

```
function getNoSimplify
  output Boolean noSimplify;
end getNoSimplify;
```

30.1.171 getNthAlgorithm

Returns the n:th algorithm section in a class.

```
function getNthAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithm;
```

30.1.172 getNthAlgorithmItem

Returns the n:th algorithm statement in a class.

```
function getNthAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithmItem;
```

30.1.173 getNthAnnotationString

Returns the n:th annotation section as string.

```
function getNthAnnotationString
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAnnotationString;
```

30.1.174 getNthComponent

Returns the type, name, and description string of the n:th component in the given class.

```
function getNthComponent
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthComponent;
```

30.1.175 getNthComponentAnnotation

Returns the annotation for the n:th component in the given class.

```
function getNthComponentAnnotation
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthComponentAnnotation;
```

30.1.176 getNthComponentCondition

Returns the condition for the n:th component in the given class as a string.

```
function getNthComponentCondition
  input TypeName className;
  input Integer n;
  output String result;
end getNthComponentCondition;
```

30.1.177 getNthComponentModification

Returns the modification for the n:th component in the given class.

```
function getNthComponentModification
  input TypeName className;
  input Integer n;
  output ExpressionOrModification result[:];
end getNthComponentModification;
```

30.1.178 getNthConnection

Returns the n:th connection.

```
function getNthConnection
  input TypeName className;
  input Integer index;
  output String[:] result;
end getNthConnection;
```

30.1.179 getNthConnectionAnnotation

Returns the annotation of the n:th connect clause in the class.

```
function getNthConnectionAnnotation
  input TypeName className;
  input Integer index;
```

(continues on next page)

(continued from previous page)

```
  output Expression result;
end getNthConnectionAnnotation;
```

30.1.180 getNthConnector

Returns the name and type of the n:th public connector in the given class.

```
function getNthConnector
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthConnector;
```

30.1.181 getNthConnectorIconAnnotation

Returns the Icon annotation from the type of the n:th public connector in the given class.

```
function getNthConnectorIconAnnotation
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthConnectorIconAnnotation;
```

30.1.182 getNthEquation

Returns the n:th equation section in a class.

```
function getNthEquation
  input TypeName class_;
  input Integer index;
  output String result;
end getNthEquation;
```

30.1.183 getNthEquationItem

Returns the n:th equation in a class.

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

30.1.184 getNthImport

Returns the n:th import-clause.

```
function getNthImport
  input TypeName class_;
  input Integer index;
  output String out[3] "{\"Path\",\"Id\",\"Kind\"}";
end getNthImport;
```

30.1.185 getNthInheritedClass

Returns the name of the n:th inherited class in the given class.

```
function getNthInheritedClass
  input TypeName className;
  input Integer n;
  output TypeName baseClass;
end getNthInheritedClass;
```

30.1.186 getNthInheritedClassDiagramMapAnnotation

Returns the IconMap annotation for the n:th inherited class in the given class.

```
function getNthInheritedClassDiagramMapAnnotation
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthInheritedClassDiagramMapAnnotation;
```

30.1.187 getNthInheritedClassIconMapAnnotation

Returns the IconMap annotation for the n:th inherited class in the given class.

```
function getNthInheritedClassIconMapAnnotation
  input TypeName className;
  input Integer n;
  output Expression result;
end getNthInheritedClassIconMapAnnotation;
```

30.1.188 getNthInitialAlgorithm

Returns the n:th initial algorithm section in a class.

```
function getNthInitialAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithm;
```

30.1.189 getNthInitialAlgorithmItem

Returns the n:th initial algorithm statement in a class.

```
function getNthInitialAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithmItem;
```

30.1.190 getNthInitialEquation

Returns the n:th initial equation section in a class.

```
function getNthInitialEquation
  input TypeName class_;
  input Integer index;
```

(continues on next page)

(continued from previous page)

```
  output String result;
end getNthInitialEquation;
```

30.1.191 getNthInitialEquationItem

Returns the n:th initial equation in a class.

```
function getNthInitialEquationItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialEquationItem;
```

30.1.192 getOrderConnections

Returns true if orderConnections flag is set.

```
function getOrderConnections
  output Boolean orderConnections;
end getOrderConnections;
```

30.1.193 getPackages

Returns the list of packages defined in the class.

```
function getPackages
  input TypeName class_ = $Code(AllLoadedClasses);
  output TypeName classNames[:];
end getPackages;
```

30.1.194 getParameterNames

Returns the names of all parameters in a given class.

```
function getParameterNames
  input TypeName class_;
  output String[:] parameters;
end getParameterNames;
```

30.1.195 getParameterValue

Returns the value of a parameter of the class.

```
function getParameterValue
  input TypeName class_;
  input String parameterName;
  output String parameterValue;
end getParameterValue;
```

30.1.196 getReplaceableChoices

Returns all replaceable choices for a class with choicesAllMatching.

```
function getReplaceableChoices
  input TypeName baseClass;
```

(continues on next page)

(continued from previous page)

```
  input TypeName parentClass;
  input Boolean includePartial = false;
  input Boolean sort = false;
  output String choices[:, :];
end getReplaceableChoices;
```

30.1.197 getSettings

Returns some settings.

```
function getSettings
  output String settings;
end getSettings;
```

30.1.198 getShortDefinitionBaseClassInformation

Returns information about a short class definition.

```
function getShortDefinitionBaseClassInformation
  input TypeName className;
  output Expression result;
end getShortDefinitionBaseClassInformation;
```

30.1.199 getShowAnnotations

Returns the value of the --showAnnotations flag.

```
function getShowAnnotations
  output Boolean show;
end getShowAnnotations;
```

30.1.200 getSimulationOptions

Returns the startTime, stopTime, tolerance, and interval based on the experiment annotation.

```
function getSimulationOptions
  input TypeName name;
  input Real defaultStartTime = 0.0;
  input Real defaultStopTime = 1.0;
  input Real defaultTolerance = 1e-6;
  input Integer defaultNumberOfIntervals = 500 "May be overridden by defining␣
↪defaultInterval instead";
  input Real defaultInterval = 0.0 "If = 0.0, then numberOfIntervals is used to␣
↪calculate the step size";
  output Real startTime;
  output Real stopTime;
  output Real tolerance;
  output Integer numberOfIntervals;
  output Real interval;
end getSimulationOptions;
```

30.1.201 getSourceFile

Returns the filename of the class.

```
function getSourceFile
  input TypeName class_;
  output String filename "empty on failure";
end getSourceFile;
```

30.1.202 getTearingMethod

Returns the currently used tearing method.

```
function getTearingMethod
  output String selected;
end getTearingMethod;
```

30.1.203 getTempDirectoryPath

Returns the current user temporary directory location.

```
function getTempDirectoryPath
  output String tempDirectoryPath;
end getTempDirectoryPath;
```

30.1.204 getTimeStamp

Returns the timestamp for a class.

```
function getTimeStamp
  input TypeName cl;
  output Real timeStamp;
  output String timeStampAsString;
end getTimeStamp;
```

30.1.205 getTotalModel

Saves a model and dependencies to a single string.

```
function getTotalModel
  input TypeName className;
  input Boolean stripAnnotations = false;
  input Boolean stripComments = false;
  input Boolean obfuscate = false;
  output String result;
end getTotalModel;
```

30.1.206 getTransitions

Returns list of transitions for the given class.

```
function getTransitions
  input TypeName cl;
  output String[:, :] transitions;
end getTransitions;
```

30.1.207 getUsedClassNames

Returns the list of class names used in the total program defined by the class.

```
function getUsedClassNames
  input TypeName className;
  output TypeName classNames[:];
end getUsedClassNames;
```

30.1.208 getUses

Returns the libraries used by a package.

```
function getUses
  input TypeName pack;
  output String[:, :] uses;
end getUses;
```

30.1.209 getVectorizationLimit

Returns the vectorization limit used by the old frontend.

```
function getVectorizationLimit
  output Integer vectorizationLimit;
end getVectorizationLimit;
```

30.1.210 getVersion

Returns the version of the compiler or a Modelica library.

```
function getVersion
  input TypeName cl = $Code(OpenModelica);
  output String version;
end getVersion;
```

30.1.211 help

Display the OpenModelica help text.

```
function help
  input String topic = "topics";
  output String helpText;
end help;
```

30.1.212 iconv

Converts a string from one character encoding to another.

```
function iconv
  input String string;
  input String from;
  input String to = "UTF-8";
  output String result;
end iconv;
```

30.1.213 importFMU

Imports a Functional Mockup Unit.

```
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.";
  input TypeName modelName = $Code(Default) "Name of the generated model. If default␣
↪then the name is auto generated using FMU information.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMU;
```

30.1.214 importFMUModelDescription

Imports modelDescription.xml

```
function importFMUModelDescription
  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 importFMUModelDescription;
```

30.1.215 inferBindings

Update bindings for a verification model.

```
function inferBindings
  input TypeName path;
  output Boolean success;
end inferBindings;
```

30.1.216 installPackage

Installs a package.

```
function installPackage
  input TypeName pkg;
  input String version = "";
```

(continues on next page)

(continued from previous page)

```
  input Boolean exactMatch = false;
  output Boolean result;
end installPackage;
```

30.1.217 instantiateModel

Instantiates a model and returns the flattened model.

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

30.1.218 isBlock

Checks if a given class is a block.

```
function isBlock
  input TypeName cl;
  output Boolean b;
end isBlock;
```

30.1.219 isClass

Checks if a given class is a class.

```
function isClass
  input TypeName cl;
  output Boolean b;
end isClass;
```

30.1.220 isConnector

Checks if a given class is a connector or expandable connector.

```
function isConnector
  input TypeName cl;
  output Boolean b;
end isConnector;
```

30.1.221 isConstant

Checks if a component in a class is a constant.

```
function isConstant
  input TypeName componentName;
  input TypeName className;
  output Boolean result;
end isConstant;
```

30.1.222 isEnumeration

Checks if a given class is an enumeration.

```
function isEnumeration
  input TypeName cl;
  output Boolean b;
end isEnumeration;
```

30.1.223 isExperiment

Checks if a class is an experiment.

```
function isExperiment
  input TypeName name;
  output Boolean res;
end isExperiment;
```

30.1.224 isExtendsModifierFinal

Returns whether a modifier on an extends clause is final or not.

```
function isExtendsModifierFinal
  input TypeName className;
  input TypeName extendsName;
  input TypeName modifierName;
  output Boolean isFinal;
end isExtendsModifierFinal;
```

30.1.225 isFunction

Checks if a given class is a function.

```
function isFunction
  input TypeName cl;
  output Boolean b;
end isFunction;
```

30.1.226 isModel

Checks if a given class is a model.

```
function isModel
  input TypeName cl;
  output Boolean b;
end isModel;
```

30.1.227 isOperator

Checks if a given class is an operator.

```
function isOperator
  input TypeName cl;
  output Boolean b;
end isOperator;
```

30.1.228 isOperatorFunction

Checks if a given class is an operator function.

```
function isOperatorFunction
  input TypeName cl;
  output Boolean b;
end isOperatorFunction;
```

30.1.229 isOperatorRecord

Checks if a given class is an operator record.

```
function isOperatorRecord
  input TypeName cl;
  output Boolean b;
end isOperatorRecord;
```

30.1.230 isOptimization

Checks if a given class is an optimization.

```
function isOptimization
  input TypeName cl;
  output Boolean b;
end isOptimization;
```

30.1.231 isPackage

Checks if a given class is a package.

```
function isPackage
  input TypeName cl;
  output Boolean b;
end isPackage;
```

30.1.232 isParameter

Checks whether a component in a class is a parameter.

```
function isParameter
  input TypeName componentName;
  input TypeName className;
  output Boolean result;
end isParameter;
```

30.1.233 isPartial

Checks if a given class is partial.

```
function isPartial
  input TypeName cl;
  output Boolean b;
end isPartial;
```

30.1.234 isPrimitive

Checks if a type is primitive.

```
function isPrimitive
  input TypeName className;
  output Boolean result;
end isPrimitive;
```

30.1.235 isProtected

Checks if a component in a class is protected.

```
function isProtected
  input TypeName componentName;
  input TypeName className;
  output Boolean result;
end isProtected;
```

30.1.236 isProtectedClass

Returns true if the given class c1 has class c2 as one of its protected class.

```
function isProtectedClass
  input TypeName cl;
  input String c2;
  output Boolean b;
end isProtectedClass;
```

30.1.237 isRecord

Checks if a given class is a record.

```
function isRecord
  input TypeName cl;
  output Boolean b;
end isRecord;
```

30.1.238 isRedeclare

Checks if a given element is a redeclare element.

```
function isRedeclare
  input TypeName element;
  output Boolean b;
end isRedeclare;
```

30.1.239 isReplaceable

Checks if a given element is replaceable.

```
function isReplaceable
  input TypeName element;
  output Boolean b;
end isReplaceable;
```

30.1.240 isShortDefinition

Returns true if the given class is defined as a short class.

```
function isShortDefinition
  input TypeName class_;
  output Boolean isShortCls;
end isShortDefinition;
```

30.1.241 isType

Checks if a given class is a type.

```
function isType
  input TypeName cl;
  output Boolean b;
end isType;
```

30.1.242 linearize

Creates a model with symbolic linearization matrices.

```
function linearize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.
↪0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file. <default>␣
↪= 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "<default>" "integration method used for simulation. <default>
↪ = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String linearizationResult;
end linearize;
```

30.1.243 list

Lists the contents of the given class, or all loaded classes.

```
function list
  input TypeName class_ = $Code(AllLoadedClasses);
  input Boolean interfaceOnly = false;
  input Boolean shortOnly = false "only short class definitions";
  input ExportKind exportKind = ExportKind.Absyn;
  output String contents;
end list;
```

30.1.244 listFile

Lists the contents of the file given by the class.

```
function listFile
  input TypeName class_;
  input Boolean nestedClasses = true;
  output String contents;
end listFile;
```

30.1.245 listVariables

Lists the names of the active variables in the scripting environment.

```
function listVariables
  output TypeName variables[:];
end listVariables;
```

30.1.246 loadClassContentString

Loads class elements from a string and inserts them into the given loaded class.

```
function loadClassContentString
  input String data;
  input TypeName className;
  input Integer offsetX = 0;
  input Integer offsetY = 0;
  output Boolean success;
end loadClassContentString;
```

30.1.247 loadEncryptedPackage

Loads an encrypted package.

```
function loadEncryptedPackage
  input String fileName;
  input String workdir = "<default>" "The output directory for imported encrypted␣
↪files. <default> will put the files to current working directory.";
  input Boolean skipUnzip = false "Skips the unzip of .mol if true. In that case we␣
↪expect the files are already extracted e.g., because of parseEncryptedPackage()␣
↪call.";
  input Boolean uses = true;
  input Boolean notify = true "Give a notification of the libraries and versions that␣
↪were loaded";
  input Boolean requireExactVersion = false "If the version is required to be exact,␣
↪if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not match it.";
  output Boolean success;
end loadEncryptedPackage;
```

30.1.248 loadFile

Loads a Modelica file (*.mo).

```
function loadFile
  input String fileName;
  input String encoding = "UTF-8";
  input Boolean uses = true;
```

(continues on next page)

(continued from previous page)

```
  input Boolean notify = true "Give a notification of the libraries and versions that␣
↪were loaded";
  input Boolean requireExactVersion = false "If the version is required to be exact,␣
↪if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not match it.";
  input Boolean allowWithin = true "Whether to allow the file to have a within-clause␣
↪other than 'within;'.";
  output Boolean success;
end loadFile;
```

30.1.249 loadFileInteractive

Loads a Modelica file and returns a list of all loaded top-level classes.

```
function loadFileInteractive
  input String filename;
  input String encoding = "UTF-8";
  input Boolean uses = true;
  input Boolean notify = true "Give a notification of the libraries and versions that␣
↪were loaded";
  input Boolean requireExactVersion = false "If the version is required to be exact,␣
↪if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not match it.";
  output TypeName names[:];
end loadFileInteractive;
```

30.1.250 loadFileInteractiveQualified

Loads a Modelica file and returns a list of the top-level classes that were loaded.

```
function loadFileInteractiveQualified
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end loadFileInteractiveQualified;
```

30.1.251 loadFiles

Loads Modelica files (*.mo).

```
function loadFiles
  input String[:] fileNames;
  input String encoding = "UTF-8";
  input Integer numThreads = OpenModelica.Scripting.numProcessors();
  input Boolean uses = true;
  input Boolean notify = true "Give a notification of the libraries and versions that␣
↪were loaded";
  input Boolean requireExactVersion = false "If the version is required to be exact,␣
↪if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not match it.";
  input Boolean allowWithin = true "Whether to allow the files to have a within-
↪clause other than 'within;'.";
  output Boolean success;
end loadFiles;
```

30.1.252 loadModel

Loads a Modelica library.

```
function loadModel
  input TypeName className;
  input String[:] priorityVersion = {"default"};
  input Boolean notify = false "Give a notification of the libraries and versions␣
↪that were loaded";
  input String languageStandard = "" "Override the set language standard. Parse with␣
↪the given setting, but do not change it permanently.";
  input Boolean requireExactVersion = false "If the version is required to be exact,␣
↪if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not match it.";
  output Boolean success;
end loadModel;
```

30.1.253 loadModelica3D

Loads Modelica3D.

```
function loadModelica3D
  input String version = "3.2.1";
  output Boolean status;
end loadModelica3D;
```

30.1.254 loadOMSimulator

Loads the OMSimulator DLL from the default path.

```
function loadOMSimulator
  output Integer status;
end loadOMSimulator;
```

30.1.255 loadString

Loads Modelica definitions from a string.

```
function loadString
  input String data;
  input String filename = "<interactive>";
  input String encoding = "UTF-8" "Deprecated as *ALL* strings are now UTF-8 encoded";
  input Boolean merge = false "if merge is true the parsed AST is merged with the␣
↪existing AST,
                               default to false which means that is replaced, not␣
↪merged";
  input Boolean uses = true;
  input Boolean notify = true "Give a notification of the libraries and versions that␣
↪were loaded";
  input Boolean requireExactVersion = false "If the version is required to be exact,
                                             if there is a uses Modelica(version=\
↪"3.2\"),
                                             Modelica 3.2.1 will not match it.";
  output Boolean success;
end loadString;
```

30.1.256 mkdir

Creates a directory.

```
function mkdir
  input String newDirectory;
  output Boolean success;
end mkdir;
```

30.1.257 modifierToJSON

Parses a modifier given as a string and dumps it as JSON.

```
function modifierToJSON
  input String modifier;
  input Boolean prettyPrint = false;
  output String json;
end modifierToJSON;
```

30.1.258 moveClass

Moves a class up or down in a package.

```
function moveClass
  input TypeName className "the class that should be moved";
  input Integer offset "Offset in the class list.";
  output Boolean result;
end moveClass;
```

30.1.259 moveClassToBottom

Moves a class to the bottom of its enclosing class.

```
function moveClassToBottom
  input TypeName className;
  output Boolean result;
end moveClassToBottom;
```

30.1.260 moveClassToTop

Moves a class to the top of its enclosing class.

```
function moveClassToTop
  input TypeName className;
  output Boolean result;
end moveClassToTop;
```

30.1.261 newModel

Creates a new empty model in the given package.

```
function newModel
  input TypeName className;
  input TypeName withinPath;
  output Boolean success;
end newModel;
```

30.1.262 ngspicetoModelica

Converts ngspice netlist to Modelica code.

```
function ngspicetoModelica
  input String netlistfileName;
  output Boolean success = false;
end ngspicetoModelica;
```

30.1.263 numProcessors

Returns the number of available processors or threads.

```
function numProcessors
  output Integer result;
end numProcessors;
```

30.1.264 oms_RunFile

Simulates a single FMU or SSP model.

```
function oms_RunFile
  input String filename;
  output Integer status;
end oms_RunFile;
```

30.1.265 oms_addBus

OMSimulator: Adds a bus to a given component.

```
function oms_addBus
  input String cref;
  output Integer status;
end oms_addBus;
```

30.1.266 oms_addConnection

Adds a new connection between connectors A and B.

```
function oms_addConnection
  input String crefA;
  input String crefB;
  output Integer status;
end oms_addConnection;
```

30.1.267 oms_addConnector

Adds a connector to a given component.

```
function oms_addConnector
  input String cref;
  input oms_causality causality;
  input oms_signal_type type_;
  output Integer status;
end oms_addConnector;
```

30.1.268 oms_addConnectorToBus

Adds a connector to a bus.

```
function oms_addConnectorToBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_addConnectorToBus;
```

30.1.269 oms_addConnectorToTLMBus

Adds a connector to a TLM bus.

```
function oms_addConnectorToTLMBus
  input String busCref;
  input String connectorCref;
  input String type_;
  output Integer status;
end oms_addConnectorToTLMBus;
```

30.1.270 oms_addDynamicValueIndicator

Adds a dynamic value indicator.

```
function oms_addDynamicValueIndicator
  input String signal;
  input String lower;
  input String upper;
  input Real stepSize;
  output Integer status;
end oms_addDynamicValueIndicator;
```

30.1.271 oms_addEventIndicator

Adds an event indicator.

```
function oms_addEventIndicator
  input String signal;
  output Integer status;
end oms_addEventIndicator;
```

30.1.272 oms_addExternalModel

Adds an external model to a TLM system.

```
function oms_addExternalModel
  input String cref;
  input String path;
  input String startscript;
  output Integer status;
end oms_addExternalModel;
```

30.1.273 oms_addSignalsToResults

Adds all variables that match the given regex to the result file.

```
function oms_addSignalsToResults
  input String cref;
  input String regex;
  output Integer status;
end oms_addSignalsToResults;
```

30.1.274 oms_addStaticValueIndicator

Adds a static value indicator.

```
function oms_addStaticValueIndicator
  input String signal;
  input Real lower;
  input Real upper;
  input Real stepSize;
  output Integer status;
end oms_addStaticValueIndicator;
```

30.1.275 oms_addSubModel

Adds a component to a system.

```
function oms_addSubModel
  input String cref;
  input String fmuPath;
  output Integer status;
end oms_addSubModel;
```

30.1.276 oms_addSystem

Adds a (sub-)system to a model or system.

```
function oms_addSystem
  input String cref;
  input oms_system type_;
  output Integer status;
end oms_addSystem;
```

30.1.277 oms_addTLMBus

Adds a TLM bus.

```
function oms_addTLMBus
  input String cref;
  input oms_tlm_domain domain;
  input Integer dimensions;
  input oms_tlm_interpolation interpolation;
  output Integer status;
end oms_addTLMBus;
```

30.1.278 oms_addTLMConnection

Connects two TLM connectors.

```
function oms_addTLMConnection
  input String crefA;
```

(continues on next page)

(continued from previous page)

```
  input String crefB;
  input Real delay;
  input Real alpha;
  input Real linearimpedance;
  input Real angularimpedance;
  output Integer status;
end oms_addTLMConnection;
```

30.1.279 oms_addTimeIndicator

Adds a time indicator.

```
function oms_addTimeIndicator
  input String signal;
  output Integer status;
end oms_addTimeIndicator;
```

30.1.280 oms_compareSimulationResults

Compares a given signal in two result files.

```
function oms_compareSimulationResults
  input String filenameA;
  input String filenameB;
  input String var;
  input Real relTol;
  input Real absTol;
  output Integer status;
end oms_compareSimulationResults;
```

30.1.281 oms_copySystem

Copies a system.

```
function oms_copySystem
  input String source;
  input String target;
  output Integer status;
end oms_copySystem;
```

30.1.282 oms_delete

Deletes a connector, component, system, or model object.

```
function oms_delete
  input String cref;
  output Integer status;
end oms_delete;
```

30.1.283 oms_deleteConnection

Deletes the connection between two connectors.

```
function oms_deleteConnection
  input String crefA;
```

(continues on next page)

(continued from previous page)

```
  input String crefB;
  output Integer status;
end oms_deleteConnection;
```

30.1.284 oms_deleteConnectorFromBus

Deletes a connector from a given bus.

```
function oms_deleteConnectorFromBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_deleteConnectorFromBus;
```

30.1.285 oms_deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

```
function oms_deleteConnectorFromTLMBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_deleteConnectorFromTLMBus;
```

30.1.286 oms_export

Exports a composite model to a SPP file.

```
function oms_export
  input String cref;
  input String filename;
  output Integer status;
end oms_export;
```

30.1.287 oms_exportDependencyGraphs

Exports the dependency graphs of a given model to dot files.

```
function oms_exportDependencyGraphs
  input String cref;
  input String initialization;
  input String event;
  input String simulation;
  output Integer status;
end oms_exportDependencyGraphs;
```

30.1.288 oms_exportSnapshot

Lists the SSD representation of a given model, system, or component.

```
function oms_exportSnapshot
  input String cref;
  output String contents;
  output Integer status;
end oms_exportSnapshot;
```

30.1.289 oms_extractFMIKind

Extracts the FMI kind of a given FMU from the file system.

```
function oms_extractFMIKind
  input String filename;
  output Integer kind;
  output Integer status;
end oms_extractFMIKind;
```

30.1.290 oms_faultInjection

Defines a new fault injection block.

```
function oms_faultInjection
  input String signal;
  input oms_fault_type faultType;
  input Real faultValue;
  output Integer status;
end oms_faultInjection;
```

30.1.291 oms_getBoolean

Get boolean value of a given signal.

```
function oms_getBoolean
  input String cref;
  output Boolean value;
  output Integer status;
end oms_getBoolean;
```

30.1.292 oms_getFixedStepSize

Gets the fixed step size.

```
function oms_getFixedStepSize
  input String cref;
  output Real stepSize;
  output Integer status;
end oms_getFixedStepSize;
```

30.1.293 oms_getInteger

Get integer value of a given signal.

```
function oms_getInteger
  input String cref;
  input Integer value;
  output Integer status;
end oms_getInteger;
```

30.1.294 oms_getModelState

Gets the model state of the given model cref.

```
function oms_getModelState
  input String cref;
```

(continues on next page)

(continued from previous page)

```
  output Integer modelState;
  output Integer status;
end oms_getModelState;
```

30.1.295 oms_getReal

Get real value.

```
function oms_getReal
  input String cref;
  output Real value;
  output Integer status;
end oms_getReal;
```

30.1.296 oms_getSolver

Gets the selected solver method of the given system.

```
function oms_getSolver
  input String cref;
  output Integer solver;
  output Integer status;
end oms_getSolver;
```

30.1.297 oms_getStartTime

Gets the start time from the model.

```
function oms_getStartTime
  input String cref;
  output Real startTime;
  output Integer status;
end oms_getStartTime;
```

30.1.298 oms_getStopTime

Gets the stop time from the model.

```
function oms_getStopTime
  input String cref;
  output Real stopTime;
  output Integer status;
end oms_getStopTime;
```

30.1.299 oms_getSubModelPath

Returns the path of a given component.

```
function oms_getSubModelPath
  input String cref;
  output String path;
  output Integer status;
end oms_getSubModelPath;
```

30.1.300 oms_getSystemType

Gets the type of a given system.

```
function oms_getSystemType
  input String cref;
  output Integer type_;
  output Integer status;
end oms_getSystemType;
```

30.1.301 oms_getTolerance

Gets the tolerance of a given system or component.

```
function oms_getTolerance
  input String cref;
  output Real absoluteTolerance;
  output Real relativeTolerance;
  output Integer status;
end oms_getTolerance;
```

30.1.302 oms_getVariableStepSize

Gets the step size parameters.

```
function oms_getVariableStepSize
  input String cref;
  output Real initialStepSize;
  output Real minimumStepSize;
  output Real maximumStepSize;
  output Integer status;
end oms_getVariableStepSize;
```

30.1.303 oms_getVersion

Returns the version of the OMSimulator.

```
function oms_getVersion
  output String version;
end oms_getVersion;
```

30.1.304 oms_importFile

Imports a composite model from a SSP file.

```
function oms_importFile
  input String filename;
  output String cref;
  output Integer status;
end oms_importFile;
```

30.1.305 oms_importSnapshot

Loads a snapshot to restore a previous model state.

```
function oms_importSnapshot
  input String cref;
```

(continues on next page)

(continued from previous page)

```
  input String snapshot;
  output Integer status;
end oms_importSnapshot;
```

30.1.306 oms_initialize

Initializes a composite model.

```
function oms_initialize
  input String cref;
  output Integer status;
end oms_initialize;
```

30.1.307 oms_instantiate

Instantiates a given composite model.

```
function oms_instantiate
  input String cref;
  output Integer status;
end oms_instantiate;
```

30.1.308 oms_list

Lists the SSD representation of a given model, system, or component.

```
function oms_list
  input String cref;
  output String contents;
  output Integer status;
end oms_list;
```

30.1.309 oms_listUnconnectedConnectors

Lists all unconnected connectors of a given system.

```
function oms_listUnconnectedConnectors
  input String cref;
  output String contents;
  output Integer status;
end oms_listUnconnectedConnectors;
```

30.1.310 oms_loadSnapshot

Loads a snapshot to restore a previous model state.

```
function oms_loadSnapshot
  input String cref;
  input String snapshot;
  output String newCref;
  output Integer status;
end oms_loadSnapshot;
```

30.1.311 oms_newModel

Creates a new composite model.

```
function oms_newModel
  input String cref;
  output Integer status;
end oms_newModel;
```

30.1.312 oms_removeSignalsFromResults

Removes all variables that match the given regex from the result file.

```
function oms_removeSignalsFromResults
  input String cref;
  input String regex;
  output Integer status;
end oms_removeSignalsFromResults;
```

30.1.313 oms_rename

Renames a model, system, or component.

```
function oms_rename
  input String cref;
  input String newCref;
  output Integer status;
end oms_rename;
```

30.1.314 oms_reset

Reset the composite model after a simulation run.

```
function oms_reset
  input String cref;
  output Integer status;
end oms_reset;
```

30.1.315 oms_setBoolean

Sets the value of a given boolean signal.

```
function oms_setBoolean
  input String cref;
  input Boolean value;
  output Integer status;
end oms_setBoolean;
```

30.1.316 oms_setCommandLineOption

Sets special flags.

```
function oms_setCommandLineOption
  input String cmd;
  output Integer status;
end oms_setCommandLineOption;
```

30.1.317 oms_setFixedStepSize

Sets the fixed step size.

```
function oms_setFixedStepSize
  input String cref;
  input Real stepSize;
  output Integer status;
end oms_setFixedStepSize;
```

30.1.318 oms_setInteger

Sets the value of a given integer signal.

```
function oms_setInteger
  input String cref;
  input Integer value;
  output Integer status;
end oms_setInteger;
```

30.1.319 oms_setLogFile

Redirects logging output to file or std streams.

```
function oms_setLogFile
  input String filename;
  output Integer status;
end oms_setLogFile;
```

30.1.320 oms_setLoggingInterval

Sets the logging interval of the simulation.

```
function oms_setLoggingInterval
  input String cref;
  input Real loggingInterval;
  output Integer status;
end oms_setLoggingInterval;
```

30.1.321 oms_setLoggingLevel

Enables/disables debug logging.

```
function oms_setLoggingLevel
  input Integer logLevel;
  output Integer status;
end oms_setLoggingLevel;
```

30.1.322 oms_setReal

Sets the value of a given real signal.

```
function oms_setReal
  input String cref;
  input Real value;
  output Integer status;
end oms_setReal;
```

30.1.323 oms_setRealInputDerivative

Sets the first order derivative of a real input signal.

```
function oms_setRealInputDerivative
  input String cref;
  input Real value;
  output Integer status;
end oms_setRealInputDerivative;
```

30.1.324 oms_setResultFile

Sets the result file of the simulation.

```
function oms_setResultFile
  input String cref;
  input String filename;
  input Integer bufferSize;
  output Integer status;
end oms_setResultFile;
```

30.1.325 oms_setSignalFilter

Sets a signal filter.

```
function oms_setSignalFilter
  input String cref;
  input String regex;
  output Integer status;
end oms_setSignalFilter;
```

30.1.326 oms_setSolver

Sets the solver method for the given system.

```
function oms_setSolver
  input String cref;
  input oms_solver solver;
  output Integer status;
end oms_setSolver;
```

30.1.327 oms_setStartTime

Sets the start time of the simulation.

```
function oms_setStartTime
  input String cref;
  input Real startTime;
  output Integer status;
end oms_setStartTime;
```

30.1.328 oms_setStopTime

Sets the stop time of the simulation.

```
function oms_setStopTime
  input String cref;
```

(continues on next page)

(continued from previous page)

```
  input Real stopTime;
  output Integer status;
end oms_setStopTime;
```

30.1.329 oms_setTLMPositionAndOrientation

Sets initial position and orientation for a TLM 3D interface.

```
function oms_setTLMPositionAndOrientation
  input String cref;
  input Real x1;
  input Real x2;
  input Real x3;
  input Real A11;
  input Real A12;
  input Real A13;
  input Real A21;
  input Real A22;
  input Real A23;
  input Real A31;
  input Real A32;
  input Real A33;
  output Integer status;
end oms_setTLMPositionAndOrientation;
```

30.1.330 oms_setTLMSocketData

Sets data for TLM socket communication.

```
function oms_setTLMSocketData
  input String cref;
  input String address;
  input Integer managerPort;
  input Integer monitorPort;
  output Integer status;
end oms_setTLMSocketData;
```

30.1.331 oms_setTempDirectory

Sets new temp directory.

```
function oms_setTempDirectory
  input String newTempDir;
  output Integer status;
end oms_setTempDirectory;
```

30.1.332 oms_setTolerance

Sets the tolerance for a given model or system.

```
function oms_setTolerance
  input String cref;
  input Real absoluteTolerance;
  input Real relativeTolerance;
  output Integer status;
end oms_setTolerance;
```

30.1.333 oms_setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
function oms_setVariableStepSize
  input String cref;
  input Real initialStepSize;
  input Real minimumStepSize;
  input Real maximumStepSize;
  output Integer status;
end oms_setVariableStepSize;
```

30.1.334 oms_setWorkingDirectory

Sets a new working directory.

```
function oms_setWorkingDirectory
  input String newWorkingDir;
  output Integer status;
end oms_setWorkingDirectory;
```

30.1.335 oms_simulate

Simulates a composite model.

```
function oms_simulate
  input String cref;
  output Integer status;
end oms_simulate;
```

30.1.336 oms_stepUntil

Simulates a composite model until a given time value.

```
function oms_stepUntil
  input String cref;
  input Real stopTime;
  output Integer status;
end oms_stepUntil;
```

30.1.337 oms_terminate

Terminates a given composite model.

```
function oms_terminate
  input String cref;
  output Integer status;
end oms_terminate;
```

30.1.338 optimize

Generates an optimization executable for a Modelica/Optimica model and runs it.

```
function optimize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.
↪0";
```

(continues on next page)

(continued from previous page)

```
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file. <default>␣
↪= 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = DAE.SCONST("optimization") "optimize a modelica/optimica␣
↪model.";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String optimizationResults;
end optimize;
```

30.1.339 parseEncryptedPackage

Parses an encrypted package and returns the names of the parsed classes.

```
function parseEncryptedPackage
  input String fileName;
  input String workdir = "<default>" "The output directory for imported encrypted␣
↪files. <default> will put the files to current working directory.";
  output TypeName names[:];
end parseEncryptedPackage;
```

30.1.340 parseFile

Parses a Modelica file and returns the parsed classes.

```
function parseFile
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end parseFile;
```

30.1.341 parseString

Parses a string containing Modelica definitions and returns the parsed classes.

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

30.1.342 plot

Displays a plot with selected variables using OMPlot.

```
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 = "" "This text will be used as the diagram title.";
  input String grid = "simple" "Sets the grid for the plot i.e simple, detailed, none.
↪";
  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 left 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 left vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2,␣
↪DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  input String yAxis = "L" "Sets the variable to be plotted on the left (L) or right␣
↪(R) y-axis.";
  input String yLabelRight = "" "This text will be used as the right vertical label␣
↪in the diagram.";
  input Real yRangeRight[2] = {0.0, 0.0} "Determines the right vertical interval that␣
↪is visible in the diagram. {0,0} will select a suitable range.";
  output Boolean success "Returns true on success";
end plot;
```

30.1.343 plotAll

Displays a plot with all variables using OMPlot.

```
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 = "" "This text will be used as the diagram title.";
  input String grid = "simple" "Sets the grid for the plot i.e simple, detailed, none.
↪";
  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␣
```

(continues on next page)

(continued from previous page)

```
↪diagram.";
  input String yLabel = "" "This text will be used as the left 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 left vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2,␣
↪DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  input String yAxis = "L" "Sets the variable to be plotted on the left (L) or right␣
↪(R) y-axis.";
  input String yLabelRight = "" "This text will be used as the right vertical label␣
↪in the diagram.";
  input Real yRangeRight[2] = {0.0, 0.0} "Determines the right vertical interval that␣
↪is visible in the diagram. {0,0} will select a suitable range.";
  output Boolean success "Returns true on success";
end plotAll;
```

30.1.344 plotParametric

Displays a parametric plot with two variables using OMPlot.

```
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 = "" "This text will be used as the diagram title.";
  input String grid = "simple" "Sets the grid for the plot i.e simple, detailed, none.
↪";
  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 = "" "This text will be used as the horizontal label in the␣
↪diagram.";
  input String yLabel = "" "This text will be used as the left 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 left vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1, DashLine=2,␣
↪DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
  input String footer = "" "This text will be used as the diagram footer.";
```

(continues on next page)

(continued from previous page)

```
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  input String yAxis = "L" "Sets the variable to be plotted on the left (L) or right␣
↪(R) y-axis.";
  input String yLabelRight = "" "This text will be used as the right vertical label␣
↪in the diagram.";
  input Real yRangeRight[2] = {0.0, 0.0} "Determines the right vertical interval that␣
↪is visible in the diagram. {0,0} will select a suitable range.";
  output Boolean success "Returns true on success";
end plotParametric;
```

30.1.345 qualifyPath

Returns the fully qualified path for the given path in a class.

```
function qualifyPath
  input TypeName classPath;
  input TypeName path;
  output TypeName qualifiedPath;
end qualifyPath;
```

30.1.346 readFile

Returns the contents of a file.

```
impure function readFile
  input String fileName;
  output String contents;
end readFile;
```

30.1.347 readSimulationResult

Reads a result file, returning a matrix corresponding to the variables and size given.

```
function readSimulationResult
  input String filename;
  input VariableNames variables "e.g. {a.b, a[1].b[3].c}, or a single VariableName";
  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;
```

30.1.348 readSimulationResultSize

Returns the number of intervals that are present in the output file.

```
function readSimulationResultSize
  input String fileName;
  output Integer sz;
end readSimulationResultSize;
```

30.1.349 readSimulationResultVars

Returns the variables in a simulation results file.

```
function readSimulationResultVars
  input String fileName;
  input Boolean readParameters = true;
  input Boolean openmodelicaStyle = false;
  output String[:] vars;
end readSimulationResultVars;
```

30.1.350 realpath

Get full path name of file or directory name

```
function realpath
  input String name "Absolute or relative file or directory name";
  output String fullName "Full path of 'name'";
end realpath;
```

30.1.351 reduceTerms

Calls buildModel with the --reduceTerms flag enabled.

```
function reduceTerms
  input TypeName className "the class that should be built";
  input Real startTime = 0.0 "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "dassl" "integration method used for simulation. <default> =␣
↪dassl";
  input String fileNamePrefix = "" "fileNamePrefix. <default> = \"\"";
  input String options = "" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "" "cflags. <default> = \"\"";
  input String simflags = "" "simflags. <default> = \"\"";
  input String labelstoCancel = "";
  output String[2] buildModelResults;
end reduceTerms;
```

30.1.352 refactorClass

Updates old graphical annotations to Modelica standard ones in a class.

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

30.1.353 refactorDiagramAnnotation

Updates an old Diagram annotation to a Modelica standard one in the given class.

```
function refactorDiagramAnnotation
  input TypeName className;
  output Expression result;
end refactorDiagramAnnotation;
```

30.1.354 refactorIconAnnotation

Updates an old Icon annotation to a Modelica standard one in the given class.

```
function refactorIconAnnotation
  input TypeName className;
  output Expression result;
end refactorIconAnnotation;
```

30.1.355 regex

Matches a string with a regular expression and returns the result.

```
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;
```

30.1.356 regexBool

Returns true if the string matches the regular expression.

```
function regexBool
  input String str;
  input String re;
  input Boolean extended = true "Use POSIX extended or regular syntax";
  input Boolean caseInsensitive = false;
  output Boolean matches;
end regexBool;
```

30.1.357 regularFileExists

Returns whether the given file exists or not.

```
function regularFileExists
  input String fileName;
  output Boolean exists;
end regularFileExists;
```

30.1.358 releaseModelInstanceReference

Releases a handle returned by getModelInstanceReference or getModelInstanceAnnotationReference. Returns true on success.

```
function releaseModelInstanceReference
  input Integer handle;
  output Boolean success;
end releaseModelInstanceReference;
```

30.1.359 reloadClass

Reloads the file associated with the given loaded class.

```
function reloadClass
  input TypeName name;
  input String encoding = "UTF-8";
  output Boolean success;
end reloadClass;
```

30.1.360 remove

Removes a file or directory.

```
function remove
  input String path;
  output Boolean success "Returns true on success.";
end remove;
```

30.1.361 removeComponentModifiers

Removes the component modifiers.

```
function removeComponentModifiers
  input TypeName class_;
  input String componentName;
  input Boolean keepRedeclares = false;
  output Boolean success;
end removeComponentModifiers;
```

30.1.362 removeElementModifiers

Removes the element (component or short class) modifiers.

```
function removeElementModifiers
  input TypeName className;
  input String componentName;
  input Boolean keepRedeclares = false;
  output Boolean success;
end removeElementModifiers;
```

30.1.363 removeExtendsModifiers

Removes the extends modifiers of a class.

```
function removeExtendsModifiers
  input TypeName className;
```

(continues on next page)

(continued from previous page)

```
  input TypeName baseClassName;
  input Boolean keepRedeclares = false;
  output Boolean success;
end removeExtendsModifiers;
```

30.1.364 renameClass

Renames a class and updates references to it.

```
function renameClass
  input TypeName oldName "The path of the class to rename.";
  input TypeName newName "The new non-qualified name of the class.";
  output TypeName[:] result;
end renameClass;
```

30.1.365 renameComponent

Renames a component and updates references to it.

```
function renameComponent
  input TypeName classPath;
  input VariableName oldName;
  input VariableName newName;
  output TypeName[:] result;
end renameComponent;
```

30.1.366 renameComponentInClass

Renames a component in a class.

```
function renameComponentInClass
  input TypeName classPath;
  input VariableName oldName;
  input VariableName newName;
  output TypeName[:] result;
end renameComponentInClass;
```

30.1.367 reopenStandardStream

Changes which file is associated with a standard stream.

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

30.1.368 restoreAST

Restores an AST that was previously stored with storeAST.

```
function restoreAST
  input Integer id;
  output Boolean success;
end restoreAST;
```

30.1.369 reverseLookup

```
function reverseLookup
  input TypeName name;
  input TypeName scope = $Code(AllLoadedClasses);
  input Boolean exactMatch = true;
  input Boolean prettyPrint = false;
  output String matches;
end reverseLookup;
```

30.1.370 rewriteBlockCall

Function for property modeling, transforms block calls into instantiations for a loaded model

```
function rewriteBlockCall
  input TypeName className;
  input TypeName inDefs;
  output Boolean success;
end rewriteBlockCall;
```

30.1.371 runConversionScript

Runs a conversion script on a selected package.

```
function runConversionScript
  input TypeName packageToConvert;
  input String scriptFile;
  output Boolean success;
end runConversionScript;
```

30.1.372 runScript

Runs the mos-script specified by the filename.

```
impure function runScript
  input String fileName "*.mos";
  output String result;
end runScript;
```

30.1.373 runScriptParallel

Runs multiple scripts in parallel.

```
function runScriptParallel
  input String scripts[:];
  input Integer numThreads = numProcessors();
  input Boolean useThreads = false;
  output Boolean results[:];
end runScriptParallel;
```

30.1.374 save

Saves a class to the file(s) it's defined in.

```
function save
  input TypeName className;
```

(continues on next page)

(continued from previous page)

```
  output Boolean success;
end save;
```

30.1.375 saveAll

Saves the entire loaded AST to file.

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

30.1.376 saveModel

Saves a loaded model to the given file.

```
function saveModel
  input String fileName;
  input TypeName className;
  output Boolean success;
end saveModel;
```

30.1.377 saveTotalModel

Saves a model and dependencies to a single file.

```
function saveTotalModel
  input String fileName;
  input TypeName className;
  input Boolean stripAnnotations = false;
  input Boolean stripComments = false;
  input Boolean obfuscate = false;
  output Boolean success;
end saveTotalModel;
```

30.1.378 saveTotalModelDebug

Saves a model and dependencies to a single file using a heuristic.

```
function saveTotalModelDebug
  input String filename;
  input TypeName className;
  input Boolean stripAnnotations = false;
  input Boolean stripComments = false;
  input Boolean obfuscate = false;
  output Boolean success;
end saveTotalModelDebug;
```

30.1.379 saveTotalSCode

Alias for saveTotalModel

30.1.380 searchClassNames

Searches for a string in the loaded classes.

```
function searchClassNames
  input String searchText;
  input Boolean findInText = false;
  output TypeName classNames[:];
end searchClassNames;
```

30.1.381 setAnnotationVersion

Sets the annotation version.

```
function setAnnotationVersion
  input String annotationVersion;
  output Boolean success;
end setAnnotationVersion;
```

30.1.382 setCFlags

Sets the C compiler flags (CFLAGS) used for simulation code.

```
function setCFlags
  input String inString;
  output Boolean success;
end setCFlags;
```

30.1.383 setCXXCompiler

Sets the C++ compiler (CXX) used for simulation code.

```
function setCXXCompiler
  input String compiler;
  output Boolean success;
end setCXXCompiler;
```

30.1.384 setCheapMatchingAlgorithm

Sets the cheap matching algorithm.

```
function setCheapMatchingAlgorithm
  input Integer matchingAlgorithm;
  output Boolean success;
end setCheapMatchingAlgorithm;
```

30.1.385 setClassComment

Sets a class comment.

```
function setClassComment
  input TypeName class_;
  input String filename;
  output Boolean success;
end setClassComment;
```

30.1.386 setCommandLineOptions

Sets command line options for the compiler.

```
function setCommandLineOptions
  input String options;
  output Boolean success;
end setCommandLineOptions;
```

30.1.387 setCompiler

Sets the C compiler (CC) used for simulation code.

```
function setCompiler
  input String compiler;
  output Boolean success;
end setCompiler;
```

30.1.388 setCompilerFlags

Same as setCFlags.

```
function setCompilerFlags
  input String compilerFlags;
  output Boolean success;
end setCompilerFlags;
```

30.1.389 setComponentComment

Sets the comment on a component.

```
function setComponentComment
  input TypeName className;
  input TypeName componentName;
  input String comment;
  output Boolean success;
end setComponentComment;
```

30.1.390 setComponentDimensions

Sets the array dimensions of a component.

```
function setComponentDimensions
  input TypeName className;
  input TypeName componentName;
  input Expression dimensions;
  output Boolean success;
end setComponentDimensions;
```

30.1.391 setComponentModifierValue

Deprecated; alias for setElementModifierValue.

30.1.392 setComponentProperties

Sets the properties of a component in a class.

```
function setComponentProperties
  input TypeName className;
  input TypeName componentName;
  input Boolean[:] prefixArray;
  input String[1] variability;
  input Boolean[2] innerOuter;
  input String[1] direction;
  output Boolean success;
end setComponentProperties;
```

30.1.393 setConnectionComment

Sets the description string on a connect equation in the given class.

```
function setConnectionComment
  input TypeName className;
  input VariableName connector1;
  input VariableName connector2;
  input String comment;
  output Boolean success;
end setConnectionComment;
```

30.1.394 setDebugFlags

Sets compiler debug flags.

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

30.1.395 setDefaultOpenCLDevice

Sets the default OpenCL device to be used. 0 selects one automatically.

```
function setDefaultOpenCLDevice
  input Integer defdevid;
  output Boolean success;
end setDefaultOpenCLDevice;
```

30.1.396 setDocumentationAnnotation

Sets the Documentation annotation in a class.

```
function setDocumentationAnnotation
  input TypeName class_;
  input String info = "";
  input String revisions = "";
  output Boolean bool;
end setDocumentationAnnotation;
```

30.1.397 setElementAnnotation

Sets the annotation on a component or class element.

```
function setElementAnnotation
  input TypeName elementName;
  input ExpressionOrModification annotationMod;
  output Boolean success;
end setElementAnnotation;
```

30.1.398 setElementModifierValue

Sets a modifier on an element in a class definition.

```
function setElementModifierValue
  input TypeName className;
  input TypeName elementName;
  input ExpressionOrModification modifier;
  output Boolean success;
end setElementModifierValue;
```

30.1.399 setElementType

Changes the type of a component or short class element.

```
function setElementType
  input TypeName elementName;
  input VariableName typeName;
  output Boolean success;
end setElementType;
```

30.1.400 setEnvironmentVar

Sets the value of the given environment variable.

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

30.1.401 setExtendsModifier

Sets a modifier on an extends clause in a class definition.

```
function setExtendsModifier
  input TypeName className;
  input TypeName extendsName;
  input ExpressionOrModification modifier;
  output Boolean success;
end setExtendsModifier;
```

30.1.402 setExtendsModifierValue

Sets a modifier on an element in an extends clause in a class.

```
function setExtendsModifierValue
  input TypeName className;
  input TypeName extendsName;
  input TypeName elementName;
```

(continues on next page)

(continued from previous page)

```
  input ExpressionOrModification modifier;
  output Boolean success;
end setExtendsModifierValue;
```

30.1.403 setIndexReductionMethod

Sets the index reduction method.

```
function setIndexReductionMethod
  input String method;
  output Boolean success;
end setIndexReductionMethod;
```

30.1.404 setInitXmlStartValue

Sets the start value for a variable in an initialization file.

```
function setInitXmlStartValue
  input String fileName;
  input String variableName;
  input String startValue;
  input String outputFile;
  output Boolean success = false;
end setInitXmlStartValue;
```

30.1.405 setInstallationDirectoryPath

Sets the OPENMODELICAHOME environment variable.

```
function setInstallationDirectoryPath
  input String installationDirectoryPath;
  output Boolean success;
end setInstallationDirectoryPath;
```

30.1.406 setLanguageStandard

Sets the Modelica Language Standard.

```
function setLanguageStandard
  input String inVersion;
  output Boolean success;
end setLanguageStandard;
```

30.1.407 setLinker

Sets the linker (LINK) used for simulation code.

```
function setLinker
  input String linker;
  output Boolean success;
end setLinker;
```

30.1.408 setLinkerFlags

Sets the linker flags (LDFLAGS) used for simulation code.

```
function setLinkerFlags
  input String linkerFlags;
  output Boolean success;
end setLinkerFlags;
```

30.1.409 setMatchingAlgorithm

Sets the matching algorithm.

```
function setMatchingAlgorithm
  input String matchingAlgorithm;
  output Boolean success;
end setMatchingAlgorithm;
```

30.1.410 setModelicaPath

Sets the Modelica library path.

```
function setModelicaPath
  input String modelicaPath;
  output Boolean success;
end setModelicaPath;
```

30.1.411 setNoSimplify

Sets the noSimplify flag.

```
function setNoSimplify
  input Boolean noSimplify;
  output Boolean success;
end setNoSimplify;
```

30.1.412 setOrderConnections

Sets the orderConnection flag.

```
function setOrderConnections
  input Boolean orderConnections;
  output Boolean success;
end setOrderConnections;
```

30.1.413 setParameterValue

Sets the binding equation of a component.

```
function setParameterValue
  input TypeName className;
  input TypeName variableName;
  input Expression value;
  output Boolean success;
end setParameterValue;
```

30.1.414 setPostOptModules

Sets post optimization modules for the backend.

```
function setPostOptModules
  input String modules;
  output Boolean success;
end setPostOptModules;
```

30.1.415 setPreOptModules

Sets pre optimization modules for the backend.

```
function setPreOptModules
  input String modules;
  output Boolean success;
end setPreOptModules;
```

30.1.416 setShowAnnotations

Sets the value of the --showAnnotations flag.

```
function setShowAnnotations
  input Boolean show;
  output Boolean success;
end setShowAnnotations;
```

30.1.417 setSourceFile

Sets the filename for a class.

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

30.1.418 setTearingMethod

Sets the tearing method used by the backend.

```
function setTearingMethod
  input String tearingMethod;
  output Boolean success;
end setTearingMethod;
```

30.1.419 setTempDirectoryPath

Sets the current user temporary directory location.

```
function setTempDirectoryPath
  input String tempDirectoryPath;
  output Boolean success;
end setTempDirectoryPath;
```

30.1.420 setVectorizationLimit

Sets the vectorization limit used by the old frontend.

```
function setVectorizationLimit
  input Integer vectorizationLimit;
  output Boolean success;
end setVectorizationLimit;
```

30.1.421 simulate

Simulates a model.

```
function simulate
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.
↪0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "<default>" "integration method used for simulation. <default>
↪ = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  input String resimulateExecutable = "" "If non-empty, skip translation and build␣
↪and simulate this already-built executable directly.";
  output SimulationResult simulationResults;
  record SimulationResult
    String resultFile;
    String simulationOptions;
    String messages;
    Real timeFrontend;
    Real timeBackend;
    Real timeSimCode;
    Real timeTemplates;
    Real timeCompile;
    Real timeSimulation;
    Real timeTotal;
  end SimulationResult;
end simulate;
```

30.1.422 solveLinearSystem

Solve A*X = B using dgesv.

```
function solveLinearSystem
  input Real[size(B, 1), size(B, 1)] A;
  input Real[:] B;
  output Real[size(B, 1)] X;
  output Integer info;
end solveLinearSystem;
```

30.1.423 sortStrings

Sorts a string array in ascending order.

```
function sortStrings
  input String arr[:];
  output String sorted[:];
end sortStrings;
```

30.1.424 stat

Returns status for a file.

```
impure function stat
  input String fileName;
  output Boolean success;
  output Real fileSize;
  output Real mtime;
end stat;
```

30.1.425 storeAST

Stores the AST and returns an id that can be used to restore it with restoreAST.

```
function storeAST
  output Integer id;
end storeAST;
```

30.1.426 stringReplace

Replaces all occurrences of a token with another token in a string.

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

30.1.427 stringSplit

Splits a string at the places given by the character

```
function stringSplit
  input String string;
  input String token "single character only";
  output String[:] strings;
end stringSplit;
```

30.1.428 stringTypeName

Constructs a TypeName from a string.

```
function stringTypeName
  input String str;
  output TypeName cl;
end stringTypeName;
```

30.1.429 stringVariableName

Constructs a VariableName from a string.

```
function stringVariableName
  input String str;
  output VariableName cl;
end stringVariableName;
```

30.1.430 strtok

Splits a string at the places given by the token.

```
function strtok
  input String string;
  input String token;
  output String[:] strings;
end strtok;
```

30.1.431 system

Similar to system(3). Executes the given command in the system shell.

```
impure function system
  input String callStr "String to call: sh -c $callStr";
  input String outputFile = "" "The output is redirected to this file (unless already␣
↪done by callStr)";
  output Integer retval "Return value of the system call; usually 0 on success";
end system;
```

30.1.432 system_parallel

Similar to system(3). Executes the given commands in the system shell, in parallel if omc was compiled using OpenMP.

```
impure function system_parallel
  input String callStr[:] "String to call: sh -c $callStr";
  input Integer numThreads = numProcessors();
  output Integer retval[:] "Return value of the system call; usually 0 on success";
end system_parallel;
```

30.1.433 testsuiteFriendlyName

Converts a filename to a testsuite friendly one.

```
function testsuiteFriendlyName
  input String path;
  output String fixed;
end testsuiteFriendlyName;
```

30.1.434 threadWorkFailed

Exits the current thread with a failure.

30.1.435 translateGraphics

Translates old graphical annotations to Modelica standard annotations.

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

30.1.436 translateModel

Translates a modelica model into C code without building it.

```
function translateModel
  input TypeName className "the class that should be built";
  input Real startTime = "<default>" "the start time of the simulation. <default> = 0.
↪0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default> =␣
↪1e-6";
  input String method = "<default>" "integration method used for simulation. <default>
↪ = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\"";
  input String variableFilter = ".*" "Only variables fully matching the regexp are␣
↪stored in the result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output Boolean success;
end translateModel;
```

30.1.437 translateModelFMU

Translates a model into C code for a FMU without building it.

```
function translateModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation), me_
↪cs (both model exchange and co-simulation)";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"className\"
↪";
  input String platforms[:] = {"static"} "The list of platforms to generate code for.
                                          \"dynamic\"=current platform, dynamically␣
↪link the runtime.
                                          \"static\"=current platform, statically␣
↪link everything.
                                          \"<cpu>-<vendor>-<os>\", host tripple, e.
↪g. \"x86_64-linux-gnu\" or \"x86_64-w64-mingw32\".
                                          \"<cpu>-<vendor>-<os> docker run ghcr.io/
↪openmodelica/crossbuild:v1.27.0\" host triple with OpenModelica supplied Docker␣
↪image, e.g. \"x86_64-linux-gnu docker run ghcr.io/openmodelica/crossbuild:v1.27.0\".
                                          \"<cpu>-<vendor>-<os> docker run <image>\
↪" host triple with Docker image, e.g. \"x86_64-linux-gnu docker run --pull=never␣
↪multiarch/crossbuild\"";
```

(continues on next page)

(continued from previous page)

```
  input Boolean includeResources = false "include Modelica based resources via␣
↪loadResource or not";
  output Boolean success;
end translateModelFMU;
```

30.1.438 translateResidualsDAE

```
function translateResidualsDAE
  input TypeName className;
  input String fileNamePrefix = "<default>";
  output Boolean success;
end translateResidualsDAE;
```

30.1.439 typeNameString

Converts a TypeName to a string.

```
function typeNameString
  input TypeName cl;
  output String out;
end typeNameString;
```

30.1.440 typeNameStrings

Converts a TypeName to a list of strings.

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

30.1.441 typeOf

Returns the type of an interactive variable.

```
function typeOf
  input VariableName variableName;
  output String result;
end typeOf;
```

30.1.442 unloadOMSimulator

Frees the OMSimulator instances.

```
function unloadOMSimulator
  output Integer status;
end unloadOMSimulator;
```

30.1.443 updateComponent

Updates an existing component.

```
function updateComponent
  input TypeName componentName;
  input TypeName typeName;
```

(continues on next page)

(continued from previous page)

```
  input TypeName classPath;
  input Expression binding = $Code(());
  input ExpressionOrModification modification = $Code();
  input Expression comment = $Code(());
  input Expression annotate = $Code(());
  output Boolean success;
end updateComponent;
```

30.1.444 updateConnection

Updates the connection annotation in the class.

```
function updateConnection
  input TypeName className;
  input String from;
  input String to;
  input ExpressionOrModification annotate;
  output Boolean result;
end updateConnection;
```

30.1.445 updateConnectionAnnotation

Updates the connection annotation in the class.

```
function updateConnectionAnnotation
  input TypeName className;
  input String from;
  input String to;
  input String annotate;
  output Boolean result;
end updateConnectionAnnotation;
```

30.1.446 updateConnectionNames

Updates the connection connector names in the class.

```
function updateConnectionNames
  input TypeName className;
  input String from;
  input String to;
  input String fromNew;
  input String toNew;
  output Boolean result;
end updateConnectionNames;
```

30.1.447 updateEquation

Replaces an equation with another equation in a class.

```
function updateEquation
  input TypeName className "The name of the class";
  input String oldEq "The equation to replace";
  input String newEq "The equation to replace with";
  input Boolean matchAll = false "Update all matching equations if true, otherwise␣
↪only the first matching";
  input Boolean matchShallow = true "Ignore nested equations if true, otherwise match␣
```

(continues on next page)

(continued from previous page)

```
↪the equations recursively";
  input Boolean matchDescription = false "Match description strings/annotations if␣
↪true, otherwise ignore them";
  input Boolean mergeDescription = false "Keep description string/annotations from␣
↪the old equation if true";
  output Boolean success "true if any equation was updated, otherwise false";
end updateEquation;
```

30.1.448 updateInitialState

Updates an initial state in a class.

```
function updateInitialState
  input TypeName cl;
  input String state;
  input ExpressionOrModification annotate;
  output Boolean bool;
end updateInitialState;
```

30.1.449 updatePackageIndex

Updates the package index.

```
function updatePackageIndex
  output Boolean result;
end updatePackageIndex;
```

30.1.450 updateTransition

Updates a transition in a class.

```
function updateTransition
  input TypeName cl;
  input String from;
  input String to;
  input String oldCondition;
  input Boolean oldImmediate;
  input Boolean oldReset;
  input Boolean oldSynchronize;
  input Integer oldPriority;
  input String newCondition;
  input Boolean newImmediate;
  input Boolean newReset;
  input Boolean newSynchronize;
  input Integer newPriority;
  input ExpressionOrModification annotate;
  output Boolean bool;
end updateTransition;
```

30.1.451 upgradeInstalledPackages

Upgrades installed packages.

```
function upgradeInstalledPackages
  input Boolean installNewestVersions = true;
```

(continues on next page)

(continued from previous page)

```
  output Boolean result;
end upgradeInstalledPackages;
```

30.1.452 uriToFilename

Converts a URI to a filename.

```
function uriToFilename
  input String uri;
  output String filename = "";
end uriToFilename;
```

30.1.453 val

Return the value of a variable at a given time in the simulation results

```
function val
  input VariableName var;
  input Real timePoint = 0.0;
  input String fileName = "<default>" "The contents of the currentSimulationResult␣
↪variable";
  output Real valAtTime;
end val;
```

30.1.454 writeFile

Writes data to a file.

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

30.2 Simulation Parameter Sweep

Following example shows how to update the parameters and re-run the simulation without compiling the model.

```
loadFile("BouncingBall.mo");
getErrorString();
// build the model once
buildModel(BouncingBall);
getErrorString();
for i in 1:3 loop
  // We update the parameter e start value from 0.7 to "0.7 + i".
  value := 0.7 + i;
  // call the generated simulation code to produce a result file BouncingBall%i%_res.
↪mat
  system("./BouncingBall -override=e="+String(value)+" -r=BouncingBall" + String(i) +
↪"_res.mat");
  getErrorString();
end for;
```

We used the BouncingBall.mo in the example above. The above example produces three result files each containing different start value for e i.e., 1.7, 2.7, 3.7.

30.3 Examples

The following is an interactive session with the OpenModelica environment including some of the abovementioned commands and examples. Run the examples below using OMShell or OMNotebook.

We type in a very small model in the command window:

```
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0+time; y = 6.0;
end Test;
```

We give the command to flatten a model:

```
>>> instantiateModel(Test)
class Test "Testing OpenModelica Scripts"
  Real x;
  Real y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

A range expression is typed in:

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

It is multiplied by 2:

```
>>> a*2
{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

The variables are cleared:

```
>>> clearVariables()
true
```

We print the loaded class test from its internal representation:

```
>>> list(Test)
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

We get the name and other properties of a class:

```
>>> getClassNames()
{Test, ProfilingTest}
>>> getClassComment(Test)
"Testing OpenModelica Scripts"
>>> isPartial(Test)
false
>>> isPackage(Test)
false
```

(continues on next page)

(continued from previous page)

```
>>> isModel(Test)
true
>>> checkModel(Test)
"Check of Test completed successfully.
Class Test has 2 equation(s) and 2 variable(s).
2 of these are trivial equation(s)."
```

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

```
>>> simulate(Test, stopTime=3.0)
record SimulationResult
    resultFile = "«DOCHOME»/Test_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500,␣
↪tolerance = 1e-6, method = 'dassl', fileNamePrefix = 'Test', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
LOG_STDOUT        | info    | Time measurements are stored in Test_prof.html (human-
↪readable) and Test_prof.xml (for XSL transforms or more details)
",
    timeFrontend = 7.754270000000001e-4,
    timeBackend = 0.0040063600000000005,
    timeSimCode = 0.001187071,
    timeTemplates = 0.001702701,
    timeCompile = 0.16516914100000002,
    timeSimulation = 0.011115003,
    timeTotal = 0.18400481600000002
end SimulationResult;
>>> val(x , 2.0)
7.0
```

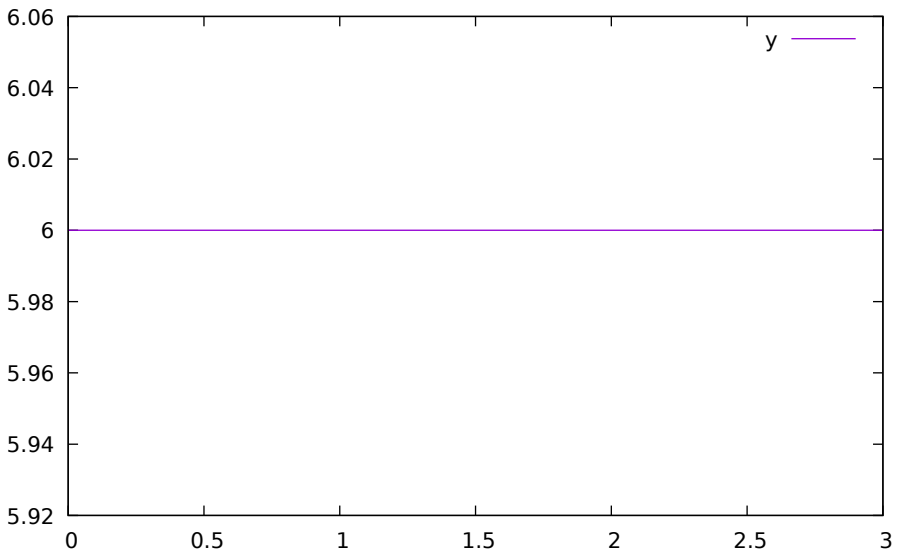


Figure 30.1: Plot generated by OpenModelica+gnuplot

```
>>> plotAll()
```

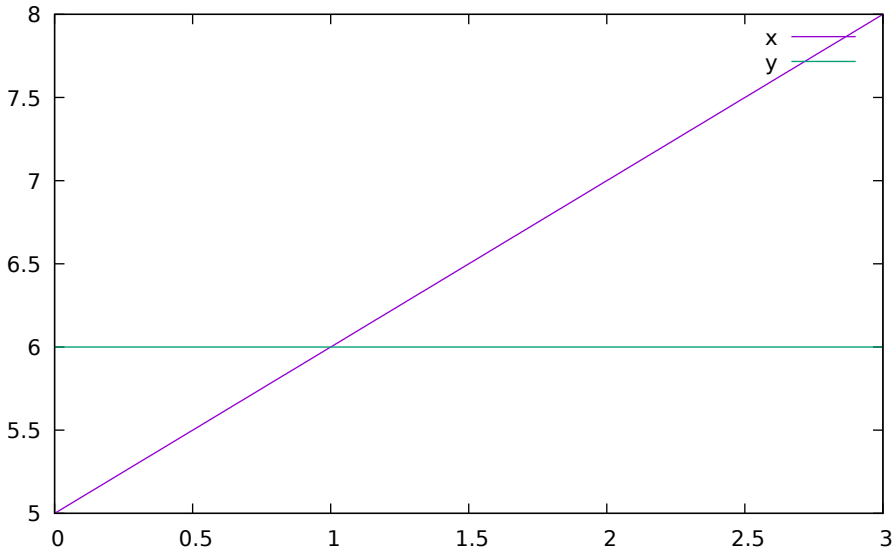


Figure 30.2: Plot generated by OpenModelica+gnuplot

30.3.1 Interactive Function Calls, Reading, and Writing

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

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

The function bubblesort is called to sort this vector in descending order. The sorted result is returned together with its type. Note that the result vector is of type Real[:], instantiated as Real[12], since this is the declared type of the function result. The input Integer vector was automatically converted to a Real vector according to the Modelica type coercion rules.

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/bubblesort.mo
↪")
true
>>> bubblesort(x)
{12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0}
```

Now we want to try another small application, a simplex algorithm for optimization. First read in a small matrix containing coefficients that define a simplex problem to be solved:

```
>>> a := {
  {-1,-1,-1, 0, 0, 0, 0, 0, 0},
  {-1, 1, 0, 1, 0, 0, 0, 0, 5},
  { 1, 4, 0, 0, 1, 0, 0, 0, 45},
  { 2, 1, 0, 0, 0, 1, 0, 0, 27},
  { 3,-4, 0, 0, 0, 0, 1, 0, 24},
  { 0, 0, 1, 0, 0, 0, 0, 1, 4}
}
{{-1, -1, -1, 0, 0, 0, 0, 0, 0}, {-1, 1, 0, 1, 0, 0, 0, 0, 5}, {1, 4, 0, 0, 1, 0, 0,␣
↪0, 45}, {2, 1, 0, 0, 0, 1, 0, 0, 27}, {3, -4, 0, 0, 0, 0, 1, 0, 24}, {0, 0, 1, 0, 0,
↪ 0, 0, 1, 4}}
```

```
function pivot1
  input Real b[:,:];
```

(continues on next page)

(continued from previous page)

```
  input Integer p;
  input Integer q;
  output Real a[size(b,1),size(b,2)];
protected
  Integer M;
  Integer N;
algorithm
  a := b;
  N := size(a,1)-1;
  M := size(a,2)-1;
  for j in 1:N loop
    for k in 1:M loop
      if j<>p and k<>q then
       a[j,k] := a[j,k]-0.3*j;
      end if;
    end for;
  end for;
  a[p,q] := 0.05;
end pivot1;

function misc_simplex1
  input Real matr[:,:];
  output Real x[size(matr,2)-1];
  output Real z;
  output  Integer q;
  output  Integer p;
protected
  Real a[size(matr,1),size(matr,2)];
  Integer M;
  Integer N;
algorithm
  N := size(a,1)-1;
  M := size(a,2)-1;
  a := matr;
  p:=0;q:=0;
  a := pivot1(a,p+1,q+1);
  while not (q==(M) or p==(N)) loop
    q := 0;
    while not (q == (M) or a[0+1,q+1]>1) loop
      q:=q+1;
    end while;
    p := 0;
    while not (p == (N) or a[p+1,q+1]>0.1) loop
      p:=p+1;
    end while;
    if (q < M) and (p < N) and(p>0) and (q>0) then
      a := pivot1(a,p,q);
    end if;
  if(p<=0) and (q<=0) then
     a := pivot1(a,p+1,q+1);
  end if;
  if(p<=0) and (q>0) then
     a := pivot1(a,p+1,q);
  end if;
  if(p>0) and (q<=0) then
     a := pivot1(a,p,q+1);
```

(continues on next page)

(continued from previous page)

```
  end if;
  end while;
  z := a[1,M];
  x := {a[1,i] for i in 1:size(x,1)};
  for i in 1:10 loop
   for j in 1:M loop
    x[j] := x[j]+x[j]*0.01;
   end for;
  end for;
end misc_simplex1;
```

Then call the simplex algorithm implemented as the Modelica function simplex1. This function returns four results, which are represented as a tuple of four return values:

```
>>> misc_simplex1(a)
({0.05523110627056022, -1.1046221254112045, -1.1046221254112045, 0.0, 0.0, 0.0, 0.0,␣
↪0.0}, 0.0, 8, 1)
```

CHAPTER

THIRTYONE

OPENMODELICA COMPILER FLAGS

Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]

- Libraries: Fully qualified names of libraries to load before processing Model or Script. The libraries should be separated by spaces: Lib1 Lib2 ... LibN.

31.1 Options

-d, --debug

Sets debug flags. Use *--help=debug* to see available flags.

String list (default *empty*).

-h, --help

Displays the help text. Use --help=topics for more information.

String (default *empty*).

--version

Print the version and exit.

Boolean (default `false`).

--target

Sets the target compiler to use.

String (default gcc). Valid options:

- gcc
- msvc
- msvc10
- msvc12
- msvc13
- msvc15
- msvc19
- vxworks69
- debugrt

-g, --grammar

Sets the grammar and semantics to accept.

String (default Modelica).

--annotationVersion

Sets the annotation version that should be used.

String (default 3.x). Valid options:

- 1.x
- 2.x
- 3.x

--std

Sets the language standard that should be used.

String (default latest).

--showErrorMessages

Show error messages immediately when they happen.

Boolean (default `false`).

--showAnnotations

Show annotations in the flattened code.

Boolean (default `false`).

--noSimplify

Do not simplify expressions if set.

Boolean (default `false`).

--preOptModules

Sets the pre optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default normalInlineFunction,evaluateParameters,simplifyIfEquations,expandDerOperator,clockPartitioning,findStateOrd
Valid options:

- introduceOutputAliases (Introduces aliases for top-level outputs.)
- clockPartitioning (Does the clock partitioning.)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)
- comSubExp (Introduces alias assignments for variables which are assigned to simple terms i.e. a = b/c; d = b/c; --> a=d)
- dumpDAE (dumps the DAE representation of the current transformation state)
- dumpDAEXML (dumps the DAE as xml representation of the current transformation state)
- encapsulateWhenConditions (This module replaces each when condition with a boolean variable.)
- evalFunc (evaluates functions partially)
- evaluateParameters (Evaluates parameters with annotation(Evaluate=true). Use '--evaluateFinalParameters=true' or '--evaluateProtectedParameters=true' to specify additional parameters to be evaluated. Use '--replaceEvaluatedParameters=true' if the evaluated parameters should be replaced in the DAE. To evaluate all parameters in the Frontend use -d=evaluateAllParameters.)
- expandDerOperator (Expands der(expr) using Derive.differentiteExpTime.)
- findStateOrder (Sets derivative information to states.)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- normalInlineFunction (Perform function inlining for function with annotation Inline=true.)
- inputDerivativesForDynOpt (Allowed derivatives of inputs in dyn. optimization.)
- introduceDerAlias (Adds for every der-call an alias equation e.g. dx = der(x).)

- removeEqualRHS (Detects equal expressions of the form a=<exp> and b=<exp> and substitutes them to get speed up.)
- removeProtectedParameters (Replace all parameters with protected=true in the system.)
- removeSimpleEquations (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- removeUnusedParameter (Strips all parameter not present in the equations from the system.)
- removeUnusedVariables (Strips all variables not present in the equations from the system.)
- removeVerySimpleEquations ([Experimental] Like removeSimpleEquations, but less thorough. Note that this always uses the experimental new alias elimination, --removeSimpleEquations=new, which makes it unstable. In particular, MultiBody systems fail to translate correctly. It can be used for simple (but large) systems of equations.)
- replaceEdgeChange (Replace edge(b) = b and not pre(b) and change(b) = v <> pre(v).)
- residualForm (Transforms simple equations x=y to zero-sum equations 0=y-x.)
- resolveLoops (resolves linear equations in loops)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyIfEquations (Tries to simplify if equations by use of information from evaluated parameters.)
- sortEqnsVars (Heuristic sorting for equations and variables.)
- unitChecking (This module is no longer available and its use is deprecated. Use --unitChecking instead.)
- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--cheapmatchingAlgorithm

Sets the cheap matching algorithm to use. A cheap matching algorithm gives a jump start matching by heuristics.

Integer (default `3`). Valid options:

- 0 (No cheap matching.)
- 1 (Cheap matching, traverses all equations and match the first free variable.)
- 3 (Random Karp-Sipser: R. M. Karp and M. Sipser. Maximum matching in sparse random graphs.)

--matchingAlgorithm

Sets the matching algorithm to use. See *--help=optmodules* for more info.

String (default PFPlusExt). Valid options:

- BFSB (Breadth First Search based algorithm.)
- DFSB (Depth First Search based algorithm.)
- MC21A (Depth First Search based algorithm with look ahead feature.)
- PF (Depth First Search based algorithm with look ahead feature.)
- PFPlus (Depth First Search based algorithm with look ahead feature and fair row traversal.)
- HK (Combined BFS and DFS algorithm.)
- HKDW (Combined BFS and DFS algorithm.)
- ABMP (Combined BFS and DFS algorithm.)
- PR (Matching algorithm using push relabel mechanism.)
- DFSBExt (Depth First Search based Algorithm external c implementation.)
- BFSBExt (Breadth First Search based Algorithm external c implementation.)

- MC21AExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFPlusExt (Depth First Search based Algorithm with look ahead feature and fair row traversal external c implementation.)
- HKExt (Combined BFS and DFS algorithm external c implementation.)
- HKDWExt (Combined BFS and DFS algorithm external c implementation.)
- ABMPExt (Combined BFS and DFS algorithm external c implementation.)
- PRExt (Matching algorithm using push relabel mechanism external c implementation.)
- BB (BBs try.)
- SBGraph (Set-Based Graph matching algorithm for efficient array handling.)
- pseudo (Pseudo array matching that uses scalar matching and reconstructs arrays afterwards as much as possible.)

--indexReductionMethod

Sets the index reduction method to use. See *--help=optmodules* for more info.

String (default dynamicStateSelection). Valid options:

- none (Skip index reduction)
- uode (Use the underlying ODE without the constraints.)
- dynamicStateSelection (Simple index reduction method, select (dynamic) dummy states based on analysis of the system.)
- dummyDerivatives (Simple index reduction method, select (static) dummy states based on heuristic.)

--postOptModules

Sets the post optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default lateInlineFunction,wrapFunctionCalls,inlineArrayEqn,constantLinearSystem,simplifysemiLinear,removeSimpleE
Valid options:

- addScaledVars_states (added var_norm = var/nominal, where var is state)
- addScaledVars_inputs (added var_norm = var/nominal, where var is input)
- addTimeAsState (Experimental feature: this replaces each occurrence of variable time with a new introduced state $time with equation der($time) = 1.0)
- calculateStateSetsJacobians (Generates analytical jacobian for dynamic state selection sets.)
- calculateStrongComponentJacobians (Generates analytical jacobian for torn linear and non-linear strong components. By default linear components and non-linear components with user-defined function calls are skipped. See also debug flags: LSanalyticJacobian, NLSanalyticJacobian and forceNLSanalyticJacobian)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)
- constantLinearSystem (Evaluates constant linear systems (a*x+b*y=c; d*x+e*y=f; a,b,c,d,e,f are constants) at compile-time.)
- countOperations (Count the mathematical operations of the system.)
- cseBinary (Common Sub-expression Elimination)
- dumpComponentsGraphStr (Dumps the assignment graph used to determine strong components to format suitable for Mathematica)
- dumpDAE (dumps the DAE representation of the current transformation state)
- dumpDAEXML (dumps the DAE as xml representation of the current transformation state)

- evaluateParameters (Evaluates parameters with annotation(Evaluate=true). Use '--evaluateFinalParameters=true' or '--evaluateProtectedParameters=true' to specify additional parameters to be evaluated. Use '--replaceEvaluatedParameters=true' if the evaluated parameters should be replaced in the DAE. To evaluate all parameters in the Frontend use -d=evaluateAllParameters.)
- extendDynamicOptimization (Move loops to constraints.)
- generateSymbolicLinearization (Generates symbolic linearization matrices A,B,C,D for linear model:$\dot{x} = Ax + Buy = Cx + Du$)
- generateSymbolicSensitivities (Generates symbolic Sensivities matrix, where der(x) is differentiated w.r.t. param.)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- lateInlineFunction (Perform function inlining for function with annotation LateInline=true.)
- partlintornsystem (partitions linear torn systems.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires --postOptModules+=reduceDynamicOptimization.)
- relaxSystem (relaxation from gausian elemination)
- removeConstants (Remove all constants in the system.)
- removeEqualRHS (Detects equal function calls of the form a=f(b) and c=f(b) and substitutes them to get speed up.)
- removeSimpleEquations (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- removeUnusedParameter (Strips all parameter not present in the equations from the system to get speed up for compilation of target code.)
- removeUnusedVariables (Strips all variables not present in the equations from the system to get speed up for compilation of target code.)
- reshufflePost (Reshuffles algebraic loops.)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires +gDynOpt.)
- simplifyLoops (Simplifies algebraic loops. This modules requires +simplifyLoops.)
- simplifyTimeIndepFuncCalls (Simplifies time independent built in function calls like pre(param) -> param, der(param) -> 0.0, change(param) -> false, edge(param) -> false.)
- simplifysemiLinear (Simplifies calls to semiLinear.)
- solveLinearSystem (solve linear system with newton step)
- solveSimpleEquations (Solves simple equations)
- symSolver (Rewrites the ode system for implicit Euler method. This module requires +symSolver.)
- symbolicJacobian (Detects the sparse pattern of the ODE system and calculates also the symbolic Jacobian if flag '--generateDynamicJacobian=symbolic'.)
- tearingSystem (For method selection use flag tearingMethod.)

- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--simCodeTarget

Sets the target language for the code generation.

String (default C). Valid options:

- None
- C
- C+Rust
- Cpp
- omsicpp
- ExperimentalEmbeddedC
- ESP32
- JavaScript
- omsic
- XML
- MidC
- wasm-jit
- wasm

--orderConnections

Orders connect equations alphabetically if set.

Boolean (default `true`).

-t, --typeinfo

Prints out extra type information if set.

Boolean (default `false`).

-a, --keepArrays

Sets whether to split arrays or not.

Boolean (default `false`).

-m, --modelicaOutput

Enables valid modelica output for flat modelica.

Boolean (default `false`).

-q, --silent

Turns on silent mode.

Boolean (default `false`).

-n, --numProcs

Sets the number of processors to use (0=default=auto).

Integer (default `0`).

-i, --instClass

Instantiate the class given by the fully qualified path.

String (default *empty*).

-v, --vectorizationLimit

Sets the vectorization limit, arrays and matrices larger than this will not be vectorized.

Integer (default `0`).

-s, --simulationCg

Turns on simulation code generation.

Boolean (default `false`).

--evalAnnotationParams

Sets whether to evaluate parameters in annotations or not.

Boolean (default `false`).

--unitChecking

Enable unit checking.

Boolean (default `false`).

--generateLabeledSimCode

Turns on labeled SimCode generation for reduction algorithms.

Boolean (default `false`).

--reduceTerms

Turns on reducing terms for reduction algorithms.

Boolean (default `false`).

--reductionMethod

Sets the reduction method to be used.

String (default deletion). Valid options:

- deletion
- substitution
- linearization

--demoMode

Disable Warning/Error Massages.

Boolean (default `false`).

--locale

Override the locale from the environment.

String (default *empty*).

-o, --defaultOCLDevice

Sets the default OpenCL device to be used for parallel execution. 0 (the default) picks a suitable device automatically.

Integer (default `0`).

--maxTraversals

Maximal traversals to find simple equations in the acausal system.

Integer (default `2`).

--dumpTarget

Redirect the dump to file. If the file ends with .html HTML code is generated.

String (default *empty*).

--delayBreakLoop

Enables (very) experimental code to break algebraic loops using the delay() operator. Probably messes with initialization.

Boolean (default `true`).

--tearingMethod

Sets the tearing method to use. Select no tearing or choose tearing method.

String (default cellier). Valid options:

- noTearing (Deprecated, use minimalTearing.)
- minimalTearing (Minimal tearing method to only tear discrete variables.)
- omcTearing (Tearing method developed by TU Dresden: Frenkel, Schubert.)
- cellier (Tearing based on Celliers method, revised by FH Bielefeld: Täuber, Patrick)
- guruTearing (Tearing based solely on TearingSelect annotation. Forces prefer/always variables to be iteration variables.)

--tearingHeuristic

Sets the tearing heuristic to use for Cellier-tearing.

String (default MC3). Valid options:

- MC1 (Original cellier with consideration of impossible assignments and discrete Vars.)
- MC2 (Modified cellier, drop first step.)
- MC11 (Modified MC1, new last step 'count impossible assignments'.)
- MC21 (Modified MC2, new last step 'count impossible assignments'.)
- MC12 (Modified MC1, step 'count impossible assignments' before last step.)
- MC22 (Modified MC2, step 'count impossible assignments' before last step.)
- MC13 (Modified MC1, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC23 (Modified MC2, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC231 (Modified MC23, Two rounds, choose better potentials-set.)
- MC3 (Modified cellier, build sum of impossible assignment and causalizable equations for all vars, choose var with biggest sum.)
- MC4 (Modified cellier, use all heuristics, choose var that occurs most in potential sets)

--scalarizeMinMax

Scalarizes the builtin min/max reduction operators if true.

Boolean (default `false`).

--strict

Enables stricter enforcement of Modelica language rules.

Boolean (default `false`).

--scalarizeBindings

Always scalarizes bindings if set.

Boolean (default `false`).

--hpcomScheduler

Sets the scheduler for task graph scheduling (list | listr | level | levelfix | ext | metis | mcp | taskdep | tds | bls | rand | none). Default: level.

String (default level).

--hpcomCode

Sets the code-type produced by hpcom (openmp | pthreads | pthreads_spin | tbb | mpi). Default: openmp.

String (default openmp).

--rewriteRulesFile

Activates user given rewrite rules for Absyn expressions. The rules are read from the given file and are of the form rewrite(fromExp, toExp);

String (default *empty*).

--replaceHomotopy

Replaces homotopy(actual, simplified) with the actual expression or the simplified expression. Good for debugging models which use homotopy. The default is to not replace homotopy.

String (default none). Valid options:

- none (Default, do not replace homotopy.)
- actual (Replace homotopy(actual, simplified) with actual.)
- simplified (Replace homotopy(actual, simplified) with simplified.)

--generateDynamicJacobian

Select how Jacobian matrix is generated, where der(x) is differentiated w.r.t. x.

String (default numeric). Valid options:

- none (Does not generate Jacobian. For use with explicit solvers.)
- numeric (Generates sparsity pattern for numeric Jacobian.)
- symbolic (Generates symbolic Jacobian. Used by dassl or ida solver with simulation flag '-jacobian'.)
- symbolicadjoint (Generates adjoint Jacobian symbolically.)
- bidirectional (Generates bidirectional Jacobian using star bicoloring (ColPack). Combines forward and adjoint modes.)

--generateSymbolicLinearization

Generates symbolic linearization matrices A,B,C,D for linear model:
$\dot{x} = Ax + Bu\ y = Cx + Du$

Boolean (default `false`).

--intEnumConversion

Allow Integer to enumeration conversion.

Boolean (default `false`).

--profiling

Sets the profiling level to use. Profiled equations and functions record execution time and count for each time step taken by the integrator.

String (default none). Valid options:

- none (Generate code without profiling)
- blocks (Generate code for profiling function calls as well as linear and non-linear systems of equations)
- blocks+html (Like blocks, but also run xsltproc and gnuplot to generate an html report)
- all (Generate code for profiling of all functions and equations)

- all_perf (Generate code for profiling of all functions and equations with additional performance data using the papi-interface (cpp-runtime))
- all_stat (Generate code for profiling of all functions and equations with additional statistics (cpp-runtime))

--reshuffle

sets tolerance of reshuffling algorithm: 1: conservative, 2: more tolerant, 3 resolve all

Integer (default `1`).

--gDynOpt

Generate dynamic optimization problem based on annotation approach.

Boolean (default `false`).

--maxSizeSolveLinearSystem

Max size for solveLinearSystem.

Integer (default `0`).

--cppFlags

Sets extra flags for compilation with the C++ compiler (e.g. +cppFlags=-O3,-Wall)

String list (default).

--removeSimpleEquations

Specifies method that removes simple equations.

String (default default). Valid options:

- none (Disables module)
- default (Performs alias elimination and removes constant variables. Default case uses in preOpt phase the fastAcausal and in postOpt phase the causal implementation.)
- causal (Performs alias elimination and removes constant variables. Causal implementation.)
- fastAcausal (Performs alias elimination and removes constant variables. fastImplementation fastAcausal.)
- allAcausal (Performs alias elimination and removes constant variables. Implementation allAcausal.)
- new (New implementation (experimental))

--dynamicTearing

Activates dynamic tearing (TearingSet can be changed automatically during runtime, strict set vs. casual set.)

String (default false). Valid options:

- false (No dynamic tearing.)
- true (Dynamic tearing for linear and nonlinear systems.)
- linear (Dynamic tearing only for linear systems.)
- nonlinear (Dynamic tearing only for nonlinear systems.)

--symSolver

Activates symbolic implicit solver (original system is not changed).

String (default none).

--loop2con

Specifies method that transform loops in constraints. hint: using initial guess from file!

String (default none). Valid options:

- none (Disables module)
- lin (linear loops --> constraints)

- noLin (no linear loops --> constraints)
- all (loops --> constraints)

--forceTearing

Use tearing set even if it is not smaller than the original component.

Boolean (default `false`).

--simplifyLoops

Simplify algebraic loops.

Integer (default `0`). Valid options:

- 0 (do nothing)
- 1 (special modification of residual expressions)
- 2 (special modification of residual expressions with helper variables)

--recursiveTearing

Inline and repeat tearing.

Integer (default `0`). Valid options:

- 0 (do nothing)
- 1 (linear tearing set of size 1)
- 2 (linear tearing)

--flowThreshold

Sets the minium threshold for stream flow rates

Real (default `1e-7`).

--matrixFormat

Sets the matrix format type in cpp runtime which should be used (dense | sparse). Default: dense.

String (default dense).

--partlintorn

Sets the limit for partitionin of linear torn systems.

Integer (default `0`).

--initOptModules

Sets the initialization optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default simplifyComplexFunction,tearingSystem,solveSimpleEquations,calculateStrongComponentJacobians,simplifyAl
Valid options:

- calculateStrongComponentJacobians (Generates analytical jacobian for torn linear and non-linear strong components. By default linear components and non-linear components with user-defined function calls are skipped. See also debug flags: LSanalyticJacobian, NLSanalyticJacobian and forceNLSanalyticJacobian)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- constantLinearSystem (Evaluates constant linear systems (a*x+b*y=c; d*x+e*y=f; a,b,c,d,e,f are constants) at compile-time.)
- extendDynamicOptimization (Move loops to constraints.)
- generateHomotopyComponents (Finds the parts of the DAE that have to be handled by the homotopy solver and creates a strong component out of it.)

- inlineHomotopy (Experimental: Inlines the homotopy expression to allow symbolic simplifications.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires --postOptModules+=reduceDynamicOptimization.)
- replaceHomotopyWithSimplified (Replaces the homotopy expression homotopy(actual, simplified) with the simplified part.)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires +gDynOpt.)
- simplifyLoops (Simplifies algebraic loops. This modules requires +simplifyLoops.)
- solveSimpleEquations (Solves simple equations)
- tearingSystem (For method selection use flag tearingMethod.)
- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--maxMixedDeterminedIndex

Sets the maximum mixed-determined index that is handled by the initialization.

Integer (default `10`).

--useLocalDirection

Keeps the input/output prefix for all variables in the flat model, not only top-level ones.

Boolean (default `false`).

--defaultOptModulesOrdering

If this is activated, then the specified pre-/post-/init-optimization modules will be rearranged to the recommended ordering.

Boolean (default `true`).

--preOptModules+

Enables additional pre-optimization modules, e.g. --preOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--preOptModules-

Disables a list of pre-optimization modules, e.g. --preOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--postOptModules+

Enables additional post-optimization modules, e.g. --postOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--postOptModules-

Disables a list of post-optimization modules, e.g. --postOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--initOptModules+

Enables additional init-optimization modules, e.g. --initOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--initOptModules-

Disables a list of init-optimization modules, e.g. --initOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--instCacheSize

Sets the size of the internal hash table used for instantiation caching.

Integer (default `25343`).

--maxSizeLinearTearing

Sets the maximum system size for tearing of linear systems (default 200).

Integer (default `200`).

--maxSizeNonlinearTearing

Sets the maximum system size for tearing of nonlinear systems (default 10000).

Integer (default `10000`).

--noTearingForComponent

Deactivates tearing for the specified components. Use '-d=tearingdump' to find out the relevant indexes.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--daeMode

Generates code to simulate models in DAE mode. The whole system is passed directly to the DAE solver SUNDIALS/IDA and no algebraic solver is involved in the simulation process.

Boolean (default `false`).

--inlineMethod

Sets the inline method to use. replace : This method inlines by replacing in place all expressions. Might lead to very long expression. append : This method inlines by adding additional variables to the whole system. Might lead to much bigger system.

String (default replace).

--setTearingVars

Sets the tearing variables by its strong component indexes. Use '-d=tearingdump' to find out the relevant indexes. Use following format: '--setTearingVars=(sci,n,t1,...,tn)*', with sci = strong component index, n = number of tearing variables, t1,...tn = tearing variables. E.g.: '--setTearingVars=4,2,3,5' would select variables 3 and 5 in strong component 4.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--setResidualEqns

Sets the residual equations by its strong component indexes. Use '-d=tearingdump' to find out the relevant indexes for the collective equations. Use following format: '--setResidualEqns=(sci,n,r1,...,rn)*', with sci = strong component index, n = number of residual equations, r1,...rn = residual equations. E.g.: '--setResidualEqns=4,2,3,5' would select equations 3 and 5 in strong component 4. Only works in combination with 'setTearingVars'.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--ignoreCommandLineOptionsAnnotation

Ignores the command line options specified as annotation in the class.

Boolean (default `false`).

--calculateSensitivities

Generates sensitivities variables and matrices.

Boolean (default `false`).

-r, --alarm

Sets the number of seconds until omc times out. Used by the testing framework to terminate infinite running processes. The running command is first asked to stop, so that omc can still report how far it got; it exits outright a tenth of that time later (at least 5 and at most 60 seconds) if the command has no cancellation point to stop at.

Integer (default `0`).

--totalTearing

Activates total tearing (determination of all possible tearing sets) for the specified components. Use '-d=tearingdump' to find out the relevant indexes.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--ignoreSimulationFlagsAnnotation

Ignores the simulation flags specified as annotation in the class.

Boolean (default `false`).

--dynamicTearingForInitialization

Enable Dynamic Tearing also for the initialization system.

Boolean (default `false`).

--preferTVarsWithStartValue

Prefer tearing variables with start value for initialization.

Boolean (default `true`).

--equationsPerFile

Generate code for at most this many equations per C-file (partially implemented in the compiler).

Integer (default `500`).

--evaluateFinalParameters

Evaluates all the final parameters in addition to parameters with annotation(Evaluate=true).

Boolean (default `false`).

--evaluateProtectedParameters

Evaluates all the protected parameters in addition to parameters with annotation(Evaluate=true).

Boolean (default `false`).

--replaceEvaluatedParameters

Replaces all the evaluated parameters in the DAE.

Boolean (default `true`).

--condenseArrays

Sets whether array expressions containing function calls are condensed or not.

Boolean (default `true`).

--wfcAdvanced

wrapFunctionCalls ignores more then default cases, e.g. exp, sin, cos, log, (experimental flag)

Boolean (default `false`).

--tearingStrictness

Sets the strictness of the tearing method regarding the solvability restrictions.

String (default strict). Valid options:

- casual (Loose tearing rules using ExpressionSolve to determine the solvability instead of considering the partial derivative. Allows to solve for everything that is analytically possible. This could lead to singularities during simulation.)
- strict (Robust tearing rules by consideration of the partial derivative. Allows to divide by parameters that are not equal to or close to zero.)
- veryStrict (Very strict tearing rules that do not allow to divide by any parameter. Use this if you aim at overriding parameters after compilation with values equal to or close to zero.)

--interactive

Sets the interactive mode for omc.

String (default none). Valid options:

- none (do nothing)
- tcp (Starts omc as a server listening on the socket interface.)
- zmq (Starts omc as a ZeroMQ server listening on the socket interface.)

-z, --zeroMQFileSuffix

Sets the file suffix for zeroMQ port file if --interactive=zmq is used.

String (default *empty*).

--homotopyApproach

Sets the homotopy approach.

String (default equidistantGlobal). Valid options:

- equidistantLocal (Local homotopy approach with equidistant lambda steps. The homotopy parameter only effects the local strongly connected component.)
- adaptiveLocal (Local homotopy approach with adaptive lambda steps. The homotopy parameter only effects the local strongly connected component.)
- equidistantGlobal (Default, global homotopy approach with equidistant lambda steps. The homotopy parameter effects the entire initialization system.)
- adaptiveGlobal (Global homotopy approach with adaptive lambda steps. The homotopy parameter effects the entire initialization system.)

--ignoreReplaceable

Sets whether to ignore replaceability or not when redeclaring.

Boolean (default `false`).

--postOptModulesDAE

Sets the optimization modules for the DAEmode in the back end. See *--help=optmodules* for more info.

String list (default lateInlineFunction,wrapFunctionCalls,simplifysemiLinear,simplifyComplexFunction,removeConstants,simplifyT

--evalLoopLimit

The loop iteration limit used when evaluating constant function calls.

Integer (default `100000`).

--evalRecursionLimit

The recursion limit used when evaluating constant function calls.

Integer (default `256`).

--singleInstanceAglSolver

Sets to instantiate only one algebraic loop solver all algebraic loops

Boolean (default `false`).

--showStructuralAnnotations

Show annotations affecting the solution process in the flattened code.

Boolean (default `false`).

--initialStateSelection

Activates the state selection inside initialization to avoid singularities.

Boolean (default `false`).

--linearizationDumpLanguage

Sets the target language for the produced code of linearization.

String (default none). Valid options:

- none (Don't generate code for linearization.)
- modelica (Generate linearized Modelica model.)
- matlab (Generate matlab function that returns linearization matrices A,B,C,D.)
- julia (Generate julia function that returns linearization matrices A,B,C,D.)
- python (Generate python function that returns linearization matrices A,B,C,D.)

--noASSC

Disables analytical to structural singularity conversion.

Boolean (default `false`).

--fullASSC

Enables full equation replacement for BLT transformation from the ASSC algorithm.

Boolean (default `false`).

--realASSC

Enables the ASSC algorithm to evaluate real valued coefficients (usually only integers).

Boolean (default `false`).

--initASSC

Enables the ASSC algorithm for initialization.

Boolean (default `false`).

--maxSizeASSC

Sets the maximum system size for the analytical to structural conversion algorithm (default 200).

Integer (default `200`).

-f, --baseModelica

Outputs experimental Base Modelica.

Boolean (default `false`).

--fmiFilter

Specifies which model variables are exposed by the modelDescription.xml

String (default protected). Valid options:

- none (All variables are exposed, even variables introduced by the symbolic transformations. This is mainly for debugging purposes.)
- internal (All model variables are exposed, including protected ones. Variables introduced by the symbolic transformations are filtered out, with minor exceptions, e.g. for state sets.)
- protected (All public model variables are exposed. Internal and protected variables are filtered out, with small exceptions, e.g. for state sets.)
- blackBox (Only the interface is exposed. All other variables are hidden or exposed with concealed names.)

--fmiSources

Defines how much source code FMUs will be exported with. --fmiFilter=blackBox might override this, because black box FMUs do never contain their source code.

String (default false). Valid options:

- false (The FMU carries no sources at all [default].)
- true (Same as slim.)
- slim (The FMU carries the sources it was built from. Under --simCodeTarget=C+Rust that is a mix of C and Rust, and rebuilding needs cargo to fetch the Rust dependencies from crates.io.)
- full (As slim, plus a vendored copy of the Rust dependencies, so the FMU rebuilds without network access. Only differs from slim under --simCodeTarget=C+Rust.)

--fmiFlags

Add simulation flags to FMU. Will create <fmiPrefix>_flags.json in resources folder with given flags. Use --fmiFlags or --fmiFlags=none to disable [default]. Use --fmiFlags=default for the default simulation flags. To pass flags use e.g. --fmiFlags=s:cvode,nls:homotopy or --fmiFlags=path/to/yourFlags.json.

String list (default *empty*).

--fmuCMakeBuild

Configured and build FMU with CMake if true.

Boolean (default `true`).

--newBackend

Activates experimental new backend for better array handling. This also activates the new frontend. [WIP]

Boolean (default `false`).

--parmodauto

Experimental: Enable parallelization of independent systems of equations in the translated model. Only works on Linux systems.

Boolean (default `false`).

--interactivePort

Sets the port used by the interactive server.

Integer (default `0`).

--allowNonStandardModelica

Flags to allow non-standard Modelica.

String list (default *empty*). Valid options:

- nonStdMultipleExternalDeclarations (Allow several external declarations in functions.See: https://specification.modelica.org/maint/3.5/functions.html#function-as-a-specialized-class)
- nonStdEnumerationAsIntegers (Allow enumeration as integer without casting via Integer(Enum).See: https://specification.modelica.org/maint/3.5/class-predefined-types-and-declarations.html#type-conversion-of-enumeration-values-to-string-or-integer)

- nonStdIntegersAsEnumeration (Allow integer as enumeration without casting via Enum(Integer).See: https://specification.modelica.org/maint/3.5/class-predefined-types-and-declarations.html#type-conversion-of-integer-to-enumeration-values)
- nonStdDifferentCaseFileVsClassName (Allow directory or file with different case in the name than the contained class name.See: https://specification.modelica.org/maint/3.5/packages.html#mapping-package-class-structures-to-a-hierarchical-file-system)
- nonStdTopLevelOuter (Allow top level outer.See: https://specification.modelica.org/maint/3.6/scoping-name-lookup-and-flattening.html#S4.p1)
- protectedAccess (Allow access of protected elements)
- reinitInAlgorithms (Allow reinit in algorithm sections)
- unbalancedModel (Allow models to be locally unbalanced and to have unbalanced connectors)
- implicitParameterStartAttribute (Allow fixed parameters with no binding or start attribute)
- initialSimplified (Allow use of experimental operator *initialSimplified()*)
- illegalConditionalContext (Allow use of components with false conditions in illegal contexts)

--exportClocksInModelDescription

exports clocks in modeldescription.xml for fmus, The default is false.

Boolean (default `false`).

--linkType

Sets the link type for the simulation executable. dynamic: libraries are dynamically linked; the executable is built very fast but is not portable because of DLL dependencies. static: libraries are statically linked; the executable is built more slowly but it is portable and dependency-free.

String (default dynamic).

--tearingAlwaysDer

Always choose state derivatives as iteration variables in strong components.

Boolean (default `false`).

--dumpFlatModel

Dumps the flat model at the given stages of the frontend.

String list (default all). Valid options:

- flatten (After flattening but before connection handling.)
- connections (After connection handling.)
- eval (After evaluating constants.)
- simplify (After model simplification.)
- scalarize (After scalarizing arrays.)
- translateResidualsDAE (Show the result of the translateResidualsDAE API.)

-u, --simulation

Simulates the last model in the given Modelica file.

Boolean (default `false`).

--obfuscate

Obfuscates identifiers in the simulation model

String (default none). Valid options:

- none (No obfuscation.)

- encrypted (Obfuscates protected variables in encrypted models)
- protected (Obfuscates protected variables in all models.)
- full (Obfuscates everything.)

--fmuRuntimeDepends

Defines if runtime library dependencies are included in the FMU. Only used when compiler flag fmuCMakeBuild=true.

String (default default). Valid options:

- default (Depending on CMake version. If CMake version >= 3.21 use "modelica", otherwise use "none")
- none (No runtime library dependencies are copied into the FMU.)
- modelica (All modelica runtime library dependencies are copied into the FMU.System librarys located in '/lib*', '/usr/lib*' and '/usr/local/lib*' are excluded.Needs --fmuCMakeBuild=true and CMake version >= 3.21.)
- all (All runtime library dependencies are copied into the FMU.System librarys are copied as well.Needs --fmuCMakeBuild=true and CMake version >= 3.21.)

--frontendInline

Enables inlining of functions in the frontend.

Boolean (default `false`).

--exposeLocalIOs

Introduces top-level inputs/outputs for unconnected input/output connectors at requested levels, provided they are public, 0 meaning top-level (standard Modelica), 1 inputs/outputs of top-level components, >1 going deeper. This flag is particularly useful for FMI export.

Integer (default `0`).

--baseModelicaFormat

Formatting options for Base Modelica

String list (default *empty*). Valid options:

- scalarized (Include subscripts in the quoted identifiers ('a[1].x[3]').)
- partiallyScalarized (Include subscripts in the quoted identifiers, except for the last name ('a[1].x'[3]).)
- nonScalarized (Don't include subscripts in the quoted identifiers ('a'[1].'x'[3]).)
- withRecords (Keep records and don't expand them.)
- withoutRecords (Expand records into separate components.)
- showConfidence (Add comments that show confidence numbers for binding equations.)

--baseModelicaOptions

Enables optional Base Modelica options.

String list (default *empty*). Valid options:

- moveBindings (Moves movable binding equations to normal equations.)
- scalarize (Fully scalarize the Base Modelica model.)
- inlineFunctions (Inline all functions.)

--debugFollowEquations

Takes a list of equation names and prints the corresponding equations after each stage of the backend process.

String list (default *empty*).

--maxSizeLinearization

Sets the maximum system size for which linearization code is generated.

Integer (default `1000`).

--resizableArrays

Assumes all arrays are resizable. Only works with the new backend --newBackend.

Boolean (default `false`).

--evaluateStructuralParameters

Sets which structural parameters are evaluated by the frontend.

String (default all). Valid options:

- all (Evaluates all structural parameters)
- strictlyNecessary (Evaluates only structural parameters strictly required by the frontend)

--loadMissingLibraries

Automatically try to load a matching library if a name can't be found during name lookup.

Boolean (default `true`).

--causalizeDaeMode

The system is partially causalized and simple assignments are generated for equations that can be solved explicitly. Only works with --daeMode.

Boolean (default `true`).

--simCodeScalarize

Scalarizes variables during simcode phase.

Boolean (default `true`).

--cmd

Executes the string argument as a script before any other operation.

String (default *empty*).

--moo

Generate code for dynamic optimization library MOO.

Boolean (default `false`).

--fmiExtraAnnotations

Export annotations matching the given regex to extra/org.openmodelica/modelAnnotations.json.

String (default *empty*).

--interactiveDumpFormat

Selects the format interactive API calls use to print result values. default : The OpenModelica textual value format. json : JSON, for programmatic consumers such as the web clients.

String (default default).

--export-fmu

Export the model as an FMU after loading. Combine with --fmuType and --fmuPlatforms to configure the export.

Boolean (default `false`).

--fmuType

FMI type for FMU export: me, cs, me_cs, se.

String list (default me).

--fmuPlatforms

Platforms for FMU export, comma-separated (e.g. static,dynamic).

String (default static).

--fmiVersion

FMI version for FMU export: 1.0, 2.0, 3.0.

String (default 2.0). Valid options:

- 1.0
- 2.0
- 3.0

--tearingCostMargin

How much cheaper solving a linear system untorn has to be estimated before its tearing set is dropped. Raise it to keep tearing systems the estimate would give up on, lower it towards 0 to tear less (default 2).

Real (default `2.0`).

--tplOutputDir

Directory the .mo generated from a Susan .tpl is written to. Empty (the default) writes it next to the .tpl, which is how the templates used to be generated into the source tree.

String (default *empty*).

--fmuDirectory

Write an exported FMU as an unzipped directory named <prefix>.fmu rather than a zip file. An importer that reads a directory (OpenModelica's own does) then pays neither the compression nor the extraction, which for a wasm FMU carrying a precompiled artifact is most of what packing it costs. wasm FMUs only.

Boolean (default `false`).

31.2 Debug flags

The debug flag takes a comma-separated list of flags which are used by the compiler for debugging or experimental purposes. Flags prefixed with "-" or "no" will be disabled. The available flags are (+ are enabled by default, - are disabled):

Cache **(default: on)**
: Turns off the instantiation cache.

LSanalyticJacobian **(default: off)**
: Enables analytical jacobian for linear strong components. Defaults to false

NLSanalyticJacobian **(default: on)**
: Enables analytical jacobian for non-linear strong components without user-defined function calls, for that see forceNLSanalyticJacobian

acceptTooManyFields **(default: off)**
: Accepts passing records with more fields than expected to a function. This is not allowed, but is used in Fluid.Dissipation. See https://trac.modelica.org/Modelica/ticket/1245 for details.

aliasConflicts **(default: off)**
: Dumps alias sets with different start or nominal values.

arrayConnect **(default: off)**
: Use experimental array connection handler.

backendKeepEnv **(default: on)**
: When enabled, the environment is kept when entering the backend, which enables CevalFunction (function interpretation) to work. This module not essential for the backend to function in most cases, but can improve simulation performance by evaluating functions. The drawback to keeping the environment graph in memory is that it is huge (~80% of the total memory in use when returning the frontend DAE).

backendReduceDAE **(default: off)**
Prints all Reduce DAE debug information.

backenddaeinfo **(default: off)**
Enables dumping of back-end information about system (Number of equations before back-end,...).

bltdump **(default: off)**
Dumps information from index reduction.

bltmatrixdump **(default: off)**
Dumps the blt matrix in html file. IE seems to be very good in displaying large matrices.

buildExternalLibs **(default: on)**
Use the autotools project in the Resources folder of the library to build missing external libraries.

ceval **(default: off)**
Prints extra information from Ceval.

cgraph **(default: off)**
Prints out connection graph information.

cgraphGraphVizFile **(default: off)**
Generates a graphviz file of the connection graph.

cgraphGraphVizShow **(default: off)**
Displays the connection graph with the GraphViz lefty tool.

checkASUB **(default: off)**
Prints out a warning if an ASUB is created from a CREF expression.

checkBackendDae **(default: off)**
Do some simple analyses on the datastructure from the frontend to check if it is consistent.

checkDAECrefType **(default: off)**
Enables extra type checking for cref expressions.

checkDefUse **(default: off)**
Warns about variables in functions that cannot statically be proven to be defined (given a value) before they are used, e.g. variables only assigned on some control flow paths. Per the Modelica specification using an uninitialized variable is an error.

checkSimplify **(default: off)**
Enables checks for expression simplification and prints a notification whenever an undesirable transformation has been performed.

combineSubscripts **(default: off)**
Move all subscripts to the end of component references.

constjac **(default: off)**
solves linear systems with constant Jacobian and variable b-Vector symbolically

countOperations **(default: off)**
Count operations.

daedumpgraphv **(default: off)**
Dumps the DAE in graphviz format.

dataReconciliation **(default: off)**
Dumps all the dataReconciliation extraction algorithm procedure

debugAdjoint **(default: off)**
Dumps debug output for the adjoint differentiation process in the new backend.

debugAlgebraicLoopsJacobian **(default: off)**
Dumps debug output while creating symbolic jacobians for non-linear systems.

debugAlias **(default: off)**
Dumps some information about the process of removeSimpleEquations.

debugDAEmode **(default: off)**
Dump debug output for the DAEmode.

debugDifferentiation **(default: off)**
Dumps debug output for the differentiation process.

debugDifferentiationVerbose **(default: off)**
Dumps verbose debug output for the differentiation process.

disableColoring **(default: off)**
Disables coloring algorithm while spasity detection.

disableDirectionalDerivatives **(default: on)**
For FMI 2.0 only dependecy analysis will be perform.

disableFMIDependency **(default: off)**
Disables the dependency analysis and generation for FMI 2.0.

disableJacsforSCC **(default: off)**
Disables calculation of jacobians to detect if a SCC is linear or non-linear. By disabling all SCC will handled like non-linear.

disableRecordConstructorOutput **(default: off)**
Disables output of record constructors in the flat code.

disableSingleFlowEq **(default: off)**
Disables the generation of single flow equations.

disableStartCalc **(default: off)**
Deactivates the pre-calculation of start values during compile-time.

disableWindowsPathCheckWarning **(default: off)**
Disables warnings on Windows if OPENMODELICAHOME/MinGW is missing.

discreteinfo **(default: off)**
Enables dumping of discrete variables. Extends -d=backenddaeinfo.

dummyselect **(default: off)**
Dumps information from dummy state selection heuristic.

dump **(default: off)**
Dumps the absyn representation of a program.

dumpASSC **(default: off)**
Dumps the conversion process of analytical to structural singularities.

dumpBackendClocks **(default: off)**
Dumps times for each backend module (only new backend).

dumpBackendInline **(default: off)**
Dumps debug output while inline function.

dumpBackendInlineVerbose **(default: off)**
Dumps debug output while inline function.

dumpBindings **(default: off)**
Dumps information about the equations created from bindings.

dumpCSE **(default: off)**
Additional output for CSE module.

dumpCSE_verbose **(default: off)**
Additional output for CSE module.

dumpCheckModel **(default: off)**
Dumps the variables and equations found by checkModel.

dumpConstrepl **(default: off)**
Dump the found replacements for constants.

dumpConversionRules **(default: off)**
Dumps the rules when converting a package using a conversion script.

dumpEArepl **(default: off)**
Dump the found replacements for evaluate annotations (evaluate=true) parameters.

dumpEncapsulateConditions **(default: off)**
Dumps the results of the preOptModule encapsulateWhenConditions.

dumpEqInUC **(default: off)**
Dumps all equations handled by the unit checker.

dumpEqUCStruct **(default: off)**
Dumps all the equations handled by the unit checker as tree-structure.

dumpEvents **(default: off)**
Dumps information about the detected event functions.

dumpExcludedSymJacExps **(default: off)**
This flags dumps all expression that are excluded from differentiation of a symbolic Jacobian.

dumpFPrepl **(default: off)**
Dump the found replacements for final parameters.

dumpFunctions **(default: off)**
Add functions to backend dumps.

dumpHomotopy **(default: off)**
Dumps the results of the postOptModule optimizeHomotopyCalls.

dumpInlineSolver **(default: off)**
Dumps the inline solver equation system.

dumpJL **(default: off)**
Dumps the absyn representation of a program as a Julia representation

dumpLoops **(default: off)**
Dumps loop equation.

dumpLoopsVerbose **(default: off)**
Dumps loop equation and enhanced adjacency matrix.

dumpPPrepl **(default: off)**
Dump the found replacements for protected parameters.

dumpParamrepl **(default: off)**
Dump the found replacements for remove parameters.

dumpRecursiveTearing **(default: off)**
Dump between steps of recursiveTearing

dumpResizable **(default: off)**
Dumps information about resizable paremeter handling.

dumpSCCGraphML **(default: off)**
Dumps graphml files with the strongly connected components.

dumpSetBasedGraphs **(default: off)**
Dumps information about set based graphs for efficient array handling (only new frontend and new backend).

dumpSimCode **(default: off)**
Dumps the simCode model used for code generation.

dumpSimplify **(default: off)**
Dumps expressions before and after simplification.

dumpSimplifyLoops **(default: off)**
Dump between steps of simplifyLoops

dumpSlice **(default: off)**
Dumps information about the slicing process (pseudo-array causalization).

dumpSolve **(default: off)**
Dumps information about equation solving.

dumpSortEqnsAndVars **(default: off)**
Dumps debug output for the modules sortEqnsVars.

dumpSorting **(default: off)**
Dumps information about the process of sorting.

dumpSparsePattern **(default: off)**
Dumps sparse pattern with coloring used for simulation.

dumpSparsePatternVerbose **(default: off)**
Dumps in verbose mode sparse pattern with coloring used for simulation.

dumpSynchronous **(default: off)**
Dumps information of the clock partitioning.

dumpUnits **(default: off)**
Dumps all the calculated units.

dumpdaelow **(default: off)**
Dumps the equation system at the beginning of the back end.

dumpdgesv **(default: off)**
Enables dumping of the information whether DGESV is used to solve linear systems.

dumpeqninorder **(default: off)**
Enables dumping of the equations in the order they are calculated.

dumpindxdae **(default: off)**
Dumps the equation system after index reduction and optimization.

dumpinitialsystem **(default: off)**
Dumps the initial equation system.

dumprepl **(default: off)**
Dump the found replacements for simple equation removal.

dynload **(default: off)**
Display debug information about dynamic loading of compiled functions.

evalFuncDump **(default: off)**
dumps debug information about the function evaluation

evalOutputOnly **(default: off)**
Generates equations to calculate top level outputs only.

evalParameterDump **(default: off)**
Dumps information for evaluating parameters.

evalfunc **(default: on)**
Turns on/off symbolic function evaluation.

evaluateAllParameters **(default: off)**
Evaluates all parameters if set, except the ones that have annotation(Evaluate = false).

events **(default: on)**
Turns on/off events handling.

execHash **(default: off)**
Measures the time it takes to hash all simcode variables before code generation.

execstat **(default: off)**
Prints out execution statistics for the compiler.

execstatGCcollect **(default: off)**
When running execstat, also perform an extra full garbage collection.

experimentalReductions **(default: off)**
Turns on custom reduction functions (OpenModelica extension).

failtrace **(default: off)**
Sets whether to print a failtrace or not.

flowAliasElimination **(default: off)**
Enables simple alias elimination of flow variables in stream connectors.

fmuExperimental **(default: off)**
Adds features to the FMI export that are considered experimental as of now: fmi2GetSpecificDerivatives, canGetSetFMUState, canSerializeFMUstate

force-fmi-attributes **(default: off)**
Force to export all fmi attributes to the modelDescription.xml, including those which have default values

forceNLSanalyticJacobian **(default: off)**
Forces calculation analytical jacobian also for non-linear strong components with user-defined functions.

forceScalarize **(default: off)**
Forces scalarization to be done when it would normally be automatically disabled.

frontEndUnitCheck **(default: off)**
Checks the consistency of units in equation.

gcProfiling **(default: off)**
Prints garbage collection stats to standard output.

gen **(default: off)**
Turns on/off dynamic loading of functions that are compiled during translation. Only enable this if external functions are needed to calculate structural parameters or constants.

gendebugsymbols **(default: off)**
Generate code with debugging symbols.

generateCodeCheat **(default: off)**
Used to generate code for the bootstrapped compiler.

graphInst **(default: off)**
Do graph based instantiation.

graphInstGenGraph **(default: off)**
Dumps a graph of the program. Use with -d=graphInst

graphInstRunDep **(default: off)**
Run scode dependency analysis. Use with -d=graphInst

graphml **(default: off)**
Dumps .graphml files for the bipartite graph after Index Reduction and a task graph for the SCCs. Can be displayed with yEd.

graphviz **(default: off)**
Dumps the absyn representation of a program in graphviz format.

graphvizDump **(default: off)**
Activates additional graphviz dumps (as .dot files). It can be used in addition to one of the following flags: {dumpdaelow|dumpinitialsystems|dumpindxdae}.

hardcodedStartValues **(default: off)**
Embed the start values of variables and parameters into the c++ code and do not read it from xml file.

hpcom **(default: off)**
Enables parallel calculation based on task-graphs.

hpcomDump **(default: off)**
Dumps additional information on the parallel execution with hpcom.

hpcomMemoryOpt **(default: off)**
Optimize the memory structure regarding the selected scheduler

ignoreCycles **(default: off)**
Ignores cycles between constant/parameter components.

implOde **(default: off)**
activates implicit codegen

infoXmlOperations **(default: off)**
Enables output of the operations in the _info.xml file when translating models.

initialization **(default: off)**
Shows additional information from the initialization process.

inlineFunctions **(default: on)**
Controls if function inlining should be performed.

inlineSolver **(default: off)**
Generates code for inline solver.

instance **(default: off)**
Prints extra failtrace from InstanceHierarchy.

interactive **(default: off)**
Starts omc as a server listening on the socket interface.

interactivedump **(default: off)**
Prints out debug information for the interactive server.

iterationVars **(default: off)**
Shows a list of all iteration variables.

listAppendWrongOrder **(default: on)**
Print notifications about bad usage of listAppend.

lookup **(default: off)**
Print extra failtrace from lookup.

mergeAlgSections **(default: off)**
Disables coloring algorithm while sparsity detection.

mergeComponents **(default: off)**
Enables automatic merging of components into arrays.

metaModelicaRecordAllocWords **(default: off)**
Instrument the source code to record memory allocations (requires run-time and generated files compiled with -DOMC_RECORD_ALLOC_WORDS).

multirate **(default: off)**
The solver can switch partitions in the system.

newInst **(default: on)**
Enables new instantiation phase.

nfAPI **(default: on)**
Enables experimental new instantiation use in the OMC API.

nfAPIDynamicSelect **(default: off)**
Show DynamicSelect(static, dynamic) in annotations. Default to false and will select the first (static) expression

nfAPINoise **(default: off)**
Enables error display for the experimental new instantiation use in the OMC API.

nfEvalConstArgFuncs **(default: on)**
Evaluate all functions with constant arguments in the new frontend.

nfExpandFuncArgs **(default: off)**
Expand all function arguments in the new frontend.

nfExpandOperations **(default: on)**
Expand all unary/binary operations to scalar expressions in the new frontend.

nfScalarize **(default: on)**
Run scalarization in NF, default true.

oldFrontEndUnitCheck **(default: off)**
Checks the consistency of units in equation (for the old front-end).

omedit **(default: off)**
Set by OMEdit, so the compiler can emit output only a GUI consumes.

optdaedump **(default: off)**
Dumps information from the optimization modules.

parallelCodegen **(default: on)**
Enables code generation in parallel (disable this if compiling a model causes you to run out of RAM).

paramdlowdump **(default: off)**
Enables dumping of the parameters in the order they are calculated.

partitionInitialization **(default: on)**
This flag controls if partitioning is applied to the initialization system.

patternmAllInfo **(default: off)**
Adds notifications of all pattern-matching optimizations that are performed.

patternmDeadCodeElimination **(default: on)**
Performs dead code elimination in match-expressions.

patternmMoveLastExp **(default: on)**
Optimization that moves the last assignment(s) into the result of a match-expression. For example: equation c = fn(b); then c; => then fn(b);

patternmSkipFilterUnusedBindings (default: off)

printRecordTypes **(default: off)**
Prints out record types as part of the flat code.

printStructuralParameters **(default: off)**
Prints the structural parameters identified by the front-end

pthreads **(default: off)**
Experimental: Unused parallelization.

relidx **(default: off)**
Prints out debug information about relations, that are used as zero crossings.

relocatableFunctions **(default: off)**
Generates relocatable code: all functions become function pointers and can be replaced at run-time.

reportSerializedSize **(default: off)**
Reports serialized sizes of various data structures used in the compiler.

reshufflePost **(default: off)**
Reshuffles the systems of equations.

resolveLoopsDump **(default: off)**
Debug Output for ResolveLoops Module.

rml **(default: off)**
Converts Modelica-style arrays to lists.

runtimeStaticLinking **(default: off)**
Use the static simulation runtime libraries (C++ simulation runtime).

scodeDep **(default: on)**
Does scode dependency analysis prior to instantiation. Defaults to true.

semiLinear **(default: off)**
Enables dumping of the optimization information when optimizing calls to semiLinear.

shortOutput **(default: off)**
Enables short output of the simulate() command. Useful for tools like OMNotebook.

showDaeGeneration **(default: off)**
Show the dae variable declarations as they happen.

showEquationSource **(default: off)**
Display the element source information in the dumped DAE for easier debugging.

showExpandableInfo **(default: off)**
Show information about expandable connector handling.

showInstCacheInfo **(default: off)**
Prints information about instantiation cache hits and additions. Defaults to false.

showStartOrigin **(default: off)**
Enables dumping of the DAE startOrigin attribute of the variables.

showStatement **(default: off)**
Shows the statement that is currently being evaluated when evaluating a script.

skipInputOutputSyntacticSugar **(default: off)**
Used when bootstrapping to preserve the input output parsing of the code output by the list command.

stateselection **(default: off)**
Enables dumping of selected states. Extends -d=backenddaeinfo.

static **(default: off)**
Enables extra debug output from the static elaboration.

stripPrefix **(default: on)**
Strips the environment prefix from path/crefs. Defaults to true.

susanDebug **(default: off)**
Makes Susan generate code using try/else to better debug which function broke the expected match semantics.

symJacConstantSplit **(default: off)**
Generates all symbolic Jacobians with splitted constant parts.

symjacdump **(default: off)**
Dumps information about symbolic Jacobians.

symjacdumpverbose **(default: off)**
Dumps information in verbose mode about symbolic Jacobians.

tail **(default: off)**
Prints out a notification if tail recursion optimization has been applied.

tearingCost **(default: off)**
Dumps the estimated cost of every torn system against solving it untorn.

tearingdump **(default: off)**
Dumps tearing information.

tearingdumpV **(default: off)**
Dumps verbose tearing information.

totaltearingdump **(default: off)**
Dumps total tearing information.

totaltearingdumpV **(default: off)**
Dumps verbose total tearing information.

tplPerfTimes **(default: off)**
Enables output of template performance data for rendering text to file.

transformsbeforedump **(default: off)**
Applies transformations required for code generation before dumping flat code.

types **(default: off)**
Prints extra failtrace from Types.

uncertainties **(default: off)**
Enables dumping of status when calling modelEquationsUC.

updmod **(default: off)**
Prints information about modification updates.

useMPI **(default: off)**
Add MPI init and finalize to main method (CPPruntime).

vectorize **(default: off)**
Activates vectorization in the backend.

vectorizeBindings **(default: off)**
Turns on vectorization of bindings when scalarization is turned off.

visxml **(default: off)**
Outputs a xml-file that contains information for visualization.

warnMinMax **(default: on)**
Makes a warning assert from min/max variable attributes instead of error.

warnNoNominal **(default: off)**
Prints the iteration variables in the initialization and simulation DAE, which do not have a nominal value.

writeToBuffer **(default: off)**
Enables writing simulation results to buffer.

zmqDangerousAcceptConnectionsFromAnywhere **(default: off)**
When opening a zmq connection, listen on all interfaces instead of only connections from 127.0.0.1.

31.3 Flags for Optimization Modules

Flags that determine which symbolic methods are used to produce the causalized equation system.

The *--preOptModules* flag sets the optimization modules which are used before the matching and index reduction in the back end. These modules are specified as a comma-separated list.

The *--matchingAlgorithm* sets the method that is used for the matching algorithm, after the pre optimization modules.

The *--indexReductionMethod* sets the method that is used for the index reduction, after the pre optimization modules.

The *--initOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for initialization, specified as a comma-separated list.

The *--postOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for simulation, specified as a comma-separated list.

CHAPTER

THIRTYTWO

SIMULATION RUNTIME FLAGS

This chapter contains a *short overview of simulation flags* as well as additional details of the *numerical integration methods*.

32.1 C Runtime Simulation Flags

The simulation executable takes the following flags:

-abortSlowSimulation
Aborts if the simulation chatters.

-alarm=value ***or*** **-alarm value**
Aborts after the given number of seconds (default=0 disables the alarm).

-clock=value ***or*** **-clock value**
Selects the type of clock to use. Valid options include:

- RT (monotonic real-time clock)
- CYC (cpu cycles measured with RDTSC)
- CPU (process-based CPU-time)

-cpu
Dumps the cpu-time into the result file using the variable named $cpuTime.

-csvOstep=value ***or*** **-csvOstep value**
Value specifies csv-files for debug values for optimizer step.

-cvodeNonlinearSolverIteration=value ***or*** **-cvodeNonlinearSolverIteration value**
Nonlinear solver iteration for CVODE solver. Default: Depends on flag cvodeLinearMultistepMethod. Valid values

- **CV_ITER_NEWTON - Newton iteration.**
  Advised to use together with flag -cvodeLinearMultistepMethod=CV_BDF.
- **CV_ITER_FIXED_POINT - Fixed-Point iteration iteration.**
  Advised to use together with flag -cvodeLinearMultistepMethod=CV_ADAMS.

-cvodeLinearMultistepMethod=value ***or*** **-cvodeLinearMultistepMethod value**
Linear multistep method for CVODE solver. Default: CV_BDF. Valid values

- **CV_BDF - BDF linear multistep method for stiff problems.**
  Use together with flag -cvodeNonlinearSolverIteration=CV_ITER_NEWTON or don't set cvodeNonlinearSolverIteration.
- **CV_ADAMS - Adams-Moulton linear multistep method for nonstiff problems.**
  Use together with flag -cvodeNonlinearSolverIteration=CV_ITER_FIXED_POINT or don't set cvodeNonlinearSolverIteration.

-cx=value ***or*** **-cx value**
Value specifies an csv-file with inputs as correlation coefficient matrix Cx for DataReconciliation

-daeMode
Enables daeMode simulation if the model was compiled with the omc flag --daeMode and ida method is used.

-deltaXLinearize=value ***or*** **-deltaXLinearize value**
Value specifies the delta x value for numerical differentiation used by linearization. The default value is sqrt(DBL_EPSILON*2e1).

-deltaXSolver=value ***or*** **-deltaXSolver value**
Value specifies the delta x value for numerical differentiation used by integration method. The default values is sqrt(DBL_EPSILON).

-embeddedServer=value ***or*** **-embeddedServer value**
Enables an embedded server. Valid values:

- none - default, run without embedded server
- opc-da - [broken] run with embedded OPC DA server (WIN32 only, uses proprietary OPC SC interface)
- opc-ua - [experimental] run with embedded OPC UA server (TCP port 4841 for now; will have its own configuration option later)
- filename - path to a shared object implementing the embedded server interface (requires access to internal OMC data-structures if you want to read or write data)

-embeddedServerPort=value ***or*** **-embeddedServerPort value**
Value specifies the port number used by the embedded server. The default value is 4841.

-mat_sync=value ***or*** **-mat_sync value**
Syncs the mat file header after emitting every N time-points.

-emit_protected
Emits protected variables to the result-file.

-eps=value ***or*** **-eps value**
Value specifies the number of convergence iteration to be performed for DataReconciliation

-f=value ***or*** **-f value**
Value specifies a new setup XML file to the generated simulation code.

-help=value ***or*** **-help value**
Get detailed information that specifies the command-line flag

For example, -help=f prints detailed information for command-line flag f.

-homAdaptBend=value ***or*** **-homAdaptBend value**
Maximum trajectory bending to accept the homotopy step. Default: 0.5, which means the corrector vector has to be smaller than half of the predictor vector.

-homBacktraceStrategy=value ***or*** **-homBacktraceStrategy value**
Value specifies the backtrace strategy in the homotopy corrector step. Valid values:

- fix - default, go back to the path by fixing one coordinate
- orthogonal - go back to the path in an orthogonal direction to the tangent vector

-homHEps=value ***or*** **-homHEps value**
Tolerance respecting residuals for the homotopy H-function (default: 1e-5).

In the last step (lambda=1) newtonFTol is used as tolerance.

-homMaxLambdaSteps=value ***or*** **-homMaxLambdaSteps value**
Maximum lambda steps allowed to run the homotopy path (default: system size * 100).

-homMaxNewtonSteps=value ***or*** **-homMaxNewtonSteps value**
Maximum newton steps in the homotopy corrector step (default: 20).

-homMaxTries=value ***or*** **-homMaxTries value**
Maximum number of tries for one homotopy lambda step (default: 10).

-homNegStartDir
Start to run along the homotopy path in the negative direction.

If one direction fails, the other direction is always used as fallback option.

-homotopyOnFirstTry
If the model contains the homotopy operator, directly use the homotopy method to solve the initialization problem. This is already the default behaviour, this flag can be used to undo the effect of -noHomotopyOnFirstTry

-noHomotopyOnFirstTry
Disable the use of the homotopy method to solve the initialization problem. Without this flag, the solver tries to solve the initialization problem with homotopy if the model contains the homotopy-operator.

-homTauDecFac=value ***or*** **-homTauDecFac value**
Decrease homotopy step size tau by this factor if tau is too big in the homotopy corrector step (default: 10.0).

-homTauDecFacPredictor=value ***or*** **-homTauDecFacPredictor value**
Decrease homotopy step size tau by this factor if tau is too big in the homotopy predictor step (default: 2.0).

-homTauIncFac=value ***or*** **-homTauIncFac value**
Increase homotopy step size tau by this factor if tau can be increased after the homotopy corrector step (default: 2.0).

-homTauIncThreshold=value ***or*** **-homTauIncThreshold value**
Increase the homotopy step size tau if homAdaptBend/bend > homTauIncThreshold (default: 10).

-homTauMax=value ***or*** **-homTauMax value**
Maximum homotopy step size tau for the homotopy process (default: 10).

-homTauMin=value ***or*** **-homTauMin value**
Minimum homotopy step size tau for the homotopy process (default: 1e-4).

-homTauStart=value ***or*** **-homTauStart value**
Homotopy step size tau at the beginning of the homotopy process (default: 0.2).

-idaMaxErrorTestFails=value ***or*** **-idaMaxErrorTestFails value**
Value specifies the maximum number of error test failures in attempting one step. The default value is 7.

-idaMaxNonLinIters=value ***or*** **-idaMaxNonLinIters value**
Value specifies the maximum number of nonlinear solver iterations at one step. The default value is 3.

-idaMaxConvFails=value ***or*** **-idaMaxConvFails value**
Value specifies the maximum number of nonlinear solver convergence failures at one step. The default value is 10.

-idaNonLinConvCoef=value ***or*** **-idaNonLinConvCoef value**
Value specifies the safety factor in the nonlinear convergence test. The default value is 0.33.

-idaLS=value ***or*** **-idaLS value**
Value specifies the linear solver of the ida integration method. Valid values:

- dense (ida internal dense method.)
- klu (ida use sparse direct solver KLU. (default))
- spgmr (ida generalized minimal residual method. Iterative method)
- spbcg (ida Bi-CGStab. Iterative method)
- sptfqmr (ida TFQMR. Iterative method)

-idaScaling
Enable scaling of the IDA solver.

-idaSensitivity
Enables sensitivity analysis with respect to parameters if the model is compiled with omc flag --calculateSensitivities.

-ignoreHideResult
Emits also variables with HideResult=true annotation.

-iif=value ***or*** **-iif value**
Value specifies an external file for the initialization of the model.

-iim=value ***or*** **-iim value**
Value specifies the initialization method. Following options are available: 'symbolic' (default) and 'none'.

- none (sets all variables to their start values and skips the initialization process)
- symbolic (solves the initialization problem symbolically - default)

-iit=value ***or*** **-iit value**
Value [Real] specifies a time for the initialization of the model.

-ils=value ***or*** **-ils value**
Value specifies the number of steps for homotopy method (required: -iim=symbolic). The value is an Integer with default value 3.

-initialStepSize=value ***or*** **-initialStepSize value**
Value specifies an initial step size, used by the methods: dassl, ida, gbode

-csvInput=value ***or*** **-csvInput value**
Value specifies an csv-file with inputs for the simulation/optimization of the model

-stateFile=value ***or*** **-stateFile value**
Value specifies an file with states start values for the optimization of the model.

-inputPath=value ***or*** **-inputPath value**
Value specifies a path for reading the input files i.e., model_init.xml and model_info.json

-ipopt_hesse=value ***or*** **-ipopt_hesse value**
Value specifies the hessematrix for Ipopt(OMC, BFGS, const).

-ipopt_init=value ***or*** **-ipopt_init value**
Value specifies the initial guess for optimization (sim, const).

-ipopt_jac=value ***or*** **-ipopt_jac value**
Value specifies the Jacobian for Ipopt(SYM, NUM, NUMDENSE).

-ipopt_max_iter=value ***or*** **-ipopt_max_iter value**
Value specifies the max number of iteration for ipopt.

-ipopt_warm_start=value ***or*** **-ipopt_warm_start value**
Value specifies lvl for a warm start in ipopt: 1,2,3,...

-jacobian=value ***or*** **-jacobian value**
Select the calculation method for Jacobian used by the integration method:

- coloredNumerical (Colored numerical Jacobian, which is default for dassl and ida. Needs omc compiler flag --generateDynamicJacobian=numeric. With option -idaLS=klu a sparse matrix is used.)
- internalNumerical (Dense solver internal numerical Jacobian.)
- coloredSymbolical (Colored symbolical Jacobian. Needs omc compiler flag --generateDynamicJacobian=symbolic. With option -idaLS=klu a sparse matrix is used.)
- coloredSymbolicalAdjoint (Colored symbolical adjoint Jacobian. Needs omc compiler flags --newBackend and --generateDynamicJacobian=symbolicadjoint.)
- numerical (Dense numerical Jacobian.)
- symbolical (Dense symbolical Jacobian. Needs omc compiler flag --generateDynamicJacobian=symbolic.)
- bicoloredSymbolical (Bidirectional (star bicolored) symbolical Jacobian. Needs omc compiler flags --newBackend and --generateDynamicJacobian=bidirectional.)

-jacobianNominalFactor=value ***or*** **-jacobianNominalFactor value**
The numerical Jacobian differences column i over

```
delta_h * max(|x[i]|, |h*x'[i]|)
```

and, where that is inside the variable's own absolute tolerance and so is no scale of its own, over delta_h*factor*nominal[i] instead. Lower the factor for a model that is non-smooth at that wider step; the value is a Double with default value 1.0.

-l=value ***or*** **-l value**
Value specifies a time where the linearization of the model should be performed.

-l_datarec
Emit data recovery matrices with model linearization.

-logFormat=value ***or*** **-logFormat value**
Value specifies the log format of the executable:

- text (default)
- xml
- xmltcp (required -port flag)

-ls=value ***or*** **-ls value**
Value specifies the linear solver method

- lapack (method using LAPACK LU factorization)
- lis (method using iterative solver Lis)
- klu (method using KLU sparse linear solver)
- umfpack (method using UMFPACK sparse linear solver)
- totalpivot (method using a total pivoting LU factorization for underdetermination systems)
- default (default method - LAPACK with total pivoting as fallback)

-ls_ipopt=value ***or*** **-ls_ipopt value**
Value specifies the linear solver method for Ipopt, default mumps. Note: Use if you build ipopt with other linear solver like ma27

-lss=value ***or*** **-lss value**
Value specifies the linear sparse solver method

- default (the default sparse linear solver (or a dense solver if there is none available))
- lis (method using iterative solver Lis)
- klu (method using klu sparse linear solver)
- umfpack (method using umfpack sparse linear solver)

-lssMaxDensity=value ***or*** **-lssMaxDensity value**
Value specifies the maximum density for using a linear sparse solver. The value is a Double with default value 0.2.

-lssMinSize=value ***or*** **-lssMinSize value**
Value specifies the minimum system size for using a linear sparse solver. The value is an Integer with default value 1000.

-lv=value ***or*** **-lv value**
Value (a comma-separated String list) specifies which logging levels to enable. Multiple options can be enabled at the same time.

- LOG_STDOUT (this stream is always active, can be disabled with -lv=-LOG_STDOUT)
- LOG_ASSERT (this stream is always active, can be disabled with -lv=-LOG_ASSERT)
- LOG_DASSL (additional information about dassl solver)

- LOG_DASSL_STATES (outputs the states at every dassl call)
- LOG_DEBUG (additional debug information)
- LOG_DELAY (debug information for delay operator)
- LOG_DIVISION (Log division by zero)
- LOG_DSS (outputs information about dynamic state selection)
- LOG_DSS_JAC (outputs jacobian of the dynamic state selection)
- LOG_DT (additional information about dynamic tearing)
- LOG_DT_CONS (additional information about dynamic tearing (local and global constraints))
- LOG_EVENTS (additional information during event iteration)
- LOG_EVENTS_V (verbose logging of event system)
- LOG_GBODE (information about GBODE solver)
- LOG_GBODE_V (verbose information about GBODE solver)
- LOG_GBODE_NLS (log non-linear solver process of GBODE solver)
- LOG_GBODE_NLS_V (verbose log non-linear solver process of GBODE solver)
- LOG_GBODE_STATES (output states at every GBODE call)
- LOG_INIT (additional information during initialization)
- LOG_INIT_HOMOTOPY (log homotopy initialization)
- LOG_INIT_V (verbose information during initialization)
- LOG_IPOPT (information from Ipopt)
- LOG_IPOPT_FULL (more information from Ipopt)
- LOG_IPOPT_JAC (check jacobian matrix with Ipopt)
- LOG_IPOPT_HESSE (check hessian matrix with Ipopt)
- LOG_IPOPT_ERROR (print max error in the optimization)
- LOG_JAC (outputs the jacobian matrix used by ODE solvers)
- LOG_LS (logging for linear systems)
- LOG_LS_V (verbose logging of linear systems)
- LOG_MIXED (logging for mixed systems)
- LOG_MOO (logging for dynamic optimization library MOO)
- LOG_NLS (logging for nonlinear systems)
- LOG_NLS_V (verbose logging of nonlinear systems)
- LOG_NLS_HOMOTOPY (logging of homotopy solver for nonlinear systems)
- LOG_NLS_JAC (outputs the jacobian of nonlinear systems)
- LOG_NLS_JAC_TEST (tests the analytical jacobian of nonlinear systems)
- LOG_NLS_JAC_SUMS (tests the absolute sums of rows and cols in Kinsol Jacobian)
- LOG_NLS_NEWTON_DIAGNOSTICS (newton diagnostics (see: https://doi.org/10.1016/j.amc.2021.125991))
- LOG_NLS_DERIVATIVE_TEST (test derivatives in KINSOL nonlinear systems)
- LOG_NLS_SVD (perform a SVD analysis in KINSOL nonlinear systems)
- LOG_NLS_SVD_V (perform a SVD analysis in KINSOL nonlinear systems (verbose))

- LOG_NLS_RES (outputs every evaluation of the residual function)
- LOG_NLS_EXTRAPOLATE (outputs debug information about extrapolate process)
- LOG_RES_INIT (outputs residuals of the initialization)
- LOG_RT (additional information regarding real-time processes)
- LOG_SIMULATION (additional information about simulation process)
- LOG_SOLVER (additional information about solver process)
- LOG_SOLVER_V (verbose information about the integration process)
- LOG_SOLVER_CONTEXT (context information during the solver process)
- LOG_SOTI (final solution of the initialization)
- LOG_SPATIALDISTR (logging of internal operations for spatialDistribution)
- LOG_STATS (additional statistics about timer/events/solver)
- LOG_STATS_V (additional statistics for OMC_LOG_STATS)
- LOG_SUCCESS (this stream is always active, unless deactivated with -lv=-LOG_SUCCESS)
- LOG_SYNCHRONOUS (log clocks and sub-clocks for synchronous features)
- LOG_ZEROCROSSINGS (additional information about the zerocrossings)

***-lvMaxWarn=value* *or* -lvMaxWarn value**
Maximum number of times some repeating warnings are displayed. Default value 3.

***-lv_time=value* *or* -lv_time value**
Interval (a comma-separated Double list with two elements) specifies in which time interval logging is active. Doesn't affect LOG_STDOUT, LOG_ASSERT, and LOG_SUCCESS, LOG_STATS, LOG_STATS_V.

***-lv_system=value* *or* -lv_system value**
Value is a comma-separated list of equation indices (available in the transformational debugger) for which solver logs are shown (by default logs for all systems are shown)

***-mbi=value* *or* -mbi value**
Value specifies the maximum number of bisection iterations for state event detection or zero for default behavior

***-mei=value* *or* -mei value**
Value specifies the maximum number of event iterations. The value is an Integer with default value 20.

***-maxIntegrationOrder=value* *or* -maxIntegrationOrder value**
Value specifies maximum integration order, used by the methods: dassl, ida.

***-maxStepSize=value* *or* -maxStepSize value**
Value specifies maximum absolute step size, used by the methods: dassl, ida, gbode.

***-measureTimePlotFormat=value* *or* -measureTimePlotFormat value**
Value specifies the output format of the measure time functionality:

- svg
- jpg
- ps
- gif
- ...

-moo
Perform dynamic optimization with MOO library

***-moo_l2bn_p1_it=value* *or* -moo_l2bn_p1_it value**
Value specifies the number of phase I iterations (full bisections) for L2-Boundary-Norm mesh refinement in MOO

-moo_l2bn_p2_it=value ***or*** **-moo_l2bn_p2_it value**
Value specifies the number of phase II iterations (refinement) for L2-Boundary-Norm mesh refinement in MOO

-moo_l2bn_p2_lvl=value ***or*** **-moo_l2bn_p2_lvl value**
Value specifies the phase II refinement aggressiveness for L2-Boundary-Norm mesh refinement in MOO

-newtonFTol=value ***or*** **-newtonFTol value**
Tolerance respecting residuals for updating solution vector in Newton solver. Solution is accepted if the (scaled) 2-norm of the residuals is smaller than the tolerance newtonFTol and the (scaled) newton correction (delta_x) is smaller than the tolerance newtonXTol. The value is a Double with default value 1e-12.

-newtonMaxSteps=value ***or*** **-newtonMaxSteps value**
Maximum number of newton steps, used currently only by KINSOL in solver GBODE.

-newtonMaxStepFactor=value ***or*** **-newtonMaxStepFactor value**
Maximum newton step factor mxnewtstep = maxStepFactor * norm2(xScaling). Used currently only by KINSOL.

-newtonXTol=value ***or*** **-newtonXTol value**
Tolerance respecting newton correction (delta_x) for updating solution vector in Newton solver. Solution is accepted if the (scaled) 2-norm of the residuals is smaller than the tolerance newtonFTol and the (scaled) newton correction (delta_x) is smaller than the tolerance newtonXTol. The value is a Double with default value 1e-12.

-newtonJacUpdates=value ***or*** **-newtonJacUpdates value**
Number of steps before Jacobian is recomputed. If zero is chosen, the corresponding solution phase is skipped.

-newton=value ***or*** **-newton value**
Value specifies the damping strategy for the newton solver.

- damped (Newton with a damping strategy)
- damped2 (Newton with a damping strategy 2)
- damped_ls (Newton with a damping line search)
- damped_bt (Newton with a damping backtracking and a minimum search via golden ratio method)
- pure (Newton without damping strategy)

-nls=value ***or*** **-nls value**
Value specifies the nonlinear solver:

- hybrid (Modification of the Powell hybrid method from minpack - former default solver)
- kinsol (SUNDIALS/KINSOL includes an interface to the sparse direct solver, KLU. See simulation option -nlsLS for more information.)
- experimental-kinsol (SUNDIALS/KINSOL (with internal scaling) includes an interface to the sparse direct solver, KLU. See simulation option -nlsLS for more information.)
- newton (Newton Raphson - prototype implementation)
- mixed (Mixed strategy. First the homotopy solver is tried and then as fallback the hybrid solver.)
- homotopy (Damped Newton solver if failing case fixed-point and Newton homotopies are tried.)

-nlsInfo
Outputs detailed information about solving process of non-linear systems into csv files.

-nlsLS=value ***or*** **-nlsLS value**
Value specifies the linear solver used by the non-linear solver:

- default (chooses the nls linear solver based on which nls is being used.)
- totalpivot (internal total pivot implementation. Solve in some case even under-determined systems.)
- lapack (use external LAPACK implementation.)

- klu (use KLU direct sparse solver. Only with KINSOL available.)

***-nlssMaxDensity=value* or -nlssMaxDensity value**
Value specifies the maximum density for using a non-linear sparse solver. The value is a Double with default value 0.1.

***-nlssMinSize=value* or -nlssMinSize value**
Value specifies the minimum system size for using a non-linear sparse solver. The value is an Integer with default value 1000.

***-nlsJacTestATol=value* or -nlsJacTestATol value**
Value specifies the absolute tolerance for the Jacobian derivative test.

***-nlsJacTestRTol=value* or -nlsJacTestRTol value**
Value specifies the relative tolerance for the Jacobian derivative test.

-noemit
Do not emit any results to the result file.

-noEquidistantTimeGrid
Output the internal steps given by dassl/ida instead of interpolating results into an equidistant time grid as given by stepSize or numberOfIntervals.

***-noEquidistantOutputFrequency=value* or -noEquidistantOutputFrequency value**
Integer value n controls the output frequency in noEquidistantTimeGrid mode and outputs every n-th time step

***-noEquidistantOutputTime=value* or -noEquidistantOutputTime value**
Real value timeValue controls the output time point in noEquidistantOutputTime mode and outputs every time>=k*timeValue, where k is an integer

-noEventEmit
Do not emit event points to the result file.

-noRestart
Disables the restart of the integration method after an event is performed, used by the methods: dassl, ida

-noRootFinding
Disables the internal root finding procedure of methods: dassl and ida.

-noScaling
Disables scaling for the variables and the residuals in the algebraic nonlinear solver KINSOL.

-noSuppressAlg
Flag to not suppress algebraic variables in the local error test of the ida solver in daeMode. In general, the use of this option is discouraged when solving DAE systems of index 1, whereas it is generally encouraged for systems of index 2 or more.

***-optDebugJac=value* or -optDebugJac value**
Value specifies the number of iterations from the dynamic optimization, which will be debugged, creating .csv and .py files.

***-optimizerNP=value* or -optimizerNP value**
Value specifies the number of points in a subinterval. Currently supports numbers 1 and 3.

***-optimizerTimeGrid=value* or -optimizerTimeGrid value**
Value specifies external file with time points.

***-output=value* or -output value**
Output the variables a, b and c at the end of the simulation to the standard output: time = value, a = value, b = value, c = value

***-outputFormat=value* or -outputFormat value**
Sets file format of the result file.

***-outputPath=value* or -outputPath value**
Value specifies a path for writing the output files i.e., model_res.mat, model_prof.intdata, model_prof.realdata etc.

***-override=value* *or* -override value**
Override the variables in the XML setup file For example: var1=start1,var2=start2,par3=start3

***-overrideFile=value* *or* -overrideFile value**
Will override the variables in the XML setup file with the values from the file. Note that: -overrideFile CANNOT be used with -override. Use when variables for -override are too many. overrideFileName contains lines of the form: var=start

***-port=value* *or* -port value**
Value specifies the port for simulation status (default disabled).

***-r=value* *or* -r value**
Value specifies the name of the output result file. The default file-name is based on the model name and output format. For example: Model_res.mat.

-reconcile
Run the Data Reconciliation numerical computation algorithm for constrained equations

-reconcileBoundaryConditions
Run the Data Reconciliation numerical computation algorithm for boundary condition equations

-reconcileState
Run the State Estimation numerical computation algorithm for constrained equations

***-gbm=value* *or* -gbm value**
Value specifies the chosen solver of solver gbode (single-rate, slow states integrator).

- adams (Implicit multistep method of type Adams-Moulton (order 2))
- expl_euler (Explizit Runge-Kutta Euler method (order 1))
- impl_euler (Implizit Runge-Kutta Euler method (order 1, L-stable))
- trapezoid (Implicit Runge-Kutta trapezoid method (order 2, A-stable))
- sdirk2 (Singly-diagonal implicit Runge-Kutta (order 2, L-stable))
- sdirk3 (Singly-diagonal implicit Runge-Kutta (order 3, L-stable))
- sdirk4 (Singly-diagonal implicit Runge-Kutta (order 4, L-stable))
- esdirk2 (Explicit singly-diagonal implicit Runge-Kutta (order 2, L-stable))
- esdirk3 (Explicit singly-diagonal implicit Runge-Kutta (order 3, L-stable))
- esdirk4 (Explicit singly-diagonal implicit Runge-Kutta (order 4, L-stable, 6 stages))
- esdirk4s7 (Explicit singly-diagonal implicit Runge-Kutta (order 4, L-stable, 7 stages))
- sirk3l (Singly-implicit Runge-Kutta SIRK3(2)3L[2]SA (order 3, L-stable))
- sirk3 (Singly-implicit Runge-Kutta SIRK3(2)4L[3]SA (order 3, L-stable))
- sirk3s5 (Singly-implicit Runge-Kutta SIRK3(2)5L[3]SA (order 3, L-stable))
- sirk4lag (Singly-implicit Runge-Kutta SIRK4(3) Laguerre (order 4, A(89.548*)-stable))
- sirk4 (Singly-implicit Runge-Kutta SIRK4(3)5L[3]SA (order 4, L-stable))
- sirk5 (Singly-implicit Runge-Kutta SIRK5(4)5L[3]SA (order 5, L-stable))
- sirk4s6 (Singly-implicit Runge-Kutta SIRK4(3)6L[4]SA (order 4, L-stable))
- sirk5s6 (Singly-implicit Runge-Kutta SIRK5(4)6L[4]SA (order 5, L-stable))
- sirk5s7 (Singly-implicit Runge-Kutta SIRK5(4)7L[4]SA (order 5, L-stable))
- sirk6s7 (Singly-implicit Runge-Kutta SIRK6(5)7L[4]SA (order 6, L-stable))
- sirk6s8 (Singly-implicit Runge-Kutta SIRK6(5)8L[4]SA (order 6, L-stable))
- sirk5s7l5 (Singly-implicit Runge-Kutta SIRK5(4)7L[5]SA (order 5, L-stable))
- firk7 (Fully-implicit Runge-Kutta FIRK7(6TS)5L[4]SA (order 7, L-stable))

- radauIA2 (Implicit Runge-Kutta method of Radau family IA (order 3, L-stable))
- radauIA3 (Implicit Runge-Kutta method of Radau family IA (order 5, L-stable))
- radauIA4 (Implicit Runge-Kutta method of Radau family IA (order 7, L-stable))
- radauIIA2 (Implicit Runge-Kutta method of Radau family IIA (order 3, L-stable))
- radauIIA3 (Implicit Runge-Kutta method of Radau family IIA (order 5, L-stable))
- radauIIA4 (Implicit Runge-Kutta method of Radau family IIA (order 7, L-stable))
- radauIIA5 (Implicit Runge-Kutta method of Radau family IIA (order 9, L-stable))
- radauIIA6 (Implicit Runge-Kutta method of Radau family IIA (order 11, L-stable))
- radauIIA7 (Implicit Runge-Kutta method of Radau family IIA (order 13, L-stable))
- lobattoIIIA3 (Implicit Runge-Kutta method of Lobatto family IIIA (order 4, A-stable))
- lobattoIIIA4 (Implicit Runge-Kutta method of Lobatto family IIIA (order 6, A-stable))
- lobattoIIIB3 (Implicit Runge-Kutta method of Lobatto family IIIB (order 4, A-stable))
- lobattoIIIB4 (Implicit Runge-Kutta method of Lobatto family IIIB (order 6, A-stable))
- lobattoIIIC3 (Implicit Runge-Kutta method of Lobatto family IIIC (order 4, L-stable))
- lobattoIIIC4 (Implicit Runge-Kutta method of Lobatto family IIIC (order 6, L-stable))
- gauss2 (Implicit Runge-Kutta method of Gauss (order 4, A-stable))
- gauss3 (Implicit Runge-Kutta method of Gauss (order 6, A-stable))
- gauss4 (Implicit Runge-Kutta method of Gauss (order 8, A-stable))
- gauss5 (Implicit Runge-Kutta method of Gauss (order 10, A-stable))
- gauss6 (Implicit Runge-Kutta method of Gauss (order 12, A-stable))
- merson (Explicit Runge-Kutta Merson method (order 4))
- mersonSsc1 (Explicit Runge-Kutta Merson method with large stability region (order 1))
- mersonSsc2 (Explicit Runge-Kutta Merson method with large stability region (order 2))
- heun (Explicit Runge-Kutta Heun method (order 2))
- fehlberg12 (Explicit Runge-Kutta Fehlberg method (order 2))
- fehlberg45 (Explicit Runge-Kutta Fehlberg method (order 5))
- fehlberg78 (Explicit Runge-Kutta Fehlberg method (order 8))
- fehlbergSsc1 (Explicit Runge-Kutta Fehlberg method with large stability region (order 1))
- fehlbergSsc2 (Explicit Runge-Kutta Fehlberg method with large stability region (order 2))
- rk810 (Explicit 8-10 Runge-Kutta method (order 10))
- rk1012 (Explicit 10-12 Runge-Kutta method (order 12))
- rk1214 (Explicit 12-14 Runge-Kutta method (order 14))
- dopri45 (Explicit Runge-Kutta method Dormand-Prince (order 5))
- dopriSsc1 (Explicit Runge-Kutta method Dormand-Prince with large stability region (order 1))
- dopriSsc2 (Explicit Runge-Kutta method Dormand-Prince with large stability region (order 2))
- tsit5 (Explicit Runge-Kutta method from Tsitouras (order 5))
- rungekutta (Explicit classical Runge-Kutta method (order 4))
- rungekuttaSsc (Explicit Runge-Kutta method with large stabiliy region (order 1))

-gbctrl=value ***or*** **-gbctrl value**
Step size control of solver gbode (single-rate, slow states integrator).

- i (I controller for step size (beta=1/k, k RK error order))
- pi_33 (PI controller for step size (beta1=0.7/k, beta2=-0.4/k))
- pi_34 (PI controller for step size (beta1=2./3./k, beta2=-1./3./k))
- pi_42 (PI controller for step size (beta1=0.6/k, beta2=-0.2/k))
- pid_h312 (PID controller for step size (alpha1=1./18./k, alpha2=1./9./k, alpha3=1./18./k))
- pid_soederlind (PID controller for step size (alpha1=0.1/k, alpha2=0.2/k, alpha3=0.1/k))
- pid_stiff (PID controller for step size (alpha1=0.58/k, alpha2=0.21/k, alpha3=0.21/k))
- pc (Standard PI Predictive controller (beta1=2/k, beta2=-1/k, ratio=1))
- pc_hybrid (Hybrid I / PI Predictive controller: min(PI Predictive, I))
- pi_h211 (PI Predictive controller (beta1=0.25/k, beta2=0.25/k, ratio=-0.25))
- pi_h0_211 (PI Predictive controller (beta1=0.5/k, beta2=0.5/k, ratio=-0.5))
- pid_h0_312 (PID Predictive controller (alpha1=0.25/k, alpha2=0.5/k, alpha3=0.25/k, ratio1=-0.75, ratio2=-0.25))
- pid_h0_321 (PID Predictive controller (alpha1=1.25/k, alpha2=0.5/k, alpha3=-0.75/k, ratio1=0.25, ratio2=0.75))
- ppid (PID Predictive controller (alpha1=6./20/k, alpha2=1./20/k, alpha3=-5./20/k, ratio1=1.0, ratio2=0.0))
- const (Constant step size)

-gbctrl_evnt_reinit
Reset step size using standard inital step size selection after an event (default false)

-gbctrl_filter=value ***or*** **-gbctrl_filter value**
Applies exponential smoothing to the step size factor; gbctrl_filter = 0 yields constant step size, gbctrl_filter = 1 uses full adaptation without averaging.

-gbctrl_fhr
Applies adaptive damping to the step size factor using Führer's approach, scaling it by `h_fac *= (h_n / h_n1)^gamma` to penalize repeated rejections or reward successful step acceptance.

-gberr=value ***or*** **-gberr value**
Error estimation method for solver gbode (single-rate, slow states integrator) Possible values:

- default - depending on the Runge-Kutta method
- richardson - Richardson extrapolation
- embedded - Embedded scheme
- two_step - Two-step estimator, if available
- contractive_defect - Contractive defect estimator, if available
- contractive_filter - Contractive filter applied to the embedded estimator, if available

-gbint=value ***or*** **-gbint value**
Interpolation method of solver gbode (single-rate, slow states integrator).

- linear (Linear interpolation (1st order))
- hermite (Hermite interpolation (3rd order))
- hermite_a (Hermite interpolation (only for left hand side))
- hermite_b (Hermite interpolation (only for right hand side))
- hermite_errctrl (Hermite interpolation with error control)

- dense_output (use dense output formula for interpolation)
- dense_output_errctrl (use dense output fomular with error control)

***-gbnls=value* or -gbnls value**
Non-linear solver method of solver gbode (single-rate, slow states integrator).

- newton (Newton method, dense)
- kinsol (SUNDIALS KINSOL: Inexact Newton, sparse)
- experimental-kinsol (experimental kinsol)
- internal (simplified Newton with decoupling transformation (uses KLU))

***-gbnls_internal_damping=value* or -gbnls_internal_damping value**
Value specifies damping applied to the estimated convergence rate in the first Newton iteration (0 <= value <= 1; 0 = conservative, 1 = optimistic). Only valid for -gbnls=internal. Since no history is available in the first Newton iteration, the convergence rate is taken from the previous solve and raised to the power of this value.

***-gbnls_internal_jackeep=value* or -gbnls_internal_jackeep value**
Value specifies how often the ODE Jacobian is recalculated (0 <= value < 1). Only valid for -gbnls=internal. The Jacobian is kept, if the linear convergence rate || dz_k || / || dz_{k-1} || of the Newton iteration is smaller than the specified value. Small values result in more Jacobian callbacks.

***-gbfm=value* or -gbfm value**
Value specifies the chosen solver of solver gbode (multi-rate, fast states integrator). Current Restriction: Fully implicit (Gauss, Radau, Lobatto) RK methods are not supported, yet.

- adams (Implicit multistep method of type Adams-Moulton (order 2))
- expl_euler (Explizit Runge-Kutta Euler method (order 1))
- impl_euler (Implizit Runge-Kutta Euler method (order 1, L-stable))
- trapezoid (Implicit Runge-Kutta trapezoid method (order 2, A-stable))
- sdirk2 (Singly-diagonal implicit Runge-Kutta (order 2, L-stable))
- sdirk3 (Singly-diagonal implicit Runge-Kutta (order 3, L-stable))
- sdirk4 (Singly-diagonal implicit Runge-Kutta (order 4, L-stable))
- esdirk2 (Explicit singly-diagonal implicit Runge-Kutta (order 2, L-stable))
- esdirk3 (Explicit singly-diagonal implicit Runge-Kutta (order 3, L-stable))
- esdirk4 (Explicit singly-diagonal implicit Runge-Kutta (order 4, L-stable, 6 stages))
- esdirk4s7 (Explicit singly-diagonal implicit Runge-Kutta (order 4, L-stable, 7 stages))
- sirk3l (Singly-implicit Runge-Kutta SIRK3(2)3L[2]SA (order 3, L-stable))
- sirk3 (Singly-implicit Runge-Kutta SIRK3(2)4L[3]SA (order 3, L-stable))
- sirk3s5 (Singly-implicit Runge-Kutta SIRK3(2)5L[3]SA (order 3, L-stable))
- sirk4lag (Singly-implicit Runge-Kutta SIRK4(3) Laguerre (order 4, A(89.548*)-stable))
- sirk4 (Singly-implicit Runge-Kutta SIRK4(3)5L[3]SA (order 4, L-stable))
- sirk5 (Singly-implicit Runge-Kutta SIRK5(4)5L[3]SA (order 5, L-stable))
- sirk4s6 (Singly-implicit Runge-Kutta SIRK4(3)6L[4]SA (order 4, L-stable))
- sirk5s6 (Singly-implicit Runge-Kutta SIRK5(4)6L[4]SA (order 5, L-stable))
- sirk5s7 (Singly-implicit Runge-Kutta SIRK5(4)7L[4]SA (order 5, L-stable))
- sirk6s7 (Singly-implicit Runge-Kutta SIRK6(5)7L[4]SA (order 6, L-stable))
- sirk6s8 (Singly-implicit Runge-Kutta SIRK6(5)8L[4]SA (order 6, L-stable))

- sirk5s7l5 (Singly-implicit Runge-Kutta SIRK5(4)7L[5]SA (order 5, L-stable))
- firk7 (Fully-implicit Runge-Kutta FIRK7(6TS)5L[4]SA (order 7, L-stable))
- radauIA2 (Implicit Runge-Kutta method of Radau family IA (order 3, L-stable))
- radauIA3 (Implicit Runge-Kutta method of Radau family IA (order 5, L-stable))
- radauIA4 (Implicit Runge-Kutta method of Radau family IA (order 7, L-stable))
- radauIIA2 (Implicit Runge-Kutta method of Radau family IIA (order 3, L-stable))
- radauIIA3 (Implicit Runge-Kutta method of Radau family IIA (order 5, L-stable))
- radauIIA4 (Implicit Runge-Kutta method of Radau family IIA (order 7, L-stable))
- radauIIA5 (Implicit Runge-Kutta method of Radau family IIA (order 9, L-stable))
- radauIIA6 (Implicit Runge-Kutta method of Radau family IIA (order 11, L-stable))
- radauIIA7 (Implicit Runge-Kutta method of Radau family IIA (order 13, L-stable))
- lobattoIIIA3 (Implicit Runge-Kutta method of Lobatto family IIIA (order 4, A-stable))
- lobattoIIIA4 (Implicit Runge-Kutta method of Lobatto family IIIA (order 6, A-stable))
- lobattoIIIB3 (Implicit Runge-Kutta method of Lobatto family IIIB (order 4, A-stable))
- lobattoIIIB4 (Implicit Runge-Kutta method of Lobatto family IIIB (order 6, A-stable))
- lobattoIIIC3 (Implicit Runge-Kutta method of Lobatto family IIIC (order 4, L-stable))
- lobattoIIIC4 (Implicit Runge-Kutta method of Lobatto family IIIC (order 6, L-stable))
- gauss2 (Implicit Runge-Kutta method of Gauss (order 4, A-stable))
- gauss3 (Implicit Runge-Kutta method of Gauss (order 6, A-stable))
- gauss4 (Implicit Runge-Kutta method of Gauss (order 8, A-stable))
- gauss5 (Implicit Runge-Kutta method of Gauss (order 10, A-stable))
- gauss6 (Implicit Runge-Kutta method of Gauss (order 12, A-stable))
- merson (Explicit Runge-Kutta Merson method (order 4))
- mersonSsc1 (Explicit Runge-Kutta Merson method with large stability region (order 1))
- mersonSsc2 (Explicit Runge-Kutta Merson method with large stability region (order 2))
- heun (Explicit Runge-Kutta Heun method (order 2))
- fehlberg12 (Explicit Runge-Kutta Fehlberg method (order 2))
- fehlberg45 (Explicit Runge-Kutta Fehlberg method (order 5))
- fehlberg78 (Explicit Runge-Kutta Fehlberg method (order 8))
- fehlbergSsc1 (Explicit Runge-Kutta Fehlberg method with large stability region (order 1))
- fehlbergSsc2 (Explicit Runge-Kutta Fehlberg method with large stability region (order 2))
- rk810 (Explicit 8-10 Runge-Kutta method (order 10))
- rk1012 (Explicit 10-12 Runge-Kutta method (order 12))
- rk1214 (Explicit 12-14 Runge-Kutta method (order 14))
- dopri45 (Explicit Runge-Kutta method Dormand-Prince (order 5))
- dopriSsc1 (Explicit Runge-Kutta method Dormand-Prince with large stability region (order 1))
- dopriSsc2 (Explicit Runge-Kutta method Dormand-Prince with large stability region (order 2))
- tsit5 (Explicit Runge-Kutta method from Tsitouras (order 5))
- rungekutta (Explicit classical Runge-Kutta method (order 4))

- rungekuttaSsc (Explicit Runge-Kutta method with large stabiliy region (order 1))

***-gbfctrl=value* or -gbfctrl value**
Step size control of solver gbode (multi-rate, fast states integrator).

- i (I controller for step size (beta=1/k, k RK error order))
- pi_33 (PI controller for step size (beta1=0.7/k, beta2=-0.4/k))
- pi_34 (PI controller for step size (beta1=2./3./k, beta2=-1./3./k))
- pi_42 (PI controller for step size (beta1=0.6/k, beta2=-0.2/k))
- pid_h312 (PID controller for step size (alpha1=1./18./k, alpha2=1./9./k, alpha3=1./18./k))
- pid_soederlind (PID controller for step size (alpha1=0.1/k, alpha2=0.2/k, alpha3=0.1/k))
- pid_stiff (PID controller for step size (alpha1=0.58/k, alpha2=0.21/k, alpha3=0.21/k))
- pc (Standard PI Predictive controller (beta1=2/k, beta2=-1/k, ratio=1))
- pc_hybrid (Hybrid I / PI Predictive controller: min(PI Predictive, I))
- pi_h211 (PI Predictive controller (beta1=0.25/k, beta2=0.25/k, ratio=-0.25))
- pi_h0_211 (PI Predictive controller (beta1=0.5/k, beta2=0.5/k, ratio=-0.5))
- pid_h0_312 (PID Predictive controller (alpha1=0.25/k, alpha2=0.5/k, alpha3=0.25/k, ratio1=-0.75, ratio2=-0.25))
- pid_h0_321 (PID Predictive controller (alpha1=1.25/k, alpha2=0.5/k, alpha3=-0.75/k, ratio1=0.25, ratio2=0.75))
- ppid (PID Predictive controller (alpha1=6./20/k, alpha2=1./20/k, alpha3=-5./20/k, ratio1=1.0, ratio2=0.0))
- const (Constant step size)

***-gbferr=value* or -gbferr value**
Error estimation method for solver gbode (multi-rate, fast states integrator) Possible values:

- default - depending on the Runge-Kutta method
- richardson - Richardson extrapolation
- embedded - Embedded scheme
- two_step - Two-step estimator, if available
- contractive_defect - Contractive defect estimator, if available
- contractive_filter - Contractive filter applied to the embedded estimator, if available

***-gbfint=value* or -gbfint value**
Interpolation method of solver gbode (multi-rate, fast states integrator).

- linear (Linear interpolation (1st order))
- hermite (Hermite interpolation (3rd order))
- hermite_a (Hermite interpolation (only for left hand side))
- hermite_b (Hermite interpolation (only for right hand side))
- hermite_errctrl (Hermite interpolation with error control)
- dense_output (use dense output formula for interpolation)
- dense_output_errctrl (use dense output fomular with error control)

***-gbfnls=value* or -gbfnls value**
Non-linear solver method of solver gbode (multi-rate, fast states integrator).

- newton (Newton method, dense)

- kinsol (SUNDIALS KINSOL: Inexact Newton, sparse)
- experimental-kinsol (experimental kinsol)
- internal (simplified Newton with decoupling transformation (uses KLU))

***-gbratio=value* *or* -gbratio value**
Define percentage of states for the fast states selection of solver gbode (values from 0 to 1).

***-rt=value* *or* -rt value**
Value specifies the scaling factor for real-time synchronization (0 disables). A value > 1 means the simulation takes a longer time to simulate.

***-s=value* *or* -s value**
Value specifies the integration method. For additional information see the *User's Guide*

- dassl (default) - BDF method - implicit (dense solver), variable step size control, adaptive order 1-5, event location
- ida - SUNDIALS IDA solver - BDF method - implicit (sparse/dense solver, default sparse) variable step size control, adaptive order 1-5, event location - additional simulation flags: -idaMaxErrorTestFails -idaMaxNonLinIters -idaMaxConvFails -idaNonLinConvCoef -idaLS -idaScaling -idaSensitivity
- cvode - SUNDIALS CVODE solver - BDF or Adams-Moulton solver - implicit (dense solver), variable step-size control, adaptive order 1-12, event location - additional simulation flags -cvodeLinearMultistepMethod -cvodeNonlinearSolverIteration
- gbode - generic Runge-Kutta ODE solver - implicit (sparse solver)/explicit, fixed/variable step size control, order 1-14, event location, optional bi-rate integration - additional simulation flags -gbm -gbctrl -gbratio - additional advanced flags -gbctrl_filter -gbctrl_fhr -gberr -gbint -gbnls -gbfm -gbfctrl -gbferr -gbfint -gbfnls
- euler - explicit Euler, fixed step size, order 1
- rungekutta - classical Runge-Kutta - explicit, fixed step, order 4
- symSolver - symbolic inline Solver [compiler flag '--symSolver' needed] - fixed step size, order 1
- symSolverSsc - symbolic implicit Euler with step size control [compiler flag '--symSolver' needed] - step size control, order 1
- qss - A QSS solver [experimental]
- optimization - Special solver for dynamic optimization

***-saveInitialGuess_system=value* *or* -saveInitialGuess_system value**
Debug flag that performs standard initialization until the specified system is reached, computes only the torn part and saves the results obtained so far to a .mat file

-single
Output results in single precision (mat-format only).

-steps
Dumps the number of integration steps into the result file.

***-startTime=value* *or* -startTime value**
Sets startTime for the simulation.

-steadyState
Aborts the simulation if steady state is reached.

***-steadyStateTol=value* *or* -steadyStateTol value**
This relative tolerance is used to detect steady state: `max(|d(x_i)/dt|/nominal(x_i)) < steadyStateTol`

***-stepSize=value* *or* -stepSize value**
Sets stepSize for the simulation.

***-stopAtSystem=value* *or* -stopAtSystem value**
Performs standard initialization until the specified system is reached, then aborts the simulation.

***-stopTime=value* or -stopTime value**
Sets stopTime for the simulation.

***-svdCount=value* or -svdCount value**
Number of extremal singular values and vectors computed for LOG_NLS_SVD (0 disables).

***-svdSigma=value* or -svdSigma value**
Estimated smallest singular value for the preconditioner in SVD analysis.

***-svdTol=value* or -svdTol value**
Convergence tolerance of the sparse SVD analysis: PRIMME stops at ||r|| <= svdTol*||A||. The normal equations it uses resolve no singular value below sqrt(DBL_EPSILON)*||A||, so tightening it past that buys nothing.

***-sx=value* or -sx value**
Value specifies an csv-file with inputs as covariance matrix Sx for DataReconciliation

***-tolerance=value* or -tolerance value**
Specifies solver tolerance.

***-keepHessian=value* or -keepHessian value**
Value specifies the number of steps, which keep Hessian matrix constant.

***-variableFilter=value* or -variableFilter value**
Specifies which variables should be present in the result-file using POSIX Extended Regular Expressions. The given expression must match the full variable name.

-w
Shows all warnings even if a related log-stream is inactive.

***-parmodNumThreads=value* or -parmodNumThreads value**
Value specifies the number of threads for simulation using parmodauto. If not specified (or is 0) it will use the systems max number of threads. Note that this option is ignored if the model is not compiled with --parmodauto

***-parmodScheduler=value* or -parmodScheduler value**
Selects the parmodauto scheduler: 'flow' (default) or 'level'. Switchable at run time without recompiling the simulator.

***-parmodClustering=value* or -parmodClustering value**
Selects the parmodauto clustering strategy: 'default' (merge-based), 'fixed_width_min_height', or 'none'.

***-parmodClustersPerLevel=value* or -parmodClustersPerLevel value**
Sets the maximum number of clusters per level used by the default clustering.

***-parmodExportTaskGraph=value* or -parmodExportTaskGraph value**
Exports the parmodauto task graph (tasks, dependencies) and the resulting clustering to the given json file, keyed by equation index.

***-parmodImportClustering=value* or -parmodImportClustering value**
Imports a clustering (groups of equation indices) from the given json file instead of computing one. The simulation aborts if the clustering is invalid (forms a cycle or references unknown equations).

***-parmodDumpStages=value* or -parmodDumpStages value**
Exports the parmodauto task graph and clustering before and after each clustering optimization to a series of json files named <prefix>.NN.<stage>.json, so the effect of each optimization can be inspected.

CHAPTER

THIRTYTHREE

TECHNICAL DETAILS

This chapter gives an overview of some implementation details that might be interesting when building tools around OpenModelica.

33.1 The MATv4 Result File Format

The default result-file format of OpenModelica is based on MATLAB level 4 MAT-files as described in the MATLAB documentation. This format can be read by tools such as MATLAB, Octave, Scilab, and SciPy. OpenModelica will write the result-files in a particular way that can be read by tools such as DyMat and Dymola (OpenModelica can also read files generated by Dymola since the used format is the same).

The variables stored in the MAT-file are (in the order required by OpenModelica):

Aclass

- `Aclass(1,:)` is always `Atrajectory`
- `Aclass(2,:)` is `1.1` in OpenModelica
- `Aclass(3,:)` is empty
- `Aclass(4,:)` is either `binTrans` or `binNormal`

The most important part of the variable is `Aclass(4,:)` since there are two main ways the result-file is stored: transposed or not. For efficiency, the result-file is written time-step by time-step during simulation. But the best way to read the data for a single variable is if the variables are stored variable by variable. If `Aclass(4,:)` is `binTrans`, all matrices need to be transposed since the file was not transposed for efficient reading of the file. Note that this affects all matrices, even matrices that do not change during simulation (such as name and description).

name
Is an n x m character (int8) matrix, where n is the number of variables stored in the result-file (including time). m is the length of the longest variable. OpenModelica stores the trailing part of the name as NIL bytes (0) whereas other tools use spaces for the trailing part.

description
Is an n x m character (int8) matrix containing the comment-string corresponding to the variable in the name matrix.

dataInfo
Is an n x 4 integer matrix containing information for each variable (in the same order as the name and description matrices).

- `dataInfo(i,1)` is `1` or `2`, saying if variable i is stored in the data_1 or data_2 matrix. If it is `0`, it is the abscissa (time variable).
- `dataInfo(i,2)` contains the index in the data_1 or data_2 matrix. The index is 1-based and may contain several variables pointing to the same row (alias variables). A negative value means that the variable is a negated alias variable.
- `dataInfo(i,3)` is 0 to signify linear interpolation. In other tools the value is the number of times differentiable this variable is, which may improve plotting.

- `dataInfo(i,4)` is `-1` in OpenModelica to signify that the value is not defined outside the time range. `0` keeps the first/last value when going outside the time range and `1` performs linear interpolation on the first/last two points.

data_1
: If it is an n x 1 matrix it contains the values of parameters. If it is an n x 2 matrix, the first and second column signify start and stop-values.

data_2
: Each row contains the values of a variable at the sampled times. The corresponding time stamps are stored in `data_2(1,:)`. `data_2(2,1)` is the value of some variable at time `data_2(1,1)`. The simulation flag `-single` stores `data_1` and `data_2` in single precision (matrix type 10, 4-byte elements) instead of double.

33.2 The Arrow Result File Format

`outputFormat="arrow"` (the simulation flag `-outputFormat=arrow`) writes the result as Apache Arrow IPC.

Compared to the MATv4 file it stores each variable as a typed column, so one variable is read without striding through the others; it carries the unit, description and type of every variable inside the file; it stores a variable that changes only at events only where it changes; and it can hold `String` variables, which the MATv4 file cannot.

The format is specified in `OMCompiler/SimulationRuntime/rust/openmodelica_arrow_writer/SPECIFICATION.md`. Nothing in it is specific to OpenModelica: it is meant to carry a Modelica result or an FMI one, and to be readable by any Arrow implementation without a Modelica tool in the loop.

Under `-mat_sync=N` a batch holds at most `N` rows and reaches the file as soon as it is complete, so the file can be read while the simulation runs.

CHAPTER

THIRTYFOUR

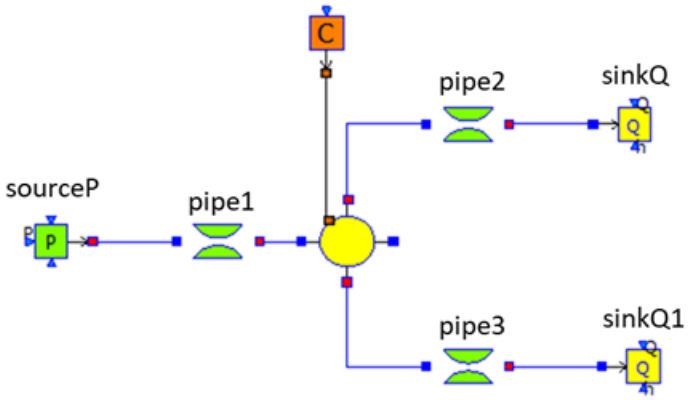
DATA RECONCILIATION

34.1 Objective of Data Reconciliation

The objective of data reconciliation is to use physical models to reduce the impact of measurement errors by decreasing measurement uncertainties and detecting faulty sensors. Data reconciliation is possible only when redundant measurements are available. Redundancy can be achieved by linking together the measured variables of interest using the physical laws that constrain them. This can be done with static Modelica models (models featuring algebraic equations only, no differential equations).

34.2 Defining the Data Reconciliation Problem in OpenModelica

Let us take the example of the Modelica model of a splitter.



Water flows from left to right, from the source to the sinks. The model is made of five different model components.

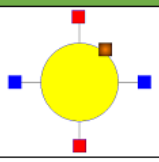
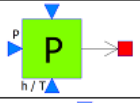
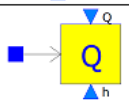
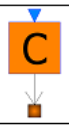
To perform data reconciliation, two kinds of variables must be declared:

1. The boundary conditions (which represent assumptions on the environment);
2. The variables of interest to be reconciled.

These two kinds of variables must be manually declared in the source code

1. With annotations for the boundary conditions;
2. With modifiers for the variables of interest.

The boundary conditions are declared with the annotation: annotation(__OpenModelica_BoundaryCondition = true). For the pressure source, the boundary conditions are the pressure P0, and the specific enthalpy h0 or the temperature T0 of the source (depending on the option chosen by the user).

Component name	Component icon	Component type	Component internal state variables
Volume		Component	P: pressure [Pa] T: temperature [K]
Pressure loss		Component	Q: mass flow rate [kg/s] Pm: mean pressure [Pa] T: temperature [K]
Pressure source		Boundary condition	P0: pressure [Pa] T0: temperature [K]
Mass flow rate sink		Boundary condition	Q0: mass flow rate [kg]
Heat source		Boundary condition	

```
model SourceP "Water/steam source with fixed pressure"
  parameter Modelica.SIunits.AbsolutePressure P0=300000 "Source pressure" annotation(_
↪_OpenModelica_BoundaryCondition = true);
  parameter Modelica.SIunits.Temperature T0=290 "Source temperature (active if option_
↪temperature=1" annotation(__OpenModelica_BoundaryCondition = true);
  parameter Modelica.SIunits.SpecificEnthalpy h0=100000
    "Source specific enthalpy (active if option_temperature=2)" annotation(__
↪OpenModelica_BoundaryCondition = true);
  parameter Integer option_temperature=1 "1:temperature fixed - 2:specific enthalpy␣
↪fixed";

  Modelica.SIunits.AbsolutePressure P "Fluid pressure";
  Modelica.SIunits.MassFlowRate Q "Mass flow rate";
  Modelica.SIunits.Temperature T "Fluid temperature";
  Modelica.SIunits.SpecificEnthalpy h "Fluid enthalpy";
equation
  P = P0;
  if (option_temperature == 1) then
    T = T0;
     h = f(T);
  else
    h = h0;
    T = g(h);
  end if;
end SourceP;
```

Boundary conditions are declared with annotations so that libraries can be modified to accomodate data reconciliation, and still can be used with tools that do not support data reconciliation (because annotations not recognized by a tool are ignored by that tool). The variables of interest are declared with the modifier "uncertain = Uncertainty.refine"

A modifier is used instead of an annotation so that checks can be performed to detect errors such as declaring a variable that does not exist to be a variable to be reconciled. The drawback is that tools that do not support data reconciliation will produce an error. To avoid this problem, variables of interest should be declared in a separate

model (Splitter_DR in the example below) that instantiates the orginal model (Splitter), so that the original model (Splitter) is not modified and can still be used with tools that do not support data reconciliation.

```
model Splitter_DR
  Splitter splitter(
     pipe1(Q(uncertain = Uncertainty.refine)),
     pipe2(Q(uncertain = Uncertainty.refine)),
     pipe3(Q(uncertain = Uncertainty.refine)),
     pipe1(Pm(uncertain = Uncertainty.refine)),
     pipe2(Pm(uncertain = Uncertainty.refine)),
     pipe3(Pm(uncertain = Uncertainty.refine)),
     pipe1(T(uncertain = Uncertainty.refine)),
     pipe2(T(uncertain = Uncertainty.refine)),
     pipe3(T(uncertain = Uncertainty.refine)));
end Splitter_DR;
```

In addition to declaring boundary conditions and variables of interest, one must provide two input files:

1. The measurement input file (mandatory).
2. The correlation matrix input file (optional).

The measurement input file is a csv file with three columns:

1. One column for the variable names [ident]
2. One column for measured values [positive floating point number]
3. One column for the half-width confidence intervals [positive floating point number].The half-with confidence interval for variable x_i is defined as $wi = \lambda_{95\%}\sigma_i$, where σ_i is the standard deviation of x_i and $\lambda_{95\%} = 1.96$

The header of the file is the row

Variable Names; Measured Value-x; Half Width Confidence Interval

It is possible to insert comments with // at the beginning of a row

// Measurement input file for the Splitter model.
Variable Name;Measured Value;Half Width Confidence Interval
splitter.pipe1.Q; 2.50; 0.196
splitter.pipe2.Q; 1.15; 0.196
splitter.pipe3.Q; 1.25; 0.196
splitter.pipe1.Pm; 6.1e5; 0.392e5
splitter.pipe2.Pm; 2.55e5; 0.392e5
splitter.pipe3.Pm; 2.45e5; 0.392e5
splitter.pipe1.T; 292; 1.96
splitter.pipe2.T; 386; 1.91
splitter.pipe3.T; 388; 1.91

The above file can be more easily visualized in matrix form:

The correlation matrix file is a csv file that contains the off-diagonal lower triangular correlation coefficients of the variables of interest:

1. The first row contains names of variables of interest [ident].
2. The first column contains names of variables of interest [ident].
3. The names in the first row and first column must be identical in the same order.
4. The first cell in the first row (which is also the first cell in the first column) must not be empty, but can contain any character string (except column separators).

Variable Name	Measured Value	Half Width Confidence Interval
splitter.pipe1.Q	2.50	0.196
splitter.pipe2.Q	1.15	0.196
splitter.pipe3.Q	1.25	0.196
splitter.pipe1.Pm	6.1e5	0.392e5
splitter.pipe2.Pm	2.55e5	0.392e5
splitter.pipe3.Pm	2.45e5	0.392e5
splitter.pipe1.T	292	1.96
splitter.pipe2.T	386	1.91
splitter.pipe3.T	388	1.91

5. The off-diagonal lower triangular matrix cells contain the correlation coefficients [positive or nul floating point number]. The correlation coefficients r_{ij} are defined such that $s_{ij} = r_{ij}\sigma_i\sigma_j$ where σ_i and σ_j are respectively the standard deviations of variables x_i and x_j, and s_{ij} is the covariance matrix. $r_{ii} = 1$ because $s_{ii} = \sigma_i^2 |r_{ij}| <= 1$
6. The upper triangular and diagonal cells are ignored because the correlation matrix is symmetric $r_{ji} = r_{ij}$, and its diagonal is $r_{ii} = 1$
7. Only variables of interest with positive correlation coefficients must appear in the matrix. Unfilled cells are equal to zero. Variables of interest that do not appear in the matrix have correlation coefficients equal to zero. Therefore, if all correlation coefficients are equal to zero, the matrix can be empty and the correlation matrix file is not needed

The following correlation file is drawn from the VDI2048 standard example of a heat circuit of a steam turbine plant.

```
Sxy;mV;mHK;mSPLL;mSPL;mFDKELL;mFDKEL
mV
mHK
mSPLL
mSPL;;;0.39951
mFDKELL;;;0;0
mFDKEL;;;0;0;0.2
```

The above file can be more easily visualized in matrix form:

Sxy	mV	mHK	mSPLL	mSPL	mFDKELL	mFDKEL
mV						
mHK						
mSPLL						
mSPL			0.39951			
mFDKELL			0		0	
mFDKEL			0		0	0.2

The variables mV and mHK could have been omitted because they do not have any positive correlation coefficients.

34.3 Data Reconciliation Support in OMEdit

The data reconciliation setup is done by:

1. Opening the Modelica model with the data reconciliation modifiers in OMEdit.
2. Selecting Data Reconciliation > Calculate Data Reconciliation.
3. Selecting the Data Reconciliation algorithm.
4. **Filling the Data Reconciliation form with**
    a. The name of the Measurement Input File (mandatory);
    b. The name of the Correlation Matrix Input file (optional). The default is the identity matrix (i.e. measurements are independent of each other);
    c. The value of Epsilon (optional). The default value is 1.e-10. Epsilon is the stopping criteria of the data reconciliation numerical iterations.
5. Clicking on Save Settings to save the above settings in the Modelica model.
6. Clicking on Calculate to launch the calculation

The data reconciliation computation is performed in three main steps:

1. **Static analysis is performed on the model to extract the equations that are necessary for data reconciliation. There are two groups of extracted equations:**
    a. The auxiliary conditions that constrain the variables of interest.
    b. The intermediate equations that solve the intermediate variables from the variables of interest (this is the numeric way of eliminating the intermediate variables).

    The auxiliary and intermediate equations are interchangeable: denoting r the number of auxiliary conditions, there are as many possibilities to construct the set of auxiliary conditions as to choose r equations among the set that contains both the auxiliary and the intermediate equations.

    Any error in the posing of the data reconciliation problem is detected at this step. The possible errors are:
    a. The number r of auxiliary equations is not strictly less than the number of variables of interest.
    b. The number r of auxiliary equations is zero.

    Both errors occur when there are too many boundary conditions related to the variables of interest. Variables of interest that are not involved in any of the auxiliary conditions or intermediate equations are not reconciled.
2. The model is simulated to compute the Jacobian matrices and eliminate numerically the intermediate variables. At this step, numerical simulation errors can occur, such as divisions by zero, non-convergence, etc. They can be corrected by improving the model or providing better start values.
3. **The input files are read, the data reconciliation calculation is performed, and the results are displayed. Errors can occur:**
    a. If input files do not exist.
    b. **If there is a mismatch between the variables of interest declared in the model and in the input files:**
        i. All variables of interest declared in the model should be declared in the measurement input file and reciprocally.
        ii. All variables of interest declared in the correlation matrix file should be declared in the model (the converse is not true).
    c. If a variable of interest has multiple entries in an input file.
    d. If the first row and the first column are different in the correlation matrix file.

e. If the numerical cells of the matrices are not positive real numbers.

The results are displayed:

1. In an html file with the title: Data Reconciliation Report. This file automatically pops up when the calculation is completed.
2. **In two csv output files:**
    a. One that contains the reconciled values and the reconciled half-width confidence intervals.
    b. The other that contains the reconciled covariance matrix.

These files do not pop up automatically when the calculation is completed. The names of the files are respectively

<working directory>\<model name>\< model name>_Outputs.csv

and

<working directory>\<model name>\< model name>_Reconciled_Sx.csv,

where <model name> denotes the full name of the model, including its path. The name of the working directory can be read with the command Tools > Options.

The Data Reconciliation Report has three sections:

1. **Overview, with the following information:**
    a. Model file: name of the Modelica model file
    b. Model name: name of the Modelica model
    c. Model directory: name of the directory of the Modelica model file
    d. Measurement input file: name of the measurement input file
    e. Correlation matrix input file: name of the correlation matrix file (if any)
    f. Generated: date and time of the generation of the data reconciliation report
2. **Analysis, with the following information:**
    a. Number of auxiliary conditions, denoted r in the sequel.
    b. Number of variables to be reconciled.
    c. Number of related boundary conditions: number of boundary conditions related to the variables to be reconciled. This number should be strictly less than the number of variables to be reconciled, otherwise the data reconciliation problem is ill-posed and data reconciliation cannot be performed.
    d. Number of iterations to convergence: number of iterations of the data reconciliation numerical loop.
    e. Final value of J*/r: final value of the data reconciliation iteration loop. This value is smaller than epsilon when the iterations are completed (cf. below).
    f. Epsilon: stopping criteria of the data reconciliation iteration loop. The recommended value by VDI 2048 is 1.e-10.
    g. Final value of the objective function J*: is equal to J*/r multiplied by r, where r is the number of auxiliary conditions.
    h. Chi-square value: value of the chi-square distribution for r degrees of freedom and statistical certainty of probability of 95%.
    i. Result of global test: true if J* is less than the chi-square value, false otherwise. If false, the results for the reconciled values should be rejected because the vector of contradictions (i.e. the discrepancy between the measured values and the reconciled values) is too large.
    j. Auxiliary conditions: set of the auxiliary conditions (i.e., the equations that constrain the variables of interest).

k. Intermediate equations: set of the intermediate equations (i.e., the equations that compute the intermediate variables from the variables of interest). This set can be empty if there are no intermediate variables.

l. Debug log: log of the numerical iteration loop.

3. **Results, which is a table with the following columns:**

   a. Variables to be Reconciled: the names of the variables of interest.

   b. Initial Measured Values: the measured values entered in the measurement input file.

   c. Reconciled Values: the reconciled values computed by the data reconciliation algorithm.

   d. Initial Half-width Confidence Intervals: the half-width confidence intervals entered in the measurement input file.

   e. Reconciled Half-width Confidence Intervals: the reconciled half-width confidence intervals computed by the data reconciliation algorithm.

   f. Results of Local Tests: true if the values of local tests (cf. below) are less than the quantile of normal distribution with probability 95% ($\lambda_{95\%}$), false otherwise.

   g. Values of Local Tests: values of the improvements (i.e., the difference between the initial and the reconciled values) divided by the square root of the diagonal element of the covariance matrix of the improvements.

   h. Margin to Correctness: $\lambda_{95\%}$ minus the values of local tests.

The data reconciliation report for the the VDI2048 standard example of a heat circuit of a steam turbine plant is given below.

34.3.1 Overview:

Model file: VDI2048Example.mo
Model name: NewDataReconciliationSimpleTests.VDI2048Example
Model directory: NewDataReconciliationSimpleTests
Measurement input file: VDI2048Example_Inputs.csv
Correlation matrix input file: VDI2048Example_Correlation.csv
Generated: Thu Jan 20 18:41:45 2022by OpenModelica v1.19.0-dev-500-g6c3a4e429f (64-bit)

34.3.2 Analysis:

Number of auxiliary conditions: 3
Number of variables to be reconciled: 11
Number of related boundary conditions: 0
Number of iterations to convergence: 2
Final value of (J*/r) : 4.56744e-28
Epsilon : 1e-10
Final value of the objective function (J*) : 1.37023e-27
Chi-square value : 7.81473
Result of global test : TRUE

34.3.3 Auxiliary conditions

1. (1): 0.0 = mHDANZ - mHDNK
2. (1): mFD3 = mHK + mA7 + mA6 + mA5 + 0.4 * mV
3. (1): mFD1 = mFDKEL + mFDKELL + (-0.2) * mV

34.3.4 Intermediate equations

1. (1): mHDANZ = mA7 + mA6 + mA5
2. (1): mFD2 = mSPL + mSPLL + (-0.6) * mV
3. (1): 0.0 = mFD2 - mFD3
4. (1): 0.0 = mFD1 - mFD2

34.3.5 Debug log

34.3.6 Results

Variables to be Reconciled	Initial Measured Values	Reconciled Values	Initial Half-width Confidence Intervals	Reconciled Half-width Confidence Intervals	Results of Local Tests	Values of Local Tests	Margin to Correctness
mFDKEL	46.241	44.6959	2.5	1.61062	TRUE	1.58381	0.376194
mFDKELL	45.668	44.1229	2.5	1.61062	TRUE	1.58381	0.376194
mSPL	44.575	44.6426	0.535	0.425429	TRUE	0.40853	1.55147
mSPLL	44.319	44.3861	0.532	0.423635	TRUE	0.40853	1.55147
mV	0.525	0.524499	0.105	0.104627	TRUE	0.0295873	1.93041
mHK	69.978	70.0049	0.854	0.615089	TRUE	0.08913	1.87087
mA7	10.364	10.3642	0.168	0.133112	TRUE	0.00387312	1.95613
mA6	3.744	3.74402	0.058	0.0566989	TRUE	0.00257972	1.95742
mA5	4.391	4.39102	0.058	0.0566989	TRUE	0.00257972	1.95742
mHDNK	18.498	18.4993	0.205	0.137264	TRUE	0.0161013	1.9439
mD	2.092	Not reconciled	0.272	Not reconciled	Not reconciled	Not reconciled	Not reconciled

mD is not reconciled because it does not appear in any of the auxiliary conditions or intermediate equations.

For the VDI2048 example, the name of the csv output file is

<working directory>\NewDataReconciliationSimpleTests.VDI2048Example\NewDataReconciliationSimpleTests.VDI2048Example_Output

and the name of the reconciled covariance matrix csv file is

<working directory>\NewDataReconciliationSimpleTests.VDI2048Example\NewDataReconciliationSimpleTests.VDI2048Example_Reconciled_Sx.csv

The Modelica model of the VDI2048 example is given below.

```
model VDI2048Example
  Real mFDKEL(uncertain=Uncertainty.refine)=46.241;
  Real mFDKELL(uncertain=Uncertainty.refine)=45.668;
  Real mSPL(uncertain=Uncertainty.refine)=44.575;
  Real mSPLL(uncertain=Uncertainty.refine)=44.319;
  Real mV(uncertain=Uncertainty.refine);
  Real mHK(uncertain=Uncertainty.refine)=69.978;
  Real mA7(uncertain=Uncertainty.refine)=10.364;
  Real mA6(uncertain=Uncertainty.refine)=3.744;
  Real mA5(uncertain=Uncertainty.refine);
  Real mHDNK(uncertain=Uncertainty.refine);
  Real mD(uncertain=Uncertainty.refine)=2.092;
```

(continues on next page)

(continued from previous page)

```
  Real mFD1;
  Real mFD2;
  Real mFD3;
  Real mHDANZ;
equation
  mFD1 = mFDKEL + mFDKELL - 0.2*mV;
  mFD2 = mSPL + mSPLL - 0.6*mV;
  mFD3 = mHK + mA7 + mA6 + mA5 + 0.4*mV;
  mHDANZ = mA7 + mA6 + mA5;

  0 = mFD1 - mFD2;
  0 = mFD2 - mFD3;
  0 = mHDANZ - mHDNK;
end VDI2048Example;
```

Note that the binding equations that assign fixed values to the variables have been automatically eliminated by the extraction algorithm. These equations are necessary to have a valid square Modelica model, but must be eliminated for data reconciliation.

The table below compares the results obtained with OpenModelica with those given by the VDI2048 standard.

Variables to be Reconciled	Initial Measured Values	Reconciled Values with OpenModelica	Reconciled Values from VDI2048	Initial Half-width Confidence Intervals	Reconciled Half-width Confidence Intervals with OpenModelica	Reconciled Half-width Confidence Intervals from VDI2048
mFDKEL	46.241	44.6959	44.696	2.5	1.61062	1.611
mFDKELL	45.668	44.1229	44.123	2.5	1.61062	1.611
mSPL	44.575	44.6426	44.643	0.535	0.425429	0.425
mSPLL	44.319	44.3861	44.386	0.532	0.423635	0.424
mV	0.525	0.524499	0.524	0.105	0.104627	0.105
mHK	69.978	70.0049	70.005	0.854	0.615089	0.615
mA7	10.364	10.3642	10.364	0.168	0.133112	0.133
mA6	3.744	3.74402	3.744	0.058	0.0566989	0.057
mA5	4.391	4.39102	4.391	0.058	0.0566989	0.057
mHDNK	18.498	18.4993	18.499	0.205	0.137264	0.137
mD	2.092	Not reconciled	2.092	0.272	Not reconciled	0.272

The value of mD is left unchanged after reconciliation. This is indicated by 'Not reconciled' in OpenModelica, and by repeating the initial measured values in VDI2048. In order to compute the reconciled values of mFD1, mFD2, mFD3 and mHDANZ, which have no measurements, it is possible to consider them as variables of interest with very large half-width confidence, e.g., 1e4, and assign them arbitrary measured values, e.g. 0. The result is

For the splitter, the results are the following:

34.4 Computing the Boundary Conditions from the Reconciled Values

The values and uncertainties of the boundary conditions that correspond to the reconciled values of the variables of interest can be computed by:

1. Selecting Data Reconciliation > Calculate Data Reconciliation

Variables to be Reconciled	Initial Measured Values	Reconciled Values with OpenModelica	Reconciled Values from VDI2048	Initial Half-width Confidence Intervals	Reconciled Half-width Confidence Intervals with OpenModelica	Reconciled Half-width Confidence Intervals fror VDI2048
mFDKEL	46.241	44.6959	44.696	2.5	1.61062	1.611
mFDKELL	45.668	44.1229	44.123	2.5	1.61062	1.611
mSPL	44.575	44.6426	44.643	0.535	0.425428	0.425
mSPLL	44.319	44.3861	44.386	0.532	0.423634	0.424
mV	0.525	0.524499	0.524	0.105	0.104627	0.105
mHK	69.978	70.0049	70.005	0.854	0.61509	0.615
mA7	10.364	10.3642	10.364	0.168	0.133112	0.133
mA6	3.744	3.74402	3.744	0.058	0.0566989	0.057
mA5	4.391	4.39102	4.391	0.058	0.0566989	0.057
mHDNK	18.498	18.4993	18.499	0.205	0.137264	0.137
mD	2.092	Not reconciled	2.092	0.272	Not reconciled	0.272
mFD1	***	88.714	88.714	***	0.613429	0.613
mFD2	***	88.714	88.714	***	0.613429	0.613
mFD3	***	88.714	88.714	***	0.613429	0.613
mHDANZ	***	18.4993	18.499	***	0.137264	0.137

Variables to be Reconciled	Initial Measured Values	Reconciled Values	Initial Half-width Confidence Intervals	Reconciled Half-width Confidence Intervals	Results of Local Tests	Values of Local Tests	Margin to Correctnes
splitter.pipe1.Q	2.5	2.43767	0.196	0.108462	TRUE	0.748276	1.21172
splitter.pipe1.Pm	610000	617668	39200	31584.6	TRUE	0.647353	1.31265
splitter.pipe1.T	292	290.015	14	8.25849	TRUE	0.344207	1.61579
splitter.pipe2.Q	1.15	1.17158	0.196	0.130814	TRUE	0.289819	1.67018
splitter.pipe2.Pm	255000	251925	39200	28890.4	TRUE	0.227449	1.73255
splitter.pipe2.T	386	387.853	13	7.79403	TRUE	0.349125	1.61087
splitter.pipe3.Q	1.25	1.26609	0.196	0.129549	TRUE	0.214432	1.74557
splitter.pipe3.Pm	245000	240406	39200	29129.9	TRUE	0.34324	1.61676
splitter.pipe3.T	388	387.859	13	7.79402	TRUE	0.026654	1.93335

2. Selecting the Boundary Conditions algorithm.
3. **Filling the Data Reconciliation form with**
    a. The name of the Reconciled Measurement File (mandatory). It is the name of the csv output file produced by the data reconciliation algorithm.
    b. The name of the Reconciled Correlation Matrix file (mandatory). It is the name of the correlation matrix csv file produced by the data reconciliation algorithm.
4. Clicking on Save Settings to save the above settings in the Modelica model.
5. Clicking on Calculate to launch the calculation.

At least one boundary condition must be declared in the model, otherwise the computation fails with a difficult to interpret error message such as

Cannot Compute Jacobian Matrix F.

A better error message will be posted in a future version.

The computation of the boundary conditions is performed in three main steps:

1. **Static analysis is performed on the model to extract the equations that compute the boundary conditions from the variables of interest. This corresponds to automatically inverting the original model. There are two groups of extracted equations:**
    a. The boundary conditions that are the equations that compute the boundary conditions.
    b. The intermediate equations that solve the intermediate variables from the variables of interest (this is the numeric way of eliminating the intermediate variables).

2. The model is simulated to compute the Jacobian matrices and eliminate numerically the intermediate equations. At this step, numerical simulation errors can occur, such as divisions by zero, non-convergence, etc. They can be corrected by improving the model or providing better start values.

3. The input files are read, the numerical calculations are performed, and the results are displayed. The half-width confidence intervals for the boundary conditions are calculated by propagating the reconciled uncertainties on the variables of interest through the inverted model.

The results are displayed:

1. In an html file with the title: Boundary Condition Report. This file pops up automatically when the calculation is completed.

2. In a csv output file. The name of the csv output file is <working directory>\<model name>\< model name>_ BoundaryConditions_Outputs.csv

The Boundary Condition Report has three sections:

34.4.1 Overview

a. Model file: name of the Modelica model file
b. Model name: name of the Modelica model
c. Model directory: name of the directory of the Modelica model file
d. Reconciled values input file: name of the csv output file produced by the data reconciliation algorithm
e. Reconciled covariance matrix input file: name of the correlation matrix csv file produced by the data reconciliation algorithm
f. Generated: date and time of the generation of the boundary condition report

34.4.2 Analysis

a. Number of boundary conditions.

b. Number of reconciled variables.

c. Boundary conditions: set of the boundary conditions (i.e., the equations that compute the boundary conditions).

d. Intermediate equations: set of the intermediate equations (i.e., the equations that compute the intermediate variables from the variables of interest).

34.4.3 Debug log

Log of the numerical iteration loop

34.4.4 Results

A table with the following columns

a. Boundary Conditions: the names of the boundary conditions.

b. Values: the computed values of the boundary conditions.

c. Reconciled Half-width Confidence Intervals: the half-with confidence intervals for the boundary conditions.

The results for the splitter are:

Boundary conditions	Values	Reconciled Half-width Confidence Intervals
splitter.sinkQ1.Q0	1.26609	0.129549
splitter.sourceP.P0	914780	53492.5
splitter.sinkQ.Q0	1.17158	0.130814
splitter.sourceP.T0	290.087	8.2886

34.5 Contacts

Daniel Bouskela (daniel.bouskela@edf.fr)
Audrey Jardin (audrey.jardin@edf.fr
Arunkumar Palanisamy (arunkumar.palanisamy@ri.se)
Lennart Ochel (lennart.ochel@ri.se)
Adrian Pop (adrian.pop@liu.se)

34.6 References

Bouskela, D., Jardin, A., Palanisamy, A., Ochel, L., & Pop, A. (2021). New Method to Perform Data Reconciliation with OpenModelica and ThermoSysPro. Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021.

VDI - Verein Deutscher Ingenieure. (2000). Uncertainty of measurement during acceptance tests on energy-conversion and power plants - Part 1: Fundamentals. VDI 2048 Blatt 1, October 2000.

CHAPTER

THIRTYFIVE

FREQUENTLY ASKED QUESTIONS (FAQ)

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

35.1 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**
  switched to this variable, in order not to interfere with other Modelica tools which might be installed on the same system. MODELICAPATH is not read at all: OPENMODELICALIBRARY does not extend or override it. Like MODELICAPATH, it is a list of directories, separated by colon (:) on all operating systems except Windows, which uses semicolon (;). If it is not set, OpenModelica defaults to OPENMODELICAHOME/lib/omlibrary/ and ~/.openmodelica/libraries/ (on Windows '~' is %APPDATA%). 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.

35.2 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.

CHAPTER

THIRTYSIX

MAJOR OPENMODELICA RELEASES

This Appendix lists the most important OpenModelica releases and a brief description of their contents. Right now versions from 1.3.1 to v1.27.1 are described.

36.1 Release Notes for nightly

36.2 Release Notes for OpenModelica 1.27.1

36.3 Patch release for 1.27.0.

- OMEdit can now re-simulate a model from compiled artifacts without requiring the Modelica source code and without the need of recompiling it #14111.
- Improved handling of parameter menus for instances of replaceable models #16248
- Many fixes to OMEdit, including some causing the GUI to crash.
- Fixed bug in the implementation of `spatialDistribution()`

Here is the complete list of closed issues.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.27.0...v1.27.1

36.4 Release Notes for OpenModelica 1.27.0

36.5 2026 summer release of OpenModelica version 1.27.0.

36.5.1 Main highlights:

- Experimental MCP server in OMEdit for interaction with AI agents.
- OMEdit now uses Qt6 instead of Qt5, better rendering on high-resolution monitors.
- State machines are now supported by the new frontend.
- Reverse model lookup.
- The Modifiers tab in the parameter dialog now collects all the modifiers applied to a component instance that were not previously displayed in the Parameters dialog.
- Improved formatting of parameter input dialogs.
- More responsive editing of diagrams in OMEdit.
- 60 Hz framerate animations of Multibody systems.
- Improved simulation of systems with implicit equations including if clauses.
- Previously compiled models can be re-simulated in a new session.
- Declarative equation-based debugger now handles models with more than 10.000 equations efficiently.

- FMU memory leaks fixed.
- Applying library conversion scripts preserves the original formatting.

36.5.2 OpenModelica Compiler (OMC)

Frontend

The new front end has been further improved with 29 issues resolved.

State machines ar now supported by the new frontend, see #8162; since there are basically no open-source libraries using them for testing, the implementation may still be buggy or incomplete, testing by OpenModelica users is very welcome.

Base Modelica export was improved, including alias simplification of flow variables in stream connectors for efficient mixed-integer optimization of thermo-fluid system models, -d=flowAliasElimination.

The use of equality among Reals outside functions, which is prohibited by the Modelica Language Specification for good numerical reasons, is now deprecated, see #14940. It will not be accepted by future versions of the compiler, unless specifically required by a `--allowNonStandardModelica` flag.

Backend and Code Generation

A bug in the resolveLoops module causing completely wrong results to be computed was fixed, see #13292. Overall, 8 issues were fixed in the currently used backend.

The work on the development of the new backend continued, with substantial improvements and over 80 pull requests. 41% of the Modelica Standard Library and 48% of the models of the Buildings library are now simulated successfully. 11 issues were closed.

Recall that the new backend, which is a lot more efficient than the currently used one, in particular when handling arrays, is still under development and experimentally available with the `--newBackend <https://openmodelica.org/doc/OpenModelicaUersGuide/latest/omchelptext.html#omcflag-newbackend>`__ compiler flag.

The experimental `initialSimplified()` operator was introduced in the new backend, see #11272. Its use and rationale are described in a paper submitted to the 2026 Asian Modelica Conference.

5 issues regarding code generation were also fixed.

C runtime

Several improvements were added to the internal nonlinear solver of GBODE, further improving its efficiency, which now can exceed multi-step methods such as DASSL and IDA for systems with many events. This can be activated with flags `-gbnls=internal`, `-gberr=embedded`.

A bug in the implementation of nonlinear solvers for systems including if-equations was fixed, drastically improving the speed and robustness of the solution for such systems, in particular piecewise-linear electrical circuit models, see #15718 for more details.

Overall 12 issues regarding the runtime were addressed.

36.5.3 Graphical Editor OMEdit

OMEdit 1.27.0 provides many new features and improvements:

- Qt6 is used instead of Qt5, leading to improved graphical rendering on wide high-resolution displays.
- Reverse model lookup: by right-clicking on a class in the Libraries Browser and selecting Find Usage, it is possible to get a list of all usages of that class in all the loaded model, see #12915.
- The whole list of modifiers added to an instantiated model which are not already represented in the General and custom parameter tabs are now visible in the Modifiers tab, see #14372. This includes:
  - binding equations for variables and input variables;
  - min/max/nominal attribute modifiers;

  - modifiers and redeclares applied to sub-components, possibly by the Show Element feature, which are now visible in the parameter dialog, tab Modifiers, of the top-level component.

- The formatting of input and text fields of parameter input dialogs was further improved.
- Many diagram editing operations that were previously very slow are now much faster, see #14804.
- It is now possible to re-simulate a model that fails when simulating for the first time; previously, one had to recompile it, see #5472.
- It is now possible to re-simulate a model that was compiled in a previous session, see #14111.
- Multibody animations now run at 60 Hz framerate, see #14851.
- The declarative equation-based debugger can now handle large models over 10,000 equations, see #9977.
- Applying conversion scripts for newer versions of the used libraries preserves the original formatting, see #13447.

Many OMEdit bugs were also fixed in this release. Overall, 52 issues were addressed.

36.5.4 FMI export

Some bugs causing FMU memory leaks and eventually leading to out-of-memory errors were fixed, see #12225, #13488, and #14509. A bug causing failure when exporting FMUs with directional derivatives was also fixed, see #15537.

Overall, 12 issues regarding FMI export were addressed.

36.5.5 Experimental MCP server in OMEdit for interaction with AI agents

An experimental MCP server was implemented in OMEdit to allow AI agents to interact with OMEdit by writing, modifying, and running models, and then by analyzing the plotted results. See #15385 and follow up #15854. This server is switched off by default and can be activated by following these instructions. You are welcome to contribute to the discussion proposing ideas and requesting features in #15854.

36.5.6 List of tickets closed in 1.27.0

List of closed tickets in the 1.27.0 release. Over 160 issues were addressed and resolved.

36.5.7 Next releases

The next release 1.28.0 is planned to be released at the end of 2026.

A major milestone in 1.28.0 will be FMI3 support. We are also planning to include much improved handling of conditional connectors and further improvements to the GUI such as a restructured Simulation Setup dialog, faster editing of large models in OMEdit, better plotting of clocked variables, support of the Figure annotation, etc.

We are also actively experimenting the option of running OMEdit in the browser, using local hardware resources through Wasm technology. A mature enough version should become available when 1.28.0 is released.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.9...v1.27.0

36.6 Release Notes for v1.26.9

36.6.1 What's Changed

- Fix for #15808 parameter value width by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15828

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.8...v1.26.9

36.7 Release Notes for v1.28.0-dev

36.8 Release Notes for OpenModelica v1.26.8

36.8.1 What's Changed

- nothing changed, just fixing some github actions

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.7...v1.26.8

36.9 Release Notes for OpenModelica v1.26.7

36.9.1 What's Changed

- nothing changed, just fixing some github actions
- fix copy-paste error by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15474

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.6...v1.26.7

36.10 Release Notes for OpenModelica 1.26.6

36.10.1 What's Changed

Nothing has changed since v1.26.4. Release 1.26.5 and 1.26.6 are made just to tests our github actions.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.5...v1.26.6

36.11 Release Notes for OpenModelica 1.26.5

36.11.1 What's Changed

- add the workflows from master by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15454
- remove the wip workflow by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15457

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.4...v1.26.5

36.12 Release Notes for OpenModelica 1.26.4

36.13 Patch release for 1.26.3.

36.13.1 What's Changed

- Check for Expression.isNull() (#15116) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/15138
- update 3rdParty to build on resolute by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15421
- force qt6 on resolute by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15422
- Fix compilation on clang 21.1.8 by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15426
- fix for #15300 - filter on discrete variables (#15302) by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/15429

Here is the complete list of closed issues.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.3...v1.26.4

36.14 Release Notes for OpenModelica 1.26.3

36.15 Patch release for 1.26.2.

Fixes this bug: #15080 - Regression: start value not displayed

Here is the complete list of closed issues.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.2...v1.26.3

36.16 Release Notes for OpenModelica 1.26.2

36.17 Patch release for 1.26.1.

Fixes three bugs:

- #14810 was fixed in 1.26.1 but introduced a regression in the C++ runtime. This is now fixed.
- #14926 units were changed when the decimal dot was inserted in parameter editing fields. This is now fixed.
- #14894 supporting the double y-axis introduced a regression on the size of new plot windows. This is now fixed.

Here is the complete list of closed issues.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.1...v1.26.2

36.18 Release Notes for OpenModelica 1.26.1

36.19 Patch release for 1.26.0.

Fixes two critical bugs introduced in 1.26.0.

- #14848 was due to PR #14819, which resolved #10145. The bug caused OMEdit 1.26.0 to crash while typing non-trivial expressions in parameter input fields, if the currently displayed expression was invalid, e.g., when starting to type a string constant " or an array {.
- #14861 was due to PR #14774 which resolved #11721. The bug caused wrong redeclare clauses to be generated when using drop-down menus to redeclare replaceable classes, e.g. replaceable Medium models from Modelica.Media

Here is the complete list of closed issues.

Windows beta releases available here

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.0...v1.26.1

36.20 Release Notes for OpenModelica 1.26.1 Beta 2

36.21 Patch release for 1.26.0.

Fixes two critical bugs introduced in 1.26.0.

- #14848 was due to PR #14819, which resolved #10145. The bug caused OMEdit 1.26.0 to crash while typing non-trivial expressions in parameter input fields, if the currently displayed expression was invalid, e.g., when starting to type a string constant " or an array {.
- #14861 was due to PR #14774 which resolved #11721. The bug caused wrong redeclare clauses to be generated when using drop-down menus to redeclare replaceable classes, e.g. replaceable Medium models from Modelica.Media

Here is the complete list of closed issues.

Windows beta releases available here

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.26.0...v1.26.1-dev.beta.2

36.22 Release Notes for OpenModelica 1.26.0

36.23 2025 Winter release of OpenModelica version 1.26.0.

36.23.1 Main highlights:

- OMEdit now allows to **load and save** models and packages with **syntax errors**.
- **Improved sizing of parameter editing dialogs** in OMEdit.
- OpenModelica now allows to **use ``break`` to**remove modifiers**and for**selective model extension.
- OpenModelica now implements **less restrictive rules for the use of conditional components**, as specified in the draft of the next Modelica Language Specification.
- **Improved operation of debugging features in OMEdit**: the generation of Equation Operations in the Equation-Based Debugger is now activated by default and it is also possible to activate profiling at runtime after running a model for the first time.
- Improved handling of **large result files** in OMEdit.
- Old **deprecated and poorly supported solvers were removed from the runtime** - gbode should be used instead.

36.23.2 OpenModelica Compiler (OMC)

Selective model extension was introduced in #11381.

New, less restrictive rules that are now defined for conditional components in the draft version of Modelica 3.7 were implemented in #12888; basically, it is now possible to refer to conditionally defined components outside of connect statements, as long as they are actually defined.

The new front end has been further improved with 25 issues resolved.

Regarding backend work, 12 issues were fixed in the currently used backend.

The work on the development of the new backend continued, with 31 issues fixed. Recall that the new backend, which is a lot more efficient in particular when handling arrays, is still under development and experimentally available with the `--newBackend <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html#omcflag-newbackend>`__ compiler flag.

13 issues regarding code generation were also fixed.

Regarding the C runtime, a new solver strategy was implemented in GBODE, which drastically reduces the number of iterations of the nonlinear solver in the implicit integration methods at each stage, while preserving the same accuracy of the solution; see also the discussion in #14089, #14022. This can be activated with flags `-gbnls=internal`, `-gberr=embedded`; only works with single-rate at the moment, but will be extended to multi-rate integration in the next release.

Old poorly supported and deprecated solvers (see #9191) were finally removed from the runtime. They are replaced by better implemented algorithm available within the GBODE solver.

Overall 21 issues regarding the runtime were addressed.

36.23.3 Graphical Editor OMEdit

OMEdit 1.26.0 provides several new features:

- It is now possible load and save models and packages with syntax errors. Until all errors are fixed, the code can be edited in text mode, with a direct mapping on the file system; then, it can be saved and re-loaded with the standard Modelica view, see #13663.
- Much better sizing of parameter input dialogs. Lenghty comments in drop-down menus are now displayed in tooltips, avoiding the need of excessively wide parameter input windows, see #11721.
- Improved operation of debugging features: the generation of Equation Operations in the Equation-Based Debugger is now activated by default and it is also possible to activate profiling at runtime after running a model for the first time.
- Loading large result files in OMEdit is now much faster and more reliable.

Many OMEdit bugs were also fixed in this release. Overall, 39 issues were addressed.

36.23.4 FMI export

CMAKE FMU export is now the default option. A critical bug was resolved about changes in discrete input variables, which did not generate events in FMI ME, see #13822. Overall, 10 issues regarding FMI export were addressed.

36.23.5 OMPython

OMPython 4.0.0 was released on Oct 20, 2025. See the release notes.

36.23.6 List of tickets closed in 1.26.0

List of closed tickets in the 1.26.0 release. Overall, 170 issues were addressed and resolved.

36.23.7 Next releases

The next release is planned to be released in spring 2026. It should include much improved handling of conditional connectors and further improvements to the GUI such as a restructured Simulation Setup dialog, faster editing of large models in OMEdit, better plotting of clocked variables, support of the Figure annotation, etc. Further improvements in FMI export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.25.7...v1.26.0

36.24 Release Notes for OpenModelica 1.26.0-dev.beta.1

36.24.1 2025 winter release of OpenModelica 1.26.0 Beta 1 version.

Main highlights:

- OMEdit now allows to **load and save** models and packages with **syntax errors**.
- **Improved sizing of parameter editing dialogs** in OMEdit.
- OpenModelica now allows to **use ``break`` to**remove modifiers**and for**selective model extension.
- OpenModelica now implements **less restrictive rules for the use of conditional components**, as specified in the draft of the next Modelica Language Specification.
- **Improved operation of debugging features in OMEdit**: the generation of Equation Operations in the Equation-Based Debugger is now activated by default and it is also possible to activate profiling at runtime after running a model for the first time.
- Improved handling of **large result files** in OMEdit.
- Old **deprecated and poorly supported solvers were removed from the runtime** - gbode should be used instead.

36.24.2 OpenModelica Compiler (OMC)

Selective model extension was introduced in #11381.

New, less restrictive rules that are now defined for conditional components in the draft version of Modelica 3.7 were implemented in #12888; basically, it is now possible to refer to conditionally defined components outside of connect statements, as long as they are actually defined.

The new front end has been further improved with 25 issues resolved.

Regarding backend work, 12 issues were fixed in the currently used backend.

The work on the development of the new backend continued, with 29 issues fixed. Recall that the new backend, which is a lot more efficient in particular when handling arrays, is still under development and experimentally available with the `--newBackend <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html#omcflag-newbackend>`__ compiler flag.

13 issues regarding code generation were also fixed.

Regarding the C runtime, a new solver strategy was implemented in GBODE, which drastically reduces the number of iterations of the nonlinear solver in the implicit integration methods at each stage, while preserving the same accuracy of the solution; see also the discussion in #14089, #14022. This can be activated with flags `-gbnls=internal`, `-gberr=embedded`; only works with single-rate at the moment, but will be extended to multi-rate integration in the next release.

Old poorly supported and deprecated solvers (see #9191) were finally removed from the runtime. They are replaced by better implemented algorithm available within the GBODE solver.

Overall 20 issues regarding the runtime were addressed.

36.24.3 Graphical Editor OMEdit

OMEdit 1.26.0 provides several a few new features:

- It is now possible load and save models and packages with syntax errors. Until all errors are fixed, the code can be edited in text mode, with a direct mapping on the file system; then, it can be saved and re-loaded with the standard Modelica view, see #13663.
- Much better sizing of parameter input dialogs. Lenghty comments in drop-down menus are now displayed in tooltips, avoiding the need of excessively wide parameter input windows, see #11721.
- Improved operation of debugging features: the generation of Equation Operations in the Equation-Based Debugger is now activated by default and it is also possible to activate profiling at runtime after running a model for the first time.
- Loading large result files in OMEdit is now much faster and more reliable.

Many OMEdit bugs were also fixed in this release. Overall, 35+ issues were addressed.

36.24.4 FMI export

CMAKE FMU export is now the default option. A critical bug was resolved about changes in discrete input variables, which did not generate events in FMI ME, see #13822. Overall, 10 issues regarding FMI export were addressed.

36.24.5 OMPython

OMPython 4.0.0 was released on Oct 20, 2025. See the release notes.

36.24.6 List of tickets closed in 1.26.0

List of closed tickets in the 1.26.0 release. Over 160 issues were addressed and resolved.

36.24.7 Next releases

The next release is planned to be released in spring 2026. It should include much improved handling of conditional connectors and further improvements to the GUI such as a restructured Simulation Setup dialog, faster editing of large models in OMEdit, better plotting of clocked variables, support of the Figure annotation, etc. Further improvements in FMI export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.25.7...v1.26.0-dev.beta.1

36.25 Release Notes for OpenModelica 1.25.7

Patch release of 1.25.6. Fixes a bug introduced in 1.25.6 that caused OMEdit to crash.

36.26 Release Notes for OpenModelica 1.25.6

Patch release of 1.25.5.

Fixes 6 issues of the OMEdit GUI, regarding animations with CAD files, OMEdit crashes, hierarchical editing. Also fixes a regression that caused slow loading of large result files.

36.27 Release Notes for OpenModelica 1.25.5

Patch release of 1.25.4. Fixes a couple of issues with FMI export, including a regression introduced in 1.25.4, as well as a couple of bugs in OMEdit.

Here is a complete list of the fixed issues.

36.28 Release Notes for OpenModelica 1.25.4

36.28.1 What's Changed

- Construct the qualified path of variable (#14341) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/14342
- Clear the hash (#14344) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/14345

This is bug fix release that fixes a regression with the plotting in OMEdit introduced in v1.25.3.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.25.3...v1.25.4

36.29 Release Notes for OpenModelica 1.25.3

Patch release of 1.25.2. Fixes 4 issues, concerning the GUI, inlining, and FMI export.

In particular, #14192 significantly improves the peformance and reliability of OMEdit when browsing simulation results of models of non-trivial size.

36.30 Release Notes for OpenModelica 1.25.2

Patch release of 1.25.1. Fixes 4 issues with the GUI that were found in 1.25.1.

Activates building using qt6 webengine by default on Linux Debian Trixie LTS, since qt6 webkit is no longer available.

36.31 Release Notes for OpenModelica 1.25.1

Patch release of 1.25.0. Fixes 17 issues that were found in 1.25.0, mostly involving the GUI.

Please use this version of OMC if you plan on installing the Modelica Standard Library version 4.1.0, as it contains a critical fix regarding version management in the class loader.

36.32 Release Notes for OpenModelica 1.25.0

This is the final release of the regular early-spring release of OpenModelica, version 1.25.0.

Main highlights: OMEdit now supports **hierarchical parameter editing** in structured models, a **Reload package** feature, and an **Open Class in text view** feature.

36.32.1 OpenModelica Compiler (OMC)

The new front end has been further improved with 20+ issues solved. Three fixes regarding the new experimental Base Modelica flat output.

Regarding backend work, 9 issues were fixed in the currently used backend. In particular, #13187 fixed a long-standing issue that caused unexplicable divisions by zero in the Buildings library. Additionally, 18 issues were address in the new backend, which is still under development and experimentally available with the `--newBackend <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html#omcflag-newbackend>`__ compiler flag.

Regarding the C runtime, 15 issues were addressed. In particular, #5709 fixed an old problem that caused equidistant time grid simulation result outputs to miss some time steps if the saving of results at events was disabled - this now allows to obtain fully periodic sampling times in simulation outputs with equidistant time grid in that case.

A new experimental feature, the Newton Diagnostics, was added to the C runtime. This feature can be activated by the simulation flag `-lv=LOG_NEWTON_DIAGNOSTICS <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/simulationflags.html#simflag-lv>`__; this computes and displays a ranking of the most likely start attributes responsible for the failure of Newton-Raphson iterations on nonlinear implicit systems during initialization. As such, it can be an invaluable tool for troubleshooting initialization failures due to convergence issues caused by badly chosen start attributes. For more details see the paper by F. Casella and B. Bachmann on this topic (here is an open-access preprint on ArXiv) and the presentation at the 2025 OpenModelica Workshop for more details. The current implementation is still incomplete, as it doesn't work yet with symbolic jacobians and does not perform residual scaling, so it is not going to work well yet when applied to systems of equations that are badly scaled due to the use of SI units. A full implementation will be available in the next release.

36.32.2 Graphical Editor OMEdit

OMEdit 1.25.0 provides several, long awaited-for features.

The first is hierarchical editing of model parameters #2891: you can now change parameters not only for top-level components, but also for its sub-components. This feature is available by right-clicking over component icons and selecting Show Element; this can be done recursively to set parameters of sub-sub-components, etc.. Replaceable components and classes can be handled in the same way. Previously, this was only possible by switching to the textual input mode and by manually adding the corresponding nested modifiers and redeclare clauses to the top-level component instances.

The Reload feature was implemented #5664: if you right-click on a top-level package, the whole package is unloaded from memory and reloaded from the file system, *while preserving the status of the already opened sub-packages in the Library browser*. Previously, you had to first select Unload, then re-open the package, and then navigate again through the sub-package hierarchy to reach the model(s) of your interest. This feature is very convenient when working with multiple Modelica tools in parallel, or when editing some parts of a library with textual editors, because it allows to update the loaded package to the updated version on the file system with a single mouse click.

When browsing the code of a class in the textual view, it is now possible to open any class mentioned in the source code by just right-clicking on it and selecting Open Class, or just by the ctrl-click shortcut #8854.

It is now possible to select a part of a model diagram by dragging a selection window, then copy and paste all the selected components either in the same model, or into a new model #12687.

OMEdit now automatically adds the `each` prefix when providing scalar parameter values to parameter arrays, avoiding the common error of forgetting to add it manually #13636.

The new instance-based infrastructure of OMEdit is now fully mature and far superior to the old graphical editing mode from all points of view. Hence, starting from version 1.25.0, support of the old graphical editing mode is dropped #13257.

Many OMEdit bugs were also fixed in this release. Overall, 40+ issues were addressed.

36.32.3 FMI export

Several issues regarding FMI export were addressed.

A serious bug #13582 affecting co-simulation FMUs using the CVODE stiff solver was solved, allowing to use FMI-CS on stiff models.

36.32.4 API documentation

The online documentation of the interactive environment API was significantly improved in terms of completeness and readability.

36.32.5 List of tickets closed in 1.25.0

List of closed tickets in the 1.25.0 release. 140 issues were addressed and resolved.

36.32.6 Next releases

The next release is planned to be released towards the end of 2025. It should include further improvements to the GUI such as a restructured Simulation Setup dialog, faster editing of large models in OMEdit, etc. Further improvements in FMI 2.0.x export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.24.5...v1.25.0

36.33 Release Notes for OpenModelica 1.24.5

Added flag to avoid non-strictly necessary compile-time evaluations, see #13570.

Required by RTE to compile Dynawo code on 1.24.x using C11 instead of C17.

36.34 Release Notes for OpenModelica 1.24.4

Some more patches for the 1.24.x, see list.

36.35 Release Notes for OpenModelica 1.24.3

Patch release for 1.24.2

Fixes 6 issues, most importantly a regression involving start attributes for state variables not showing up in the parameter input dialog.

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.24.2...v1.24.3

36.36 Release Notes for OpenModelica 1.24.2

Patch release of 1.24.0, fixes issues with the OMEdit GUI. Replaces the previously planned 1.24.1 which was skipped.

A list of the fixed issues is available here.

36.36.1 What's Changed

- Hide the start and fixed attributes based on final modifier (#13007) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/13008
- Add context to `Inst.expand` (#12962) by @perost in https://github.com/OpenModelica/OpenModelica/pull/13023
- OMEdit fixes by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/13081
- Fixes 1.24.1 by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/13100
- Cpp FMI: don't check clockSubactive in firstTick of simulation code by @rfranke in https://github.com/OpenModelica/OpenModelica/pull/13120
- Fix the icon display if the component does not have annotation by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/13200
- Draw a default grey box if the model doesn't have an icon by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/13216
- Cherry-pick instance API fixes by @perost in https://github.com/OpenModelica/OpenModelica/pull/13218

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.24.0...v1.24.2

36.37 Release Notes for OpenModelica 1.24.0

The release date of 1.24.0 was anticipated with respect to the traditional summer release and winter release date. The main motivation is that a major update of the OMEdit source code was made on the master branch to make it compatible with both Qt5 and Qt6. This makes it hard to back-port crucial fixes to the GUI in patch releases before the formerly scheduled release date of Jan-Feb 2025. The current development version on the master branch is quite stable and there are no major complaints, which allow us to anticipate the next official release to September 2024.

36.37.1 OpenModelica Compiler (OMC)

The new front end has been further improved with about 25 issues solved, here is a complete list. Nine of those issues regarded fixes to the new experimental Base Modelica flat output.

A new compiler flag --exposeLocalIOs was added to automatically add the outputs of a model components to the outputs of an FMU, instead of only adding the top-level outputs. This can easily provide monitoring information in an FMU without the need of painstakingly bring all those outputs to the top level in the original Modelica model.

Besides #12730, the work on the backend was mostly focused on developing the new backend, with 14 issues being fixed.

One small fix was made to the C runtime.

36.37.2 Graphical Editor OMEdit

The OMEdit source code was adapted to use both Qt5 and Qt6, see pull request #12485.

The quality of the OMEdit GUI was further improved over the previous version 1.23.1, with about 15 issues fixed, see the complete list.

An important improvement regarding the usability of the GUI concerns the reaction time when editing diagrams, which used to be very slow when adding or removing components and connections in large models; these operations are now much faster, while changing parameters can still be quite slow. See #11920 for more details.

36.37.3 FMI export

A few issues were addressed regarding FMI export.

36.37.4 List of tickets closed in 1.24.0

List of closed tickets in 1.24.0. Over 65 issues were addressed and resolved.

36.37.5 Next releases

The next release 1.25.0 is planned to be released in January/February 2025, with a first beta release possibly before the end of the year. It should include further improvements to the GUI such as parameter editing in hierarchically structured model, restructured Simulation Setup dialog, Rename feature with refactoring, faster editing of large models in OMEdit, etc. Further improvements in FMI 2.0.x export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.23.1...v1.24.0

36.38 Release Notes for OpenModelica 1.23.1

This is a minor release with fixes to OMEdit and omc to avoid crashes.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.23.0...v1.23.1

36.39 Release Notes for OpenModelica 1.23.0

Version 1.23.0 of OpenModelica was released on 6 June 2024. It has many improvements and bug fixes compared to 1.22.3 (1.22.4 for Linux), plus some new useful features, in particular regarding the OMEdit GUI and the FMI export feature.

36.39.1 Proper rounding of decimal numbers

Real parameter values returned by the front-end when flattening models often had bad rounding, such as `499.99999999999` instead of `500` or `1.00000000000012` instead of `1.0`. Also, bad rounding used to plague parameter values with unit conversions driven by `displayUnit` attributes in the OMEdit GUI. In some cases, very long decimal outputs such as `1200000000.0` were displayed, instead of the more easily understood `1.2e9`.

This issue was tackled systematically and resolved thanks to rounding algorithms based on the Ryu library, see #8865 for a comprehensive list of all improvements. Now all Real values numbers that are expected to be represented as round decimal numbers are actually represented as such; additionally, decimal vs. scientific format is chosen according to heuristic rules, to ensure good readability by humans in all cases, without significant loss of precision.

36.39.2 OpenModelica Compiler (OMC)

The new front end has been further improved with about 25 issues solved, here is a complete list.

The front end can now provide clean BaseModelica flat output from the command line; experimental array-preserving flat output can also be obtained. See the User's Guide for more information.

The draft MLS 3.7 Sect. 7.1 now explicitly requires annotations at the end of a class to be carried over when extending model; this includes experiment annotations. This behaviour is now implemented in the frontend, so it is no longer necessary to re-add the experiment annotation when extending a class, see #10386.

Most of the work on the backend is still focused on developing the new backend; about 10 fixes were made to the current backend. Two notable improvements are worth mentioning:

- the non-standard `annotation(tearingSelect)` was deprecated and replaced by `annotation(__OpenModelica_tearingSelect)`

- according to the Modelica Language Specification Sect. 8.6 parameters with no binding and no explicitly set value now cause the compilation to fail, see #10386. This may cause some old models to fail during compilation, but it is the required behaviour by the specification, to guard against parameters that are accidentally left without a meaningful value. The fix is to provide the modifier for the parameters without bindings, or to add an explicit `start = 0` modifier in the library model.

About 10 bug fixes were made to the C runtime.

36.39.3 Graphical Editor OMEdit

The quality of the OMEdit GUI was further improved over the previous version 1.22.3, with about 55 issues fixed, see the complete list. Most of them are bug fixes, but there are also some new features:

- Duplicate class feature now works correctly also when the class is duplicated in another scope #5346
- Top-level model parameters can now be edited with the GUI #4408
- Further optimization of the editor's response time when deleting components and adding/deleting connections #11920
- All rounding issues in the GUI resolved #8865

36.39.4 FMI export

14 issues were addressed regarding FMI export, further improving the quality of that feature. We are currently testing over 80 libraries from the Package Manager collection by exporting FMI-CS FMUs with OMC and simulating them with CVODE using the FMPy tool; test reports can be reached from this page.

36.39.5 List of tickets closed in 1.23.0

List of closed tickets in 1.23.0. Over 140 issues were resolved.

36.39.6 Next releases

The next release 1.24.0 is planned to be released in Q4 2024. It should include further improvements to the GUI such as parameter editing in hierarchically structured model, restructured Simulation Setup dialog, Rename feature with refactoring, faster editing of large models in OMEdit, etc. Further improvements in FMI 2.0.x export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.22.3...v1.23.0

36.40 Release Notes for OpenModelica 1.22.3

This patch release contains several fixes and improvements to the OMEdit GUI. See the full list of the closed issues here.

36.41 Release Notes for OpenModelica 1.22.2

This patch release contains several fixes and improvements, mostly to the OMEdit GUI

- a critical bug when handling connections that causes OMEdit to crash when editing them was fixed
- moving components around in diagrams in OMEdit is now much faster and responsive
- several fixes to diagram visualization and parameter inputs in OMEdit
- much faster FMU code export thanks to the clang compiler and parallel C source code compilation

A full list of closed issues can be found here.

36.42 Release Notes for OpenModelica 1.22.1

This patch release contains several improvements to the OMEdit GUI:

- It is now possible to handle 'each' and 'final' prefixes in parameter input dialogs
- It is possible to close the Message Browser window if one wants to maximize the real estate for result plots. Only the status bar is displayed also over multiple re-runs or re-compilations, unless warnings or errors pop up
- Some fixes to visualization glitches and OMEdit unexpected crashes

For the full list of improvements, see here.

36.43 Release Notes for OpenModelica 1.22.0

The main highlight of the 1.22.0 release of OpenModelica is a significant improvement of the OMEdit GUI, made possible by the new instance-based architecture for the interaction between the OMEdit GUI and the underlying OMC compiler, which is now activated by default; see below for further details. Many bugs were also fixed, leading to a much improved user experience, compared to the previous 1.21.0 release.

36.43.1 OpenModelica Compiler (OMC)

The new front end has been further improved with about 25 bug fixes, here is a complete list. The conversion from Integer to Enumeration, which was introduced in Modelica 3.4, is now supported by OMC.

Most of the work on the backend is currently focused on developing the new backend; a few fixes were made to the current backend. The `--tearingMethod=noTearing <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html#omcflag-tearingmethod>`__ was deprecated, because it breaks models with discrete variables involved in algebraic loops, `--tearingMethod=minimalTearing <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html#omcflag-tearingmethod>`__ should be used instead. A few bug fixes were also made to the code generation process.

Several bug fixes and improvements were made to the C runtime. A new simulation flag `-lv_system <https://openmodelica.org/doc/OpenModelicaUsersGuide/1.22/simulationflags.html#simflag-lv-system>`__ was added, to enable logging of solver behaviour for selected systems; together with `-lv_time <https://openmodelica.org/doc/OpenModelicaUsersGuide/1.22/simulationflags.html#simflag-lv-time>`__, this allows to avoid huge log files when debugging solver issues taking place at some point in time with some specific system of equations.

Several improvements were made to the C++ runtime:

- support for ExternalMedia library (fix record inputs of external functions and bindings of records to scalar variables)
- support for ThermofluidStream library (fix array dimensions of created temporary variables)
- preliminary support for DAE mode
- fixed issue with start attributes for iteration variables of implicit nonlinear systems, which were previously lost

36.43.2 Graphical Editor OMEdit

As mentioned above, the new instance-based infrastructure for model editing is now firmly in place and is activated by default. This supports long awaited-for features, namely conditional connectors and parameter input dialogs activated by boolean variables, replaceable classes and models with parameter editing. It also improves the support of the DynamicSelect annotation. In case of remaining issues, it is still possible to de-activate the new infrastructure and fall back to the old one, using the checkbox *Tools* | *Options* | *General* | *Optional Features* | *Disable new instance-based graphical editing of models*.

Connection handling and rendering is also much improved:

- it is no longer possible to make connections between incompatible connectors (that previously only failed later on when trying to compile the model);

- connection lines are drawn with a white background that allows to reveal bogus connections that were previously hidden by the component itself (e.g., inadvertently short-circuiting a resistor or capacitor). It also show clearly that two apparently crossing connection lines have no actual connection to each other;
- connection lines spanning a connection set with more than one connector are now rendered with dot-like branching points, increasing readability.

Over 70 issues were successfully addressed in the OMEdit graphical editor. About 35 issues concerned the debugging and fine-tuning of the new instance-based infrastructure.

36.43.3 Scripting interfaces

Improvement made to the OMMatlab concening the efficiency of the `linearize()` API function call were ported to OMPython and OMJulia. The updated implementation avoids recompiling the model from scratch every time when multiple calls to `linearize()` are made on the same model, resulting in dramatic performance improvement.

36.43.4 FMI support

Further development work was carried out to support source-code FMI export across all operating systems using CMAKE, see closed tickets, in addition to the old workflow that was based on GNU makefiles and only worked in POSIX environments. This functionality, documented here, is now mostly in place, though it requires more testing to become fully usable. To get the latest developments in FMI export support, consider trying the 1.23.0-dev nightly builds.

A new flag --nonStandardExposeLocalIOs was introduced to export input/output connectors of sub-models, like sources, sinks and sensors. The default value 0 expose the top-level public inputs and outputs, the value 1 also exposes public inputs and outputs of components, the value 2 also exposes public inputs and outputs of components within components, and so on.

36.43.5 List of tickets closed in 1.22.0

List of closed tickets in 1.22.0. 140 issues were resolved.

36.43.6 Next releases

The next release 1.23.0 is planned to be release in Q2 2024. It should include further improvements to the GUI such as parameter editing in hiearchically structured model, fully functional Duplicate feature, restructured Simulation Setup dialog, etc. Further improvements in FMI 2.0.x export are also planned.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.21.0...v1.22.0

36.44 Release Notes for OpenModelica 1.21.0

The 1.21.0 release of OpenModelica doesn't contain any major new feature, but it contains many improvements and bugfixes that significantly improved the quality of the software with respect to the previous 1.20.0 release.

The new single-step gbode solver has been further developed and can now be used as a serious alternative to the more established dassl, ida, and cvode solvers. A full-fledged GUI for its configuration is still not available and will be added in future versions. Currently, the method defaults to esdirk4, a general-purpose 4-th order implicit method. Other methods can be set with the -gbm simulation flag, and each method now automatically selects the best advanced options. All solvers use variable step-size with error control by default, but they can be turned to fixed time step by adding the -gbctrl=const simulation flag.

A completely overhauled architecture for the interaction between the OMEdit GUI and the underlying OMC compiler is now experimentally available in 1.21.0. It relies on the new frontend to instantiate the classes being edited, rather than trying to re-do it from scratch on the GUI side. This architecture provides faster user interaction and allowed to fix many shortcomings of the GUI, e.g. lack of proper support for replaceable classes with parameters, conditional connectors and parameter dialog activation depending on parameters, etc.

36.44.1 OpenModelica Compiler (OMC)

The new front end has been further improved with about 20 bug fixes, here is a complete list. Major development work went into supporting the development and fine-tuning a new API for the the OMEdit GUI, see tickets.

Several fixes were made to the current backend, although development work is now mostly focused on the new backend, which includes array-preserving code generation features. The new backend is still not capable of running most models from mainstream libraries. However, it is starting to show interesting results on selected large-scale test models, with size up to a million equations. These models can now be handled with reasonable amounts of time required to build and compile the simulation code. The new backend can be activated with the `--newBackend` compiler flag.

Several bug fixes were made to the code generation process. The new CMake-based build infrastructure has been further developed and is now nearly complete, including support for FMI generation.

Many bug fixes and improvements were made to the C runtime, mostly regarding the new gbode solver. This solver is fast reaching production quality and can now be used as a serious alternative to the mainstream dassl, ida, and cvode solvers; it also replaces deprecated obsolete solvers such as impeuler, trapezoid, imprungekutta, etc., that will be removed in future versions.

36.44.2 Graphical Editor OMEdit

About 60 issues were successfully addressed in the OMEdit graphical editor. Most notably, the performance of the GUI support for replaceable classes has been improved dramatically; therefore that option, which is required to use Modelica.Media and Modelica.Fluid, is no longer optional but always active by default.

About half of the addressed issues concern the development and fine tuning of the new instance-based interface, that is going to be much faster than the existing one and to address long-standing issues such as supporting conditional connectors and replaceable classes with parameter modifiers.

This new interface can be activated with the *Tools* | *Options* | *General* | *Enable instance API* option. It is mostly in place now, but it still requires further testing and debugging before it can replace the existing one for good. If you want to experiment with it, we suggest you to use the latest 1.22.0-dev nightly build version, that incorporates the latest developments and bug fixes. Please try that one before reporting issues on the bug tracker.

36.44.3 Scripting interfaces

The OMMatlab interface was significantly improved in this version, see tickets. In particular, it is now possible to linearize a system in many different conditions without recompiling it each time, which is useful model-based control design.

36.44.4 List of tickets closed in 1.21.0

List of closed tickets in 1.21.0. Over 130 issues were resolved.

36.44.5 Next releases

The next release 1.22.0 is planned to be release in Q4 2023. It will include the new instance-based editing in OMEdit, featuring support of conditional connectors, parameter-depending dialog annotation, hierarchical editing of models, and faster rendering of complex diagrams.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.20.0...v1.21.0

36.45 Release Notes for OpenModelica 1.20.0

The first most notable feature of this release is the automatic installation of the Modelica Standard Library (MSL), which is now fully integrated with the Package Manager. Every time any tool of the OpenModelica suite (e.g. OMEdit, OMNotebook, OMShell, etc.) attempts to load a version of the MSL, it checks if it has already been installed in the user's set of system libraries, which is handled by the Package Manager and is located in the user's .openmodelica directory. If that directory is empty, either because of a fresh install, or because it has been deleted,

then the MSL is automatically installed in the user's system libraries from cached zip files in the OpenModelica installation directory; in fact, both MSL 3.2.3 (which is backwards compatible with 3.2.2 and 3.2.1) and 4.0.0 are installed, so that any library created during the last ten years can run out of the box.

This automatic installation does not require any Internet connection, so it also works behind corporate firewalls or in situations with limited available bandwidth. This solution uses the same package manager that is also used to install other system libraries, contrary to the solution implemented in versions 1.18.0 and 1.19.x, which used two different directories in the MODELICAPATH, one for the package manager and one for the preinstalled MSL, leading to slightly confusing duplicate installations of MSL.

OMEdit loads MSL 4.0.0 by default in the Libraries Browswer. However, if one then loads a package using MSL 3.2.3, it is possible to unload MSL 4.0.0 and load MSL 3.2.3 just with a click of a button.

The second most notable feature is that a new general purpose ODE solver, named `gbode`, was introduced. This solver is a fully configurable single-step solver, supporting many different integration methods, both explicit and implicit, using either fixed time step or variable time step with error control, handling event detection and dense output for accurate resampling over a regular time grid. Implemented methods include Euler, Heun, Dormand-Prince, Gauss, Radau, Lobatto, Adams-Moulton, Fehlberg, SDIRK, ESDIRK, etc. Adaptive multi-rate algorithms are also available within this solver, although this feature is still experimental. This solver replaces previously available solvers like `euler`, `impeuler`, `trapezoid`, etc., which are now deprecated and will be removed in future versions of the tool.

The solver is currently only available via simulation flags, which can be set in OMEdit under Simulation Setup | Simulation Flags | Additional Simulation Flags (optional). It will be supported via drop-down menus in future releases. See the User's Guide under Solving Modelica Models for further information.

36.45.1 OpenModelica Compiler (OMC)

The new front end has been further improved, with over a dozen bug fixes.

Some issues were fixed in the current backend, although most of the development work is now focused on the new backend, which can be optionally activated with the `--newBackend` compiler flag, and will become the default choice in a future release.

Regarding code generation several issues were fixed, including one that affected the ExternalMedia library, which is now fully supported by OpenModelica.

36.45.2 C Runtime

Besides introducing the new `gbode` solver, some other issues were fixed. The support for external input from CSV files was improved and better documented. See here for a full list of closed issues.

36.45.3 Graphical Editor OMEdit

Besides improved support of the Modelica Standard Library with the package manager, over a dozen issues were fixed, see the full list.

36.45.4 FMI

Main highlights: - The FMI import feature allows to import an FMU as a block in a Modelica model. This feature was broken in versions 1.18.0 and 1.19.0, it is now functional again, with some limitations, see the User's Guide for further information. Support for this feature must be explicitly activated with the Enable FMU Import checkbox in the Tools | Options | Simulation | Translation Flags tab of OMEdit. - Support of FMI code generation using CMake, see the documentation in the User's Guide.

- Bug fixes.

See here for a full list of closed FMI issues.

36.45.5 List of tickets closed in 1.20.0

List of closed tickets in 1.20.0. Over 80 issues were resolved.

36.45.6 Next releases

A 1.20.1 bugfix version may be released if critical bugs are identified and resolved before January 2023. The next release 1.21.0 is planned to be beta-released in February 2023, and released in March 2023; it should include a completely restructured handling of model editing in OMEdit, including long awaited-for features like support of conditional connectors, parameter-depending dialog annotation, and faster rendering of complex diagrams.

Download it from: https://openmodelica.org

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.19.0...v1.20.0

36.46 Release Notes for OpenModelica 1.19.2

Version 1.19.2 is a bugfix release, where some critical bugs that were present in 1.19.0 got fixed.

The most important one is #8772: parameter dependencies were not handled correctly in some cases, leading to some parameters being wrongly evaluated to zero, which also impacted multibody system animations, see #8938.

Some further remaining issues with directory names containing accented letters were fixed in #8777. A problem with the delay operator was also fixed, see #9116.

Here is the complete list of bug fixes in release 1.19.2.

Download it from: https://www.openmodelica.org

36.47 Release Notes for OpenModelica 1.19.0

36.47.1 OpenModelica Compiler (OMC)

The new front end has been further improved, with over 40 bug fixes.

Several issues were fixed in the backend, most notably problems affecting event generation and issues involving algorithms and arrays, which led to a wrong diagnosis of under-determined system of equations. Overall, over 15 issues were addressed.

Regarding code generation 5 issues were fixed, mostly concerning corner cases that involved arrays, record, and slicing operators.

36.47.2 C Runtime

Remaining issues concerning Clock variables were resolved; so, starting from this version, the synchronous features of the Modelica language, in particular the Modelica.Clocked library, are now fully supported. The configuration of runtime and external libraries on Windows was substantially improved. A bug was fixed in daeMode that prevented correct event handling in state-less models, which now run correctly. Overall about 20 issues were fixed.

36.47.3 Graphical Editor OMEdit

The most important new feature in 1.19.0 is the fully integrated package management and conversion script support in the GUI. We suggest you to read the Package Management section of the user's guide for an introduction to the basic concepts.

The package management functionality is now available under the File | Manage Libraries menu. It is possible to install supported (or experimental) open-source libraries from the internet, and upgrade them when newer versions are released; all the required dependencies are installed automatically. The Windows installer by default also installs the basic versions of the Modelica standard library, as a fallback in case internet connection is not available, e.g. in corporate environments. All other libraries are no longer supplied with the installer and must be installed via the package manager. Off-line use of the package manager will be improved in future versions.

When right-clicking on a library in the libraries browser, a new context menu item "Convert to Newer versions of used libraries" is available. This allows you to convert your library to newer versions of its dependencies, once you have installed them. If the newer version of the used library is backwards compatible with the previous one, this just changes the uses annotation, otherwise, the conversion script is run automatically. The most common use of this feature is to upgrade models or libraries that you developed using the Modelica library 3.2.3 to use Modelica 4.0.0. We recommend that you use this functionality on your own libraries only; for libraries that you download from the internet, it is best to wait for the developers to perform the conversion and release new versions of their libraries, that you can then get through the package manager.

Support of the dynamicSelect annotation has been improved, although it is still not 100% functional in all cases.

The plotting of results has been significantly improved; appropriate prefixes are now automatically selected to avoid getting very large or very small numbers on the Y-axis.

Overall, over 40 issues were fixed since the previous 1.18.1 release.

Work is on going to provide full support of parameter-depending conditional connectors, parameter-depending dialog annotations, and editing of parameter modifiers in redeclared classes. Implementing these features requires a fundamental re-design of the way OMEdit interacts with the OMC environment to get the required information. These features should become available, at least experimentally, in the future 1.20.0 release.

36.47.4 FMI Export

- Introduced support for interpolation in CS-FMUs by setting "canInterpolateInputs=true" in modeldescription.xml - Introduced support for getting partial derivatives in ME-FMUs by setting "providesDirectionalDerivative=true" - Lots of improvements to OpenModelica FMU export configurations (black box as well as source code FMUs). - Added source-code nonlinear solver CMINPACK to source-code FMUs
- Basic source-code FMU compilation with CMake; cross compilation with Docker now available on Windows.

About 10 issues fixed.

36.47.5 OMSimulator

- Support of importing and exporting Units and UnitDefinitions to SSP files. - Support for getting directional derivative in OMSimulator using both Symbolic Jacobians and numerical approximation using KINSOL solver.

36.47.6 List of tickets closed in 1.19.0

List of closed tickets in 1.19.0. About 180 issues were resolved.

36.47.7 Next releases

A 1.19.1 bugfix version may be released if critical bugs are identified and resolved before July 2022. The next release 1.20.0 is planned to be beta-released in July 2022, and release in September 2022. On-going work to further improve the level of support of the Buildings library should ensure a significant improvement in the level of support of most Modelica libraries in that upcoming release.

Download it from: https://www.openmodelica.org

36.48 Release Notes for OpenModelica 1.18.0

36.48.1 OpenModelica Compiler (OMC)

The development of the new front end continued further, after becoming the default in the previous version. The compiler can now successfully flatten 100% of the models of the Modelica Standard Library 3.2.3 and 4.0.0 that have an experiment annotation. It can also successfully flatten 100% of many other open-source Modelica libraries, such as Chemical, HelmholtzMedia, IBPSA, ModelicaTest, OpenIPSL, PNLib, PhotoVoltaics, PlanarMechanics, PowerGrids, PowerSysPro, PowerSystems, SystemDynamics, TAeZoSysPro, ThermofluidStream. The success

ratio when flattening the models of the Buildings library is currently at 96%, on-going work aims at reaching 100% by the end of 2021. Overall, 20 front end issues were fixed since the last official release.

Regarding symbolic analysis and code generation, the handling of index reduction in non-trivial cases where the state constraints are spread among many equations has been improved by the introduction of the ASCC algorithm, coupled with the existing reshuffle loop algorithm; this had beneficial effects in particular on three-phase circuit models. Some bugs in the generation of symbolic jacobians for nonlinear systems were also fixed. Overall, 11 issues were resolved.

36.48.2 C Runtime

Full support of the spatialDistribution() operator was introduced, including accurate handling of events. Several low-level issues were resolved, involving memory leaks and segmentation faults.

36.48.3 C++ Runtime

The C++ runtime was improved to handle more Modelica language features that are used by the new frontend. The previously release OMC 1.17 had worse coverage for the C++ runtime since it used the new frontend by default without supporting it in the C++ code generator. With these improvements, the coverage of libraries improved significantly, also compared to previous versions of OpenModelica with the old frontend, since the new frontend handles more models correctly than the old one, see the regression report, column "Compilation".

36.48.4 Graphical Editor OMEdit

Several new features were added to the plotting functionality of OMEdit, in particular: - a "Fit to Diagram" button was added close to the zoom buttons to automatically adjust the extent of a model diagram to its actual content; - the sign of plotted variables can now be toggled by means of the context menu on the variable legends, to compare directly variables that are opposite in sign; - appropriate multiples of SI units (e.g. mW, W, kW, MW, GW for power) are automatically selected for display based on the order of magnitude of the variable, to avoid very large or very small numbers on the plot axes; - the rendering of plots and parameter windows on large and/or high-resolution screens was further improved.

Several bugs were also fixed; in total, 27 issues were addressed.

36.48.5 FMI Support and OMSimulator

FMI 2.0 export was further improved. OMSimulator development continues, over 30 issues were fixed.

36.48.6 Data reconciliation

The operation of power plants requires a good quality of the process data obtained through sensor measurements. Unfortunately, sensor measurements such as flow rates, temperatures, and pressures are subject to errors that lead to uncertainties in the assessment of the system's state. Operational margins are therefore set to take into account uncertainties on the system's state. Their effect is to decrease production because safety regulations put stringent limits on the thermal power of the plants. It is therefore important to compute the best estimates of the measurement uncertainties in order to increase power production by reducing operational margins as much as possible while still preserving system safety.

The best estimates can be obtained by combining data statistics with a knowledge a priori of the system in the form of a physical model. Data reconciliation is a technique that has been conceived in the process industry for that purpose. It is fully described in the VDI 2048 standard for the control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests".

An extension of Modelica to perform data reconciliation on Modelica models was developed together with EDF and is now fully implemented and integrated with the OMEdit GUI. The basic ideas are explained in this presentation, and a section DataReconciliation was added to the User's Guide, explaing how to use it step-by-step.

36.48.7 List of tickets closed in 1.18.0

We are currently transitioning from the old trac issue tracker to the GitHub issue tracker, which is fully integrated with the GitHub revision control and continuous integration framework that we use for the development of OpenModelica. Eventually, the old trac tickets will be migrated onto the GitHub issue tracker. For the time being, old tickets are still managed on trac, while new ones are on GitHub. Overall, about 80 issues were addressed successfully since the 1.17.0 release.

- older tickets
- newer tickets

36.48.8 Next releases

A 1.18.1 bugfix release is planned later this year with some critical bug fixes that have been identified and partially addressed, but need some more testing. The next release 1.19.0 is planned to be released in February 2022, close to the OpenModelica Workshop and Annual Meeting.

Version 1.19.0 is expected to contain a GUI integrated in OMEdit for library handling and conversions. It is also planned to successfully run 100% or close to 100% of the Buildings 7.0.1 library. This will be a major milestone, given the broad extent and complexity of that library.

Download it from: https://www.openmodelica.org

36.49 Release Notes for OpenModelica 1.17.0

36.49.1 OpenModelica Compiler (OMC)

The new frontend has been further improved, and is now the default choice for all the OpenModelica tools. The `-d=newInst` flag no longer needs to be set in scripts. The old frontend is no longer maintained and will progressively be replaced also for the API interface used by OMEdit. 21 tickets were fixed since the previous 1.16.5 release.

Some issues were fixed in the backend, in particular regarding `homotopy` during initialization. Some improvements to code generation, in particular enhancing the support of the HelmholtzMedia library.

The MSYS environment used on Windows was updated to a more recent version, with several benefits: - clang is now also available on Windows and used by default instead of gcc for the compilation of the generated C code, providing much faster C compile time on Windows; - dynamic linking is used instead of static linking, further reducing the overall compilation time on Windows; - a more recent version of the QT library is used, which solves some issues with the rendering of features in OMEdit.

The Sundials solvers and basic linear algebra library KLU were upgraded to the most recent available versions, with benefits in performance and roubustness at runtime.

36.49.2 Supported versions of Modelica Standard Library (MSL)

OpenModelica now supports both currently maintained versions of the Modelica Standard Library, MSL 3.2.3 and MSL 4.0.0. This means that you can use Modelica libraries that use either version of the MSL, or create new ones that do so. Automatic conversion of existing libraries from MSL 3.2.3 to MSL 4.0.0 is currently not yet available, it is planned for version 1.18.0.

Please note that the synchronous features in MSL 4.0.0 are still not fully supported, so we suggest not to rely on them in this version of the tool. Better support is planned for version 1.18.0.

36.49.3 Graphic Editor OMEdit

The user experience with OMEdit is significantly improved in version 1.17.0, in particular for the Windows version, where the compilation time was drastically reduced by using clang instead of gcc and by dynamic linking instead of static linking of the simulation executable.

Thanks to the upgrade of the employed QT graphics libraries, several issues that plagued the graphical user interface are now resolved.

Replaceable classes continue to have graphical support in OMEdit, even though parameters in redeclared classes cannot yet be modified from the GUI, thus requiring to switch to text mode to do so. The new front end can also be used for some of the API functions called by OMEdit to render the model diagrams, making the response of the GUI much faster. Please note that **both these features are currently optional and needs to be activated** by setting *Tools | Options | General | Enable replaceable support* and *Enable new frontend use in the OMC API (faster GUI response)*. These settings are retained from the previous installation upon version upgrading.

Unsaved code was sometimes lost when switching among different windows in OMEdit, this is no longer happening in this release. Several issues that caused occasional crashes of the GUI were also fixed.

OMEdit now can use both currently maintained versions of the Modelica Standard Library, MSL 3.2.3 and MSL 4.0.0. Please note that the Modelica.Clocked package in MSL 4.0.0 is still not handled in a completely reliable way, while most other models work equally well in the two versions.

When starting the program for the first time after installation, one can choose among three options: load MSL 3.2.3 automatically, which however prevents using libraries that need MSL 4.0.0; load MSL 4.0.0 automatically, which prevents using libraries that need MSL 3.2.3; not loading any version of the library upon starting OMEdit, leaving it to the tool to load the right one when a model or library is opened, thanks to the uses() annotation. The latter option allow to handle different projects that use either version of the MSL without problems, of course one at a time. This choice can be later modified by going to the Tools | Options | Libraries menu and by adding or removing the Modelica library from the automatically loaded system libraries, and/or by modifying the specific version that gets loaded.

Proper support of the package manager from the GUI, including conversion scripts to upgrade existing libraries from MSL 3.2.3 to MSL 4.0.0, is planned for version 1.18.0.

16 bug fixes were made to OMEdit in version 1.17.0, to increase usability of the GUI.

36.49.4 Direct support of macOS discontinued from 1.17.0

Until version 1.16.x, OpenModelica was built on Windows, Linux, and macOS. The core functionality of the tool is implemented in Linux, and is ported to Windows using the MinGW environment, and on macOS using the macports environment.

Unfortunately, many libraries OpenModelica depends on are not regularly updated on macports, which caused the Mac build to break every other day. Given our limited resources, we can't take on the burden of the required macports maintenance, so we regret to inform you that we decided to stop providing builds of OpenModelica for macOS. It is still possible to run OpenModelica on the Mac by running a virtual machine with a Linux installation of OpenModelica, see the instructions on the Mac download page. We are still trying to make it possible to build OpenModelica from sources on macOS, please check ticket #6306 for the latest updates and if you want to contribute to the effort. However, we do not guarantee this will be always possible in the future.

36.49.5 FMI Support

FMI 2.0 export and FMI simulation with OMSimulator was further improved.

36.49.6 Other things

OMSimulator is now integrated into pypi and installed via pip.

A prototype Flat Modelica code export feature is now available, a result of the Emphysis project and eFMI on-going standardization effort. It can be activated with the compilation flag.

The Modelica package manager is still only available from the command line in this release. We plan to integrate it the OMEdit GUI for the 1.18.0 release, together with conversion scripts.

Download it from: https://www.openmodelica.org

36.50 Release Notes for OpenModelica 1.16.5

This hopefully the final bugfix release of 1.16.0, which addresses an issue that affected diagrams in OMEdit under certain conditions.

36.50.1 What's Changed

- Keep the NFApi settings (#7215) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/7217

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.16.4...v1.16.5

36.51 Release Notes for OpenModelica 1.16.4

This is a further bugfix release of 1.16.0, which addresses an issue that affected diagrams in OMEdit under certain conditions.

36.51.1 What's Changed

- Move the nfAPI and nfAPINoise settings to general page (#7171) by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/7175
- Prepare for 1.16.4 by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/7186

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.16.3...v1.16.4

36.52 Release Notes for OpenModelica 1.16.2

This is a bug fix release.

36.52.1 What's Changed

- Omedit fixes by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/6982
- Fixes 1.16 by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/7004
- fix ticket #6300, use NF for getElementAnnotations if -d=nfAPI is on by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/7055
- partial fix for #6287 and fix for #6288 by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/7056

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.16.1...v1.16.2

36.53 Release Notes for OpenModelica 1.16.1

This is a further bugfix release of 1.16.0

36.53.1 What's Changed

- fix #6167 allow partial handling in NF when -d=nfAPI is on by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/6898
- Do not put libOMEdit.a in build/ by @sjoelund in https://github.com/OpenModelica/OpenModelica/pull/6908
- Copy OMEdit fixes from master to v1.16 by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/6911
- Use assert instead of Q_ASSERT by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/6929

- Copy #6174 fixes to 1.16 by @adrpo in https://github.com/OpenModelica/OpenModelica/pull/6939
- Backport the new ZMQ flags to 1.16 by @sjoelund in https://github.com/OpenModelica/OpenModelica/pull/6950

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.16.0...v1.16.1

36.54 Release Notes for OpenModelica 1.16.0

36.54.1 OpenModelica Compiler (OMC)

The new frontend has been further improved, with enhanced support of records and arrays. 33 tickets were fixed since the previous 1.14.0 release. The new front-end can now handle 100% of the Modelica Standard Library 3.2.3 models.

Improvements to the back-end:

- new minimal tearing option `--minimalTearing` to disable tearing except for discrete variables
- improved symbolic Jacobian evaluation option `--simJacConstantSplit` reuses constant part of the Jacobian
- improvements to dynamic state selection and common subexpression elimination
- new ASCC algorithm replaces and improves resolveLoops algorithm, allowing to detect high-index problems with indirect structural state constraints, e.g. in three-phase AC circuits
- homotopy operator now used on first try by default
- linearized model export also available in Matlab, Python, and Julia with option `--linearizationDumpLanguage`

Improvements to code generation and runtime:

- Better support for models with records
- CVODE solver from SUNDIALS added;
- 2 methods available: BDF (stiff) and Adams-Moulton (non-stiff)

It is now possible to build a version of OMC with parallel Jacobian evaluation using OpenMP

- On Windows build OMC with `make -f Makefile.omdev.mingw omc -j4 USE_PARJAC=yes`
- On Unix configure with `--enable-parjac` and build OMC.
- Only DASSL and IDA can use parallelized symbolic Jacobians.
- Simulate with `-jacobianThreads=<numberOfThreads>` or set environment variable `OMP_NUM_THREADS=<numberOfThreads>` to use a specific number of threads. Otherwise all available threads will be used.

36.54.2 Graphic Editor OMEdit

Replaceable classes now have graphical support in OMEdit. This feature was planned to be included in release 1.15.0, but it eventually turned out to be more convenient to merge it in the 1.16.0 release. Support for replaceable elements in OMEdit is currently limited to redeclare statements without parameter modifiers. This is enough to handle Modelica.Media medium models in Modelica.Fluid components. Please note that **this feature is currently optional and needs to be activated** by setting ToolsGeneral|Enable replaceable support in OMEdit.

Over 50 bug fixes were made to OMEdit to increase usability of the GUI.

OMSimulator GUI improved: text editing of SSP models, undo/redo, delete components, etc.

36.54.3 FMI Support

Many FMI export bug fixes and improvements to increase usability of OMC-generated FMUs.

FMI/CS 2.0 now includes CVODE solvers (previously only fixed-step forward Euler) and customization via simulation flags.

36.54.4 Other things

Successful integration of Modelon's license manager for encrypted libraries. A special version of OMC available to Consortium members can now handle encrypted libraries, preventing source code browsing, and license management, via Modelon's license manager, embedded in the encrypted library.

Improved Modelica package management. The `conversion(noneFromVersion)` annotation is now fully supported, which means for example that OpenModelica will automatically use the Modelica Standard Library 3.2.3 even for libraries that have a uses annotation pointing to earlier MSL versions, e.g. 3.2.2 or 3.2.1, because the `conversion(noneFromVersion)` annotations in the MSL specify that 3.2.3 can be used instead of 3.2.2 or 3.2.1 without any conversion script, i.e., it is fully backwards compatible. A full-fledged Modelica package manager is also included in OpenModelica 1.16.0, currently only with a command line interface; integration in the GUI is planned for the next 1.17.0 release, together with conversion scripts.

36.54.5 New Contributors

- @mflehmig made their first contribution in https://github.com/OpenModelica/OpenModelica/pull/302
- @stheid made their first contribution in https://github.com/OpenModelica/OpenModelica/pull/752

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.15.0-dev...v1.16.0

36.55 Release Notes for OpenModelica 1.14.2

A bug fix release for v1.14.1, fixing among other things compilation on Ubuntu 20.04.

36.55.1 What's Changed

- Fixes ticket:5733 Don't use the qualified path by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/647
- [NF] Fix some Expression.map*Opt functions. by @perost in https://github.com/OpenModelica/OpenModelica/pull/656
- ticket:5778 Fixed users guide links by @adeas31 in https://github.com/OpenModelica/OpenModelica/pull/666

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.14.1...v1.14.2

36.56 Release Notes for OpenModelica 1.14.0

36.56.1 OpenModelica Compiler (OMC)

The most dramatic enhancement of the OpenModelica Compiler is the New Frontend, which on the average gives a factor of 10-20 speed improvement in the flattening phase of compilation. The new frontend is default in this release, but a feature is implemented that allows the user to switch to the old frontend if problems appear for a specific model The speed of the OMEdit GUI has only slightly increased in this version, since it is still dependent mostly on the old frontend. Further GUI speed increases are available in the coming OpenModelica.1.15.0. About 200 issues have been fixed, including enhancements, compared to the previous 1.13.2 release. The bug fixes are on trac.

OpenModelica Compiler New Frontend news: • Implementation of expandable connectors completed, a rather large piece of work. • A number of smaller enhancements and fixes • The New Frontend (NF)gives slightly better simulation coverage on MSL 3.2.3 than the Old Frontend • The New Frontend is on the average about 20 times faster on flattening. • Remaining work is mainly on further bug fixing and testing the new frontend for all other libraries,

as well as more work on modifiers of arrays in conjunction with non-expanded arrays. (The array modifiers have now been implemented in 1.16.0 but not yet it 1.14.0 in order to not delay the 1.14.0 release)

36.56.2 Graphic Editor OMEdit

- Drag and drop for the text layer. • Auto completion of class names, components and annotations. • GUI for data reconciliation – a method for increasing sensor data precision • Improved duplication functionality for copying classes. • Better handling of Compiler flags. • Partly completed: annotations for dynamic icon update. • Support for connectorSizing annotation • Several bug fixes. You can find the list here. • Docs: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omedit.html. • Autocomplete annotations. • Support for Icon/Diagram map annotation • Copy paste functionality • Reset OMEdit settings/options. • Array plots update on re-simulation • Support for connectorSizing annotation. • Drag and drop class names to text layer in OMEdit • OMPlot: Improved plotting e.g., top and bottom margins for better view, snap to curve etc. • GUI support for replaceable libraries is being tested in a separate branch and will be made available in the coming 1.15.0 release.

36.56.3 OMC backend and run-time system

- A new more efficient and correct implementation of arrays and records.
- The FMI OMSimulator API calls are now also available in the OMC API functions, e.g. for use from OMNotebook, Jupyter notebooks.

Backend new features

- Added possibility to generate analytic Jacobians for linear strong components • Use flag LSanalyticJacobian to enable analytical Jacobian for linear loops. Default false.
- Added output language options for linearization: matlab, python, julia. • Available with --linearizationDumpLanguage=modelica/matlab/python/julia. Default is modelica.

Backend enhancements

- Unified Jacobian evaluation from DASSL and IDA integrators • Added result check for linear solvers Lis, Klu, Total Pivot and Umfpack if a residual function is available. • Improved debug dumping

• -d=bltdump (Index reduction information) • -d=initialization
• -d=dumpLoops

- Improved warning for iteration variables:

• Only warn about non-linear iteration variables with default start attribute. • Other variables have no influence on simulation at all.

- Build instructions for OpenModelica on Windows Subsystem for Linux • Improved Jacobian evaluation with translation flag -d=symJacConstantSplit (requires --generateSymbolicJacobian). Generate Jacobians with separated constant part to split equations that are independent of the seed vector. These equations only need to be evaluated only once per Jacobian evaluation.

Backend bugfixes

- Homotopy: Use simplified version only during initialization to avoid errors during matching and differentiation. • Logging for Homotopy path fixed so log can be loaded in OMEdit. • Support general function call differentiation for equations in residual form. • Equations in residual form don't fail during index reduction any more.

36.56.4 FMI Support

Bug fixes to FMI export, see below

Full Changelog: https://github.com/OpenModelica/OpenModelica/compare/v1.14.0-dev...v1.14.0

36.56.5 Release Notes for OpenModelica 1.13.0

- OMSimulator 2.0 – the second release of our efficient FMI Simulation tool including a GUI for FMI Composition, co-simulation and model-exchange simulation, and SSP standard support. - Model and library encryption/decryption support. (Only for usage by OSMC member organizations) - Improved OpenModelica DAEMode for efficient solution of large Modelica models. - Julia scripting API to OpenModelica. - Basic Matlab scripting API to OpenModelica. - OMSysIdent - parameter estimation module for linear and non-linear parametric dynamic models. - Interactive simulation and control of simulations with OPC-UA. - PDEModelica1 - experimental support for one-dimensional PDEs in Modelica. - Analytic directional derivatives for FMI export and efficient calculation of multiple Jacobian columns – giving much faster simulation for some models - Enhanced OMEdit – including fast multi-file search. - Improved error messages and stability. - A version of the new fast compiler frontend available for testing, can be enabled by a flag Currently (December 10), simulates about 84% of MSL 3.2.2

Note: the replaceable GUI support has been moved to OpenModelica 1.14.0 and will be available in nightly builds.

36.56.6 Release Notes for OpenModelica 1.12.0

- A new (stand-alone) FMI- and TLM-based simulation tool OMSimulator, first version for connected FMUs, TLM objects, Simulink models (via wrappers), Adams models (via wrappers), BEAST models (via wrappers), Modelica models
- Graphic configuration editing of composite models consisting of FMUs
- Basic graphical editing support for state machines and transitions
- Faster lookup processing, making some libraries faster to browse and compile
- Additional advanced visualization features for multibody animation
- Increased library coverage including significantly increased verification coverage
- Increased tool interoperability by addition of the ZeroMQ communications protocol
- Further enhanced OMPython including linearization, now also working with Python 3
- Support for RedHat/Fedora binary builds of OpenModelica

OpenModelica Compiler (OMC)

- Faster lookup processing
- Initializing external objects together with parameters
- Handle exceptions in numeric solvers
- Support for higher-index discrete clock partitions
- Improved unit checking
- Improved initialization of start values
- Decreased compilation time of models with large size arrays
- New approach for homotopy-based initialization (still experimental)
- A bunch of fixes: Bugs, regressions, performance issues
- Improved Dynamic Tearing by adding constraints for the casual set
- Improved module wrapFunctionCalls with one-time evaluation of Constant CSE-variables
- Added initOptModule for inlineHomotopy

- Added configuration flag tearingStrictness to influence solvability
- New methods for inline integration for continuous equations in clocked partitions, now covering: ExplicitEuler, ImplicitEuler, SemiImplicitEuler and ImplicitTrapezoid
- Complete implementation of synchronous features in C++ runtime
- Refactored linear solver of C++ runtime
- Improved Modelica_synchronous_cpp coverage
- New common linear solver module, optionally sparse, for the C++ runtime
- Coverage of most of the OpenHydraulics library
- Improved coverage of ThermoSysPro, IdealizedContact and Chemical libraries
- Support of time events for cpp-simulation and enabled time events in cpp-FMUs
- Global homotopy method for initialization
- Scripting API to compute accumulated errors (1-norm, 2-norm, max. error) of 2 time series

Graphic Editor OMEdit

- Additional advanced visualization features for multibody animation (transparency, textures, change colours by dialog)
- An HTML WYSIWYG Editor, e.g. useful for documentation
- Support for choices(checkBox=true) annotation.
- Support for loadSelector & saveSelector attribute of Dialog annotation.
- Panning of icon/diagram view and plot window.
- AutoComplete feature in text editing for keywords, types, common Modelica constructs
- Follow connector transformation from Diagram View to Icon View.
- Further stability improvements
- Improved performance for rendering some icons using the interactive API
- Improved handling of parameters that cannot be evaluated in Icon annotations
- Basic graphic editing support for state machines and transitions (not yet support for showing state internals on diagram layer)
- Interactive state manipulation for FMU-based animations

FMI Support

- A new (stand-alone) FMI- and TLM-based simulation tool OMSimulator, first version (a main deliverable of the OPENCPS project, significant contributions and code donations from SKF)
- Graphic configuration editing of composite models consisting of FMUs
- Co-simulation/simulation of connected FMUs, TLM objects, Simulink models (via wrappers), Adams models (via wrappers), BEAST models (via wrappers), Modelica models.

Other things

- Increased OpenModelica tool interoperability by adding the ZeroMQ communications protocol in addition to the previously available Corba. This also enables Python 3 usage in OMPython on all platforms.
- Textual support through the OpenModelica API and graphical support in OMEdit for generation of single or multiple requirement verification scenarios
- VVDRlib – a small library for connecting requirements and models together, with notions for mediators, scenarios, design alternatives

- Further enhanced OMPython including linearization, now also working with Python 3.¨
- Jupyter notebooks also supported with OMPython and Python 3
- New enhanced library testing script (libraries.openmodelica.org/branches).
- Addition of mutable reference data types in MetaModelica
- Support for RedHat/Fedora binary builds of OpenModelica
- Support for exporting the system of equations in GraphML (yEd) format for debugging

36.56.7 Release Notes for OpenModelica 1.11.0

- Dramatically improved compilation speed and performance, in particular for large models.
- 3D animation visualization of regular MSL MultiBody simulations and for real-time FMUs.
- Better support for synchronous and state machine language elements, now supports 90% of the clocked synchronous library.
- Several OMEdit improvements including folding of large annotations.
- 64-bit OM on Windows further stabilized
- An updated OMDev (OpenModelica Development Environment), involving msys2. This was needed for the shift to 64-bit on Windows.
- Integration of Sundials/IDA DAE solver with potentially large increase of simulation performance for large models with sparse structure.
- Improved library coverage.
- Parameter sensitivity analysis added to OMC.

OpenModelica Compiler (OMC)

- Real-time synchronization support by using simFlag -rt=1.0 (or some other time scaling factor).
- A prototype implementation of OPC UA using an open source OPC UA implementation. The old OPC implementation was not maintained and relied on a Windows-only proprietary OPC DA+UA package. (At the moment, OPC is experimental and lacks documentation; it only handles reading/writing Real/Boolean input/state variables. It is planned for OMEdit to use OPC UA to re-implement interactive simulations and plotting.)
- Dramatically improved compilation speed and dramatically reduced memory requirements for very large models. In Nov 2015, the largest power generation and transmission system model that OMC could handle had 60000 equations and it took 700 seconds to generate the simulation executable code; it now takes only 45 seconds to do so with OMC 1.11.0, which can also handle a model 10 times bigger (600 000 equations) in less than 15 minutes and with less than 32 GB of RAM. Simulation times are comparable to domain-specific simulation tools. See for example ScalableTestSuite for some of the improvements.
- Improved library coverage
- Better support for synchronous and state machine language elements, now simulates 90% of the clocked synchronous library.
- Enhanced Cpp runtime to support the PowerSystems library.
- Integration of Sundials/IDA solver as an alternative to DASSL.
- A DAEMode solver mode was added, which allows to use the sparse IDA solver to handle the DAEs directly. This can lead to substantially faster simulation on large systems with sparse structure, compared to the traditional approach.
- The direct sparse solvers KLU and SuperLU have been added, with benefits for models with large algebraic loops.
- Multi-parameter sensitivity analysis added to OMC.

- Progress on more efficient inline function mechanism.
- Stabilized 64-bit Windows support.
- Performance improvement of parameter evaluation.
- Enhanced tearing support, with prefer iteration variables and user-defined tearing.
- Support for external object aliases in connectors and equations (a non-standard Modelica extension).
- Code generation directly to file (saves maximum memory used). #3356
- Code generation in parallel is enabled since #3356 (controlled by omc flag `-n`). This improves performance since generating code directly to file avoid memory allocation.
- Allowing mixed dense and sparse linear solvers in the generated simulation (chosen depending on simflags `-ls` (dense solver), `-lss` (sparse solver), `-lssMaxDensity` and `-lssMinSize`).

Graphic Editor OMEdit

- Significantly faster browsing of most libraries.
- Several GUI improvements including folding of multi-line annotations.
- Further improved code formatting preservation during edits.
- Support for all simulation logging flags.
- Select and export variables after simulation.
- Support for Byte Order Mark. Added support enables other tools to correctly read the files written by OMEdit.
- Save files with line endings according to OS (Windows (CRLF), Unix (LF)).
- Added OMEdit support for FMU cross compilation. This makes it possible to launch OMEdit on a remote or virtual Linux machine using a Windows X server and export an FMU with Windows binaries.
- Support of DisplayUnit and unit conversion.
- Fixed automatic save.
- Initial support for DynamicSelect in model diagrams (texts and visible attribute after simulation, no expressions yet).
- An HTML documentation editor (not WYSIWYG; that editor will be available in the subsequent release).
- Improved logging in OMEdit of structured messages and standard output streams for simulations.

FMI Support

- Cross compilation of C++ FMU export. Compared to the C runtime, the C++ cross compilation covers the whole runtime for model exchange.
- Improved Newton solver for C++ FMUs (scaling and step size control).

Other things

- 3D animation visualization of regular MSL MultiBody simulations and for real-time FMUs.
- An updated OMDev (OpenModelica Development Environment), involving msys2. This was needed for the shift to 64-bit on Windows.
- OMWebbook, a web version of OMNotebook online. Also, a script is available to convert an OMNotebook to an OMWebbook.
- A Jupyter notebook Modelica mode, available in OpenModelica.

1.11.0,status=closed,severity!=trivial,resolution=fixed|-,col=changelog,group=component,format=table)

36.56.8 Release Notes for OpenModelica 1.10.0

The most important enhancements in the OpenModelica 1.10.0 release:

OpenModelica Compiler (OMC)

New features:

- Real-time synchronization support by using simFlag -rt=1.0 (or some other time scaling factor). - A prototype implementation of OPC UA using an open source OPC UA implementation. The old OPC implementation was not maintained and relied on a Windows-only proprietary OPC DA+UA package. (At the moment, OPC is experimental and lacks documentation; it only handles reading/writing Real/Boolean input/state variables. It is planned for OMEdit to use OPC UA to re-implement interactive simulations and plotting.)

Performance enhancements:

- Code generation directly to file (saves maximum memory used). #3356 - Code generation in parallel enabled since #3356 allows this without allocating too much memory (controlled by omc flag `-n`). - Various scalability enhancements, allowing the compiler to handle hundreds of thousands of equations. See for example ScalableTestSuite for some of the improvements. - Better defaults for handling tearing (OMC flags `--maxSizeLinearTearing` and `--maxSizeNonlinearTearing`).
- Allowing mixed dense and sparse linear solvers in the generated simulation (chosen depending on simflags `-ls` (dense solver), `-lss` (sparse solver), `-lssMaxDensity` and `-lssMinSize`).

Graphic Editor OMEdit

OpenModelica Notebook (OMNotebook)

Optimization

FMI Support

OpenModelica Development Environment (OMDev)

36.56.9 Release Notes for OpenModelica 1.9.4

OpenModelica v1.9.4 was released 2016-03-09. These notes cover the v1.9.4 release and its subsequent bug-fix releases (now up to 1.9.7).

OpenModelica Compiler (OMC)

- Improved simulation speed for many models. simulation speed went up for 80% of the models. The compiler frontend became faster for almost all models, average about 40% faster.
- Initial support for synchronous models with clocked equations as defined in the Modelica 3.3 standard
- Support for homotopy operator

Graphic Editor OMEdit

- Undo/Redo support.
- Preserving text formatting, including indentation and whitespace. This is especially important for diff/merge with several collaborating developers possibly using several different Modelica tools.
- Better support for inherited classes.
- Allow simulating models using visual studio compiler.
- Support for saving Modelica package in a folder structure.
- Allow reordering of classes inside a package.
- Highlight matching parentheses in text view.
- When copying the text retain the text highlighting and formatting.

- Support for global head definition in the documentation by using `__OpenModelica_infoHeader` annotation.
- Support for expandable connectors.
- Support for uses annotation.

FMI Support

- Full FMI 2.0 co-simulation support now available
- Upgrade Cpp runtime from C++03 to C++11 standard, minimizing external link dependencies. Exported FMUs don't depend on additional libraries such as boost anymore
- FMI 2.0 is broken for some models in 1.9.4. Upgrading to 1.9.6 is advised.

36.56.10 Release Notes for OpenModelica 1.9.3

The most important enhancements in the OpenModelica 1.9.3 release:

- Enhanced collaborative development and testing of OpenModelica by moving to the GIT-hub framework for versioning and parallel development.
- More accessible and up-to-date automatically generated documentation provided in both html and pdf.
- Further improved simulation speed and coverage of several libraries.
- OMEdit graphic connection editor improvements.
- OMNotebook improvements.

OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- Further improved simulation speed and coverage for several libraries.
- Faster generated code for functions involving arrays, factor 2 speedup for many power generation models.
- Better initialization.
- An implicit inline Euler solver available.
- Code generation to enable vectorization of for-loops.
- Improved non-linear, linear and mixed system solving.
- Cross-compilation for the ARMhf architecture.
- A prototype state machine implementation.
- Improved performance and stability of the C++ runtime option.
- More accessible and up-to-date automatically generated documentation provided in both html and .pdf.

Graphic Editor OMEdit

There are several improvements to the OpenModelica graphic connection editor OMEdit:

- Support for uses annotations.
- Support for declaring components as vectors.
- Faster messages browser with clickable error messages.
- Support for managing the stacking order of graphical shapes.
- Several improvements to the plot tool and text editor in OMEdit.

OpenModelica Notebook (OMNotebook)

Several improvements:

- Support for moving cells from one place to another in a notebook.
- A button for evaluation of whole notebooks.
- A new cell type called Latex cells, supporting Latex formatted input that provides mathematical typesetting of formulae when evaluated.

Optimization

Several improvements of the Dynamic Optimization module with collocation, using Ipopt:

- Better performance due to smart treatment of algebraic loops for optimization.
- Improved formulation of optimization problems with an annotation approach which also allows graphical problem formulation.
- Proper handling of constraints at final time.

FMI Support

Further improved FMI 2.0 co-simulation support.

OpenModelica Development Environment (OMDev)

A big change: version handling and parallel development has been improved by moving from SVN to GIThub. This makes it easier for each developer to test his/her fixes and enhancements before committing the code. Automatic mirroring of all code is still performed to the OpenModelica SVN site.

36.56.11 Release Notes for OpenModelica 1.9.2

The OpenModelica 1.9.2 Beta release is available now, January 31, 2015. Please try it and give feedback! The final release is planned within 1-2 weeks after some more testing. The most important enhancements in the OpenModelica 1.9.2 release:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.
- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Further improved simulation coverage for a number of libraries.
- OMEdit graphic connection editor improvements

OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.
- Further improved simulation coverage for a number of libraries compared to 1.9.1. For example:
  - MSL 3.2.1 100% compilation, 97% simulation (3% increase)
  - MSL Trunk 99% compilation (1% increase), 93% simulation (3% increase)
  - ModelicaTest 3.2.1 99% compilation (2% increase), 95% simulation (6% increase)
  - ThermoSysPro 100% compilation, 80% simulation (17% increase)

  - ThermoPower 97% compilation (5% increase), 85% simulation (5% increase)
  - Buildings 80% compilation (1% increase), 73% simulation (1% increase)
- Further enhanced OMC compiler front-end coverage, scalability, speed and memory.
- Better initialization.
- Improved tearing.
- Improved non-linear, linear and mixed system solving.
- Common subexpression elimination support - drastically increases performance of some models.

Graphic Editor OMEdit

- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Enhanced simulation setup window in OMEdit, which among other things include better support for integration methods and dassl options.
- Support for running multiple simultaneous simulation.
- Improved handling of modifiers.
- Re-simulate with changed options, including history support and re-simulating with previous options possibly edited.
- More user friendly user interface by improved connection line drawing, added snap to grid for icons and conversion of icons from PNG to SVG, and some additional fixes.

Optimization

Some smaller improvements of the Dynamic Optimization module with collocation, using Ipopt.

FMI Support

Further improved for FMI 2.0 model exchange import and export, now compliant according to the FMI compliance tests. FMI 1.0 support has been further improved.

36.56.12 Release Notes for OpenModelica 1.9.1

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.

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 (#2724, reject non-constructor functions returning external objects)
- Improved C++ runtime.

- Improved testing facilities.
- New unit checking implementation.
- Support for model rewriting expressions via rewriting rules in an external file.
- Reject more bad code (r19986, consider records with different components type-incompatible)

OpenModelica Connection 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.

OMPython

The interface was changed to version 2.0, which uses one object for each OpenModelica instance you want active. It also features a new and improved parser that returns easier to use datatypes like maps and lists.

Optimization

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

FMI Support

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

36.56.13 Release Notes for OpenModelica 1.9.0

This is the summary description of changes to OpenModelica from 1.8.1 to 1.9.0, released 2013-10-09. This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

OpenModelica Compiler (OMC)

This release mainly includes bug fixes and 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 245 (89%) simulate, compared to 30% simulating in the 1.9.0 beta1 release.
- Much better simulation for the ModelicaTest 3.2.1 library, now 401 out of 428 models build (93%) and 364 simulate (85%), compared to 32% in November 2012.
- Better simulation support for several other libraries, e.g. more than twenty examples simulate from ThermoSysPro, and all but one model from PlanarMechanics simulate.
- 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 (#1764).
- It is now possible to translate the error messages in the omc kernel (#1767).
- FMI Support. FMI co-simulation with OpenModelica as master. Improved FMI Import and export for model exchange. Most of FMI 2.0 is now also supported.
- Checking (when possible) that variables have been assigned to before they are used in algorithmic code (#1776).
- 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 (Windows only).
- 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 (r12657,r12658).
- The compiler is now fully bootstrapped and can compile itself using a modest amount of heap and stack space (less than the RML-based compiler, which is still the default).
- Some old debug-flags were removed. Others were renamed. Debug flags can now be enabled by default.
- Removed old unused simulation flags noClean and storeInTemp (r15927).
- Many stack overflow issues were resolved.
- Dynamic Optimization with OpenModelica. 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.

OpenModelica Notebook (OMNotebook)

- A `shortOutput` option has been introduced in the simulate command for less verbose output. The DrModelica interactive document has been updated and the models tested. Almost all models now simulate with OpenModelica.

OpenModelica Eclipse Plug-in (MDT)

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

OpenModelica Development Environment (OMDev)

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

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.

ModelicaML

- 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.

36.56.14 Release Notes for OpenModelica 1.8.1

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. The release was made on 2012-04-03 (r11645).

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.

OpenModelica Notebook (OMNotebook)

- Faster and more stable plottning.

OpenModelica Shell (OMShell)

- No changes.

OpenModelica Eclipse Plug-in (MDT)

- Small fixes and improvements.

OpenModelica Development Environment (OMDev)

- No changes.

Graphic Editor OMEdit

- Bug fixes.

OMOptim Optimization Subsystem

- Bug fixes.

FMI Support

- Bug fixes.

36.56.15 OpenModelica 1.8.0, November 2011

The OpenModelica 1.8.0 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.

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 1.8.0 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.

OpenModelica Notebook (OMNotebook)

No changes.

OpenModelica Shell (OMShell)

Small performance improvements.

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.

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.

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.

New OMOptim Optimization Subsystem

Bug fixes, especially in the Linux version.

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.

36.56.16 OpenModelica 1.7.0, April 2011

The OpenModelica 1.7.0 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 OMOptim optimization subsystem, and a new performance profiler for equation-based Modelica models.

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.

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.

OpenModelica Shell (OMShell)

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

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.

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.

New Performance Profiler

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

36.56.17 OpenModelica 1.6.0, November 2010

The OpenModelica 1.6.0 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.

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.

OpenModelica Notebook (OMNotebook)

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

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.

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.

36.56.18 OpenModelica 1.5.0, 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.

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.

OpenModelica Notebook (OMNotebook)

Improvements in platform availability.

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

36.56.19 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.

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.

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).

36.56.20 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.

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.

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.

OpenModelica Eclipse Plug-in (MDT)

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

OpenModelica Development Environment (OMDev)

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

36.56.21 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.

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.

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.

OpenModelica Shell (OMShell)

Now available also on the Macintosh platform.

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.

OpenModelica Development Environment (OMDev)

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

36.56.22 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.

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.

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.

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.

36.56.23 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.

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.

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).

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.

36.56.24 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..

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.

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.

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.

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.

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.

36.56.25 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.

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.

OpenModelica Shell (OMShell)

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

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.

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.

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.

CHAPTER

THIRTYSEVEN

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.

37.1 OpenModelica Contributors 2015

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Volker Waurich, TU Dresden, Dresden, Germany.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Anders Andersson, VTI, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Bouskela, EDF, Paris, France.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Atiyah Elsheikh, AIT, Vinnea, Austria.

Rüdiger Franke, ABB, Germany

Jens Frenkel, TU Dresden, Dresden, Germany.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Henning Kiel, Bocholt, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Leonardo Laguna, Wolfram MathCore AB, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Alachew Mengist, PELAB, Linköping University, Linköping, Sweden.

Abhir Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Lars Mikelsons, Bosch Rexroth, Lohr am Main, Germany.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Kannan Moudgalya, IIT Bombay, Mumbai, India.

Kenneth Nealy, USA.

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.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Vitalij Ruge, Fachhochschule Bielefeld, Bielefeld, Germany.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Roland Samlaus, Bosch, Stuttgart, Germany.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Jan Šilar, Charles University, Prague, Czech Republic

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Bernhard Thiele, PELAB, Linköping University, Linköping, Sweden

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Gustaf Thorslund, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Marcus Walther, TU Dresden, Dresden, Germany

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

37.2 OpenModelica Contributors 2014

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, 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.

37.3 OpenModelica Contributors 2013

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

37.4 OpenModelica Contributors 2012

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Mikael Axin, IEI, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

37.5 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.

37.6 OpenModelica Contributors 2010

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, 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.

37.7 OpenModelica Contributors 2009

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy

Filippo Donida, Politecnico di Milano, Milan, Italy

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Jens Frenkel, TU Dresden, Dresden, Germany.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden

Magnus Leksell, Linköping, Sweden

Oliver Lenord, Bosch-Rexroth, Lohr am Main, Germany

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Hannu Niemistö, VTT, Espoo, Finland

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Atanas Pavlov, Munich, Germany.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany

Robert Wotzlaw, Goettingen, Germany

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden

37.8 OpenModelica Contributors 2008

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.
David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.
Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.
Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.
Mikael Blom, PELAB, Linköping University, Linköping, Sweden.
David Broman, PELAB, Linköping University, Linköping, Sweden.
Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.
Anders Fernström, PELAB, Linköping University, Linköping, Sweden.
Pavel Grozman, Equa AB, Stockholm, Sweden.
Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.
Kim Jansson, PELAB, Linköping University, Linköping, Sweden.
Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.
Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.
Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.
Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.
Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.
Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.
Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.
Simon Bjorklén, PELAB, Linköping University, Linköping, Sweden.
Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

37.9 OpenModelica Contributors 2007

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

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Ola Leifler, IDA, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

William Spinelli, Politecnico di Milano, Milano, Italy

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Stefan Vorkoetter, MapleSoft, Waterloo, Canada.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

37.10 OpenModelica Contributors 2006

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

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Elmir Jagudin, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Andreas Remar, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

37.11 OpenModelica Contributors 2005

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

Peter Aronsson, PELAB, Linköping University and MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Ingemar Axelsson, PELAB, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

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

37.12 OpenModelica Contributors 2004

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

Peter Aronsson, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Peter Bunus, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Emma Larsdotter Nilsson, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

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

37.13 OpenModelica Contributors 2003

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

Peter Aronsson, Linköping University, Linköping, Sweden.

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

Peter Bunus, PELAB, Linköping University, Linköping, Sweden.

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

Daniel Hedberg, Linköping University, Linköping, Sweden.

Eva-Lena Lengquist-Sandelin, PELAB, Linköping University, Linköping, Sweden.

Susanna Monemar, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Erik Svensson, MathCore Engineering AB, Linköping, Sweden.

37.14 OpenModelica Contributors 2002

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

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

Peter Aronsson, Linköping University, Linköping, Sweden.

Daniel Hedberg, Linköping University, Linköping, Sweden.

Henrik Johansson, PELAB, Linköping University, Linköping, Sweden

Andreas Karström, PELAB, Linköping University, Linköping, Sweden

37.15 OpenModelica Contributors 2001

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

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

Peter Aronsson, Linköping University, Linköping, Sweden.

37.16 OpenModelica Contributors 2000

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

37.17 OpenModelica Contributors 1999

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden

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

37.18 OpenModelica Contributors 1998

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

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

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

BIBLIOGRAPHY

[FPA+20] Peter Fritzson, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzson, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, John Tinnerholm, and Per Östlund. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295, 2020. doi:10.4173/mic.2020.4.1.

[ABB+99] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and others. *LAPACK Users' guide*. SIAM, 1999.

[BBOR15] Bernhard Bachmann, W Braun, L Ochel, and V Ruge. Symbolical and numerical approaches for solving nonlinear systems. In *Annual OpenModelica Workshop*, volume 2015. 2015.

[CK06] Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387261028.

[DN10] T. A. Davis and E. Palamadai Natarajan. Algorithm 907: klu, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3):36:1–36:17, September 2010. URL: http://doi.acm.org/10.1145/1824801.1824814, doi:10.1145/1824801.1824814.

[Dav04] Timothy A Davis. Algorithm 832: umfpack v4. 3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004.

[DJS96] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[HNorsettW93] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag Berlin Heidelberg, 2nd rev. ed. 1993. corr. 3rd printing 2008 edition, 1993. ISBN 978-3-540-56670-0. doi:10.1007/978-3-540-78862-1.

[HNorsettW00] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I Nonstiff problems*. Springer, Berlin, second edition, 2000.

[HBG+05] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[Kel78] Herbert B Keller. Global homotopies and newton methods. In *Recent advances in numerical analysis*, pages 73–94. Elsevier, 1978.

[KC19] Christopher A. Kennedy and Mark H. Carpenter. Diagonally implicit runge–kutta methods for stiff odes. *Applied Numerical Mathematics*, 146:221–244, 2019. URL: https://www.sciencedirect.com/science/article/pii/S0168927419301801, doi:https://doi.org/10.1016/j.apnum.2019.07.008.

[Nat05] Ekanathan Palamadai Natarajan. *KLU–A high performance sparse linear solver for circuit simulation problems*. PhD thesis, University of Florida, 2005.

[Nis10] Akira Nishida. Experience in developing an open source scalable software infrastructure in japan. In *International Conference on Computational Science and Its Applications*, 448–462. Springer, 2010.

[OB13] Lennart A Ochel and Bernhard Bachmann. Initialization of equation-based hybrid models within openmodelica. In *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools; April 19; University of Nottingham; Nottingham; UK*, number 084, 97–103. Linköping University Electronic Press, 2013.

[Pet82] L.R. Petzold. Description of dassl: a differential/algebraic system solver. 1982.

[Sie12] Michael Sielemann. *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. PhD thesis, TU Hamburg-Harburg, Germany, 2012. doi:10.15480/882.1111.

[SCO+11] Michael Sielemann, Francesco Casella, Martin Otter, Christop Clauß, Jonas Eborn, Sven Erik Matsson, and Hans Olsson. Robust initialization of differential-algebraic equations using homotopy. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, number 063, 75–85. Linköping University Electronic Press, 2011.

[T+98] Allan G Taylor and others. User documentation for kinsol, a nonlinear solver for sequential and parallel computers. Technical Report, Lawrence Livermore National Lab., CA (United States), 1998.

[Axe05] Ingemar Axelsson. OpenModelica Notebook for interactive structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, October 2005. LITH-IDA-EX–05/080–SE.

[Fernstrom06] Anders Fernström. Extending OpenModelica Notebook – an interactive notebook for structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, September 2006. LITH-IDA-EX–06/057—SE.

[Fri04] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, February 2004. ISBN 0-471-471631.

[Knu84] Donald E. Knuth. Literate programming. *The Computer Journal*, 27:97–111, 1984.

[Wol96] Stephen Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge University Press, third edition, 1996.

[BOR+12] Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson. Parallel multiple-shooting and collocation Optimization with OpenModelica. In Martin Otter and Dirk Zimmer, editors, *Proceedings of the 9th International Modelica Conference*. Linköping University Electronic Press, September 2012. doi:10.3384/ecp12076659.

[RBB+14] Vitalij Ruge, Willi Braun, Bernhard Bachmann, Andrea Walther, and Kshitij Kulshreshtha. Efficient implementation of collocation methods for optimization using openmodelica and adol-c. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*. Modelica Association and Linköping University Electronic Press, March 2014. doi:10.3384/ecp140961017.

INDEX

O

OMEdit, 201
OMSimulator, 135
    Examples, 138
    Flags, 135
OMSimulatorLib, 138
    C-API, 138
OMSimulatorLua, 154
    Examples, 154
    Scripting Commands, 155
OMSimulatorPython, 170
    Examples, 171
    Scripting Commands, 172
OpenModelicaScripting, 189
    Examples, 189
    Scripting Commands, 190

S

SSP, 212
    Bus connections, 213