

# DAE Tools Project Documentation Release 1.3.1

by Dragan Nikolic

# **CONTENTS**

1	Introduction		1
	1.1 What is DAE Tools?	 	. 1
	1.2 Licence	 	. 2
	1.3 History	 	. 2
	1.4 Acknowledgements	 	. 3
	1.5 How to cite	 	. 3
2	Programming paradigms		5
	2.1 The Hybrid approach	 	. 5
	2.2 The Equation-Oriented approach	 	. 6
	2.3 The Object-Oriented approach	 	. 7
	2.4 Programming language	 	. 8
3	Architecture		9
	3.1 pyCore module		
	3.2 pyActivity module		
	3.3 pyDataReporting module		
	3.4 pyIDAS module		
	3.5 pyUnits module		
	3.6 NLP/MINLP modules		
	3.7 LA solver modules	 	. 12
4	Cotting DAE Tools		13
4	Getting DAE Tools 4.1 System requirements		_
	4.2 Getting the packages		
	4.3 Installation		
	4.3.1 GNU/Linux		
	4.3.2 MacOS		
	4.3.3 Windows		
	4.4 Compiling from source		
	4.4.1 GNU/Linux and MacOS		
	4.4.2 Windows	 	. 17
5	Getting Started with DAE Tools		19
	5.1 Running tutorials		
	5.2 Models		
	5.2.1 Developing a model		
	5.3 Simulation		
	5.3.1 Setting up a simulation		
	5.3.2 Running a simulation		
	5.4 Optimization		
	5.4.1 Setting up an optimization		
	5.5 Processing the results	 	. 29
6	pyDAE User Guide		31
	6.1 The main concepts	 	. 31
	-		

7	pyDA	E API Reference	33
	7.1	Module pyCore	. 33
		7.1.1 Overview	. 33
		7.1.2 Key modelling concepts	
		7.1.3 Autodifferentiation and equation evaluation tree support	
		7.1.5 Auxiliary functions	
		7.1.6 Enumerations	
		7.1.7 Global constants	. 50
	7.2	Message logs	. 51
		7.2.1 Overview	. 51
	7.3	Module pyActivity	
	,	7.3.1 Overview	
		7.3.3 Enumerations	
	7.4	Module pyDataReporting	
		7.4.1 Overview	. 56
		7.4.2 DataReporter classes	. 56
		7.4.3 DataReceiver classes	. 59
	7.5	Module pyIDAS	. 60
		7.5.1 Overview	
		7.5.2 Classes	
		7.5.3 Enumerations	
	7.6	Module pyUnits	
		7.6.1 Overview	
		7.6.2 Classes	
	7.7	Variable types	. 63
		7.7.1 Overview	. 63
	7.8	Third party solvers	
		7.8.1 Linear solvers	
		7.8.2 Optimization solvers	
		7.8.3 Parameter estimation solvers	
	7.0		
	7.9	Code generators	
		7.9.1 Auxiliary classes	
		7.9.2 Modelica	
		7.9.3 ANSI C	
		7.9.4 Functional Mockup Interface (FMI)	
	7.10	DAE Tools Plotter	. 68
		7.10.1 Overview	
		7.10.2 Classes	. 68
	7.11	DAE Tools Simulator	. 69
	,,,,	7.11.1 Overview	
		7.11.2 Classes	
		7.11.2 Classes	. 09
8	Tutor	ials	71
	8.1	What's the time? (AKA: Hello world!)	
	8.2	Tutorial 1	
	8.3	Tutorial 2	
	8.4	Tutorial 3	
	8.5	Tutorial 4	
	8.6	Tutorial 5	. 74
	8.7	Tutorial 6	. 75
	8.8	Tutorial 7	. 75
	8.9	Tutorial 8	. 75
	8.10	Tutorial 9	. 76
	8.11	Tutorial 10	
	8.12	Tutorial 11	
	8.13	Tutorial 12	
	8.14		
		Tutorial 13	
	8.15	Tutorial 14	
	8.16	Tutorial 15	
	8.17	Tutorial 16	. 79

In	dex		8
Py	thon N	Module Index	85
9	Indic	es and tables	83
	8.24	Optimization tutorial 6	8
		Optimization tutorial 5	
	8.22	Optimization tutorial 4	80
	8.21	Optimization tutorial 3	80
	8.20	Optimization tutorial 2	79
		Optimization tutorial 1	
	8.18	Tutorial 17	79

# INTRODUCTION

#### 1.1 What is DAE Tools?

**DAE Tools** is a free/open-source cross-platform object- and equation-oriented process modelling and optimization software. It is not a modelling language, rather a collection of software tools for:

- · Modelling development
- Simulation, optimization, and parameter estimation
- Processing of the results (plotting and exporting to various file formats)
- · Report generation
- · Code generation and model exchange

**DAE Tools** is initially developed to model and simulate processes in chemical process industry (mass, heat and momentum transfers, chemical reactions, separation processes, thermodynamics). However, **DAE Tools** can be used to develop high-accuracy models of (in general) many different kind of processes/phenomena, simulate/optimize them, visualize and analyse the results.

The following approaches/paradigms are adopted in **DAE Tools**:

- A hybrid approach between general-purpose programming languages (such as c++ and Python) and domain-specific modelling languages (such as Modelica, gPROMS, Ascend etc.) (more information: *The Hybrid approach*)
- An object-oriented approach to process modelling (more information: *The Object-Oriented approach*)
- An Equation-Oriented (acausal) approach where all model variables and equations are generated and gathered together and solved simultaneously using a suitable mathematical algorithm (more information: *The Equation-Oriented approach*)
- Separation of the model definition from the activities that can be carried out on that model. The structure of the model (parameters, variables, equations etc.) is given in the model class while the runtime information in the simulation class. This way we can have a single model definition, but one or more different simulation scenarios, and one or more optimization scenarios
- Core libraries are written in standard c++, however Python is used as the main modelling language (more information: *Programming language*)

Class of problems that can be solved by **DAE Tools**:

- Initial value problems of implicit form, described by a system of linear, non-linear, and (partial-)differential algebraic equations
- Index-1 DAE systems
- With lumped or distributed parameters: Finite Difference or Finite Elements Methods (still experimental)
- Steady-state or dynamic
- Continuous with some elements of event-driven systems (discontinuous equations, state transition networks and discrete events)

Type of activities that can be carried out with models developed in **DAE Tools**:

Simulation

- Steady-state or dynamic
- With simple or complex operating procedures
- Optimization
  - NLP problems
  - MINLP problems
- Parameter estimation
  - The least squares method (Levenberg–Marquardt algorithm)
- XML + MathML report generation
- Code generation for other modelling languages and general purpose programming languages
  - Modelica
  - ANSI C
  - Functional Mockup Interface (FMI)
- Export of the simulation results to various file formats

All core libraries are written in standard ANSI/ISO c++. It is highly portable - it runs on all major operating systems (GNU/Linux, MacOS, Windows) and all platforms with a decent c++ compiler, Boost and standard c/c++ libraries (by now it is tested on 32/64 bit x86 and ARM architectures making it suitable for use in embedded systems). Models can be developed in Python (**pyDAE** module) or c++ (**cDAE** module), compiled into an independent executable and deployed without a need for any run time libraries.

**DAE Tools** support a large number of solvers. Currently Sundials IDAS solver is used to solve DAE systems and calculate sensitivities, while BONMIN, IPOPT, and NLOPT solvers are used to solve NLP/MINLP problems. **DAE Tools** support direct dense and sparse matrix linear solvers (sequential and multi-threaded versions) at the moment. In addition to the built-in Sundials linear solvers, several third party libraries are interfaced: SuperLU/SuperLU\_MT, Intel Pardiso, AMD ACML, Trilinos Amesos (KLU, Umfpack, SuperLU, Lapack), and Trilinos AztecOO (with built-in, Ifpack or ML preconditioners) which can take advantage of multi-core/cpu computers. Linear solvers that exploit general-purpose graphics processing units (GPGPU, such as NVidia CUDA) are also available ([[SuperLU\_CUDA]], CUSP) but in an early development stage.

#### 1.2 Licence

**DAE Tools** is free software and you can redistribute it and/or modify it under the terms of the GNU General Public Licence version 3 as published by the Free Software Foundation (GNU philosophy).

# 1.3 History

"Necessity, who is the mother of invention" Plato, Greek author & philosopher (427 BC - 347 BC), The Republic

"Every good work of software starts by scratching a developer's personal itch" Eric S. Raymond, hacker, The Cathedral and the Bazaar, 1997

The latter cannot be more true <sup>1</sup>. The early ideas of starting a project like this go back into 2007. At that time I have been working on my PhD thesis using one of commercially available process modelling software. It was everything nice and well until I discovered some annoying bugs and lack of certain highly appreciated features. The developers of that proprietary program (as it is a case with all proprietary computer programs) had their own agenda fixing only what they wanted to fix and introducing new features that they anticipated. Although I was able to improve the code and introduce certain features which will help (not only) me - I was helpless. The source code was not available and nobody will ever consider giving it to me to create patches with bugs fixes/new features. Not even if I swear on the holy (c++) bible!!

Very soon the contours of a new process modelling software slowly began to form. It took me a while until I made a definite plan and initial features, and I had to abandon a couple of initial versions...

"Plan to throw one away; you will, anyhow" Eric S. Raymond, hacker, The Cathedral and the Bazaar, 1997

Damn you Eric Raymond, interfering with my business again! :-) The new project was officially born early next year - 2008.

<sup>&</sup>lt;sup>1</sup> However, I do not agree with Eric Raymond and the Open Source Iniative views - they miss the point IMO, but let us leave it beside at the moment.

## 1.4 Acknowledgements

DAE Tools use the following third party free software libraries (GNU GPL, GNU LGPL, CPL, EPL, BSD or some other type of free/permissive/copy-left licences):

- Sundials IDAS: https://computation.llnl.gov/casc/sundials/main.html
- Boost: http://www.boost.org
- ADOL-C: https://projects.coin-or.org/ADOL-C
- Qt and pyQt4: http://qt.nokia.com, http://www.riverbankcomputing.co.uk/software/pyqt/intro
- Numpy: http://numpy.scipy.orghttp://numpy.scipy.org
- Scipy: http://www.scipy.org
- Blas/Lapack/CLapack: http://www.netlib.org
- Minpack: http://www.netlib.org/minpack
- Atlas: http://math-atlas.sourceforge.net
- Trilinos Amesos: http://trilinos.sandia.gov/packages/amesos
- Trilinos AztecOO: http://trilinos.sandia.gov/packages/aztecoo
- SuperLU/SuperLU\_MT: http://crd.lbl.gov/~xiaoye/SuperLU/index.html
- Umfpack: http://www.cise.ufl.edu/research/sparse/umfpack
- MUMPS: http://graal.ens-lyon.fr/MUMPS
- IPOPT: https://projects.coin-or.org/Ipopt
- Bonmin: https://projects.coin-or.org/Bonmin
- NLOPT: http://ab-initio.mit.edu/wiki/index.php/NLopt
- CUSP: http://code.google.com/p/cusp-library

**DAE Tools** can optionally use the following proprietary software libraries:

- AMD ACML linear solver (pyAmdACML module): http://www.amd.com/acml
- Intel MKL linear solvers (pyIntelMKL and pyIntelPardiso modules): http://software.intel.com/en-us/articles/intel-mkl

Please see the corresponding websites for more details about the licences.

#### 1.5 How to cite

If you use **DAE Tools** in your work then please cite it in the following way: D. Nikolic, DAE Tools process modelling software, 2010. http://www.daetools.com

# PROGRAMMING PARADIGMS

# 2.1 The Hybrid approach

In general, there are two types of approaches that can be applied to process modelling: Domain Specific Language approach and a general-purpose programming language approach (such as c/c++, Java or Python). A Domain Specific Language (DSL) is a special-purpose programming or specification language dedicated to a particular problem domain and so designed that it directly supports the key concepts necessary to describe the underlying problems. A domain-specific language is created specifically to solve problems in a particular domain and is usually not intended to be able to solve problems outside it (although that may be technically possible in some cases). In contrast, general-purpose languages are created to solve problems in a wide variety of application domains.

Domain-specific languages are languages with very specific goals in design and implementation and commonly lack low-level functions for filesystem access, interprocess control, and other functions that characterize full-featured programming languages, scripting or otherwise.

A good example of general purpose (multi-domain) domain specific language is Modelica while single-domain (chemical processing industry related) DSLs are gPROMS, Ascend, SpeedUp etc.

**DAE Tools** approach is a sort of the hybrid approach: it applies general-purpose programming languages such as c++ and Python, but offers a class-hierarchy/API that resembles a syntax of a DSL as much as possible, an access to the low-level functions, large number of standard and third party libraries and uses state of the art free/open-source software components to accomplish particular tasks (calculating derivatives and sensitivities, solving systems of differential and algebraic systems of equations and optimization problems, processing and plotting results etc).

API comparison between Modelica, gPROMS and DAE Tools:

```
class BufferTank(daeModel):
    model BufferTank
                                                                                                             __init__(self, Name, Parent = None, Description = daeModel.__init__(self, Name, Parent, Description)
      /* Import libs */
import Modelica.Math.*;
                                                                                                              self.Density = daeParameter("Density",
                                                   PARAMETER
       parameter Real Density;
                                                                                                                              = daeParameter(
                                                                                                                                                              unit
                                                                                                                                                                        self
                                                      Density as Real
CrossSectionalArea as Real
                                                                                                                             = daeParameter("Alpha",
       parameter Real CrossSectionalA
                                                                                                              self.Alpha
                                                                                                                                                                        self)
      parameter Real Alpha;
                                                                                                             self.HoldUp = daeVariable("HoldUp",
self.FlowIn = daeVariable("FlowIn",
self.FlowOut = daeVariable("FlowOut",
self.Height = daeVariable("Height",
                                                      Alpha as Real
                                                                                               10
11
12
13
14
15
       Real HoldUp(start = 0.0);
                                                                                                                                                             no_t,
                                                6
                                                   VARTABLE
       Real FlowIn:
                                                      HoldUp as Mass
                                                                                                                                                             no_t,
                                                8
                                                      FlowIn as Flowrate
       Real Height:
                                                      FlowOut as Flowrate
                                                                                                        def DeclareEquations(self):
                                                      Height as Length
                                                                                               16
17
 14 equation
                                                                                                             # Mass balance
eq = self.CreateEquation("MassBalance"
                                                   EQUATION
      der(HoldUp) = FlowIn - FlowOut;
                                                                                                             eq.Residual = self.HoldUp.dt() - self.FlowIn() + self.Flow
                                                                                               18
                                                      # Mass balance
                                                      $HoldUp = FlowIn - FlowOut;
 18 // Relation betwee liquid level
                                                                                                             # Relation between liquid level and holdup
                                                                                               20
       HoldUp = CrossSectionalArea
                                                                                                                          CreateEquation("LiquidLevelHol
                                                      # Relation betwee liquid leve
                                                                                                             eq.Residual = self.HoldUp() - self.Area()
                                                                                                                                                                     self.Height()
    // Relation between pressure dro
FlowOut = Alpha * sqrt(Height)
                                                      HoldUp = CrossSectionalArea
                                                                                                             # Relation between pressure drop and flow
                                                      # Relation between pressure dr
                                                                                                             eq = self.CreateEquation("PressureDropFlow"
eq.Residual = self.FlowOut() - self.Alpha()
 24 end BufferTank;
                                                      FlowOut = Alpha * sqrt(Height)
                                                                                                                                                                       Sqrt(self.He:
a) Modelica
                                               b) gPROMS
                                                                                              c) DAE Tools
```

**DAE Tools** provide low-level concepts such as parameters, variables, equations, ports, models, state transition networks, discrete events etc. so that the key concepts from new application domains can be added on top of those low level concepts. For instance, the key modelling concepts from the simulator-independent xml-based domain specific language for modelling of biological neural networks NineML such as neurones, synapses, connectivity patterns, populations of neurones, projections etc. are based on **DAE Tools** low-level concepts.

Side-by-side comparison between the DSL approach and the **DAE Tools** hybrid approach:

DSL Approach	DAE Tools Approach
Domain-specific languages allow solutions to be expressed	Modelling concepts cannot be expressed directly in the
in the idiom and at the level of abstraction of the problem	programming language and have to be emulated in the API
domain (direct support for all modelling concepts by the	or in some other way
language syntax)	
Clean, concise, ellegant and natural way of building model	The support for modelling concepts is much more verbose
descriptions: the code can be self documenting	and less elegant; however, DAE Tools can generate
	XML+MathML based model reports that can be either
	rendered in XHTML format using XSLT transformations
	(representing the code documentation) or used as an
	XML-based model exchange language.
Domain-specific languages could enhance quality,	2 2 2
productivity, reliability, maintainability and portability	
DSLs could be and often are simulator independent	Programming language dependent; however, a large
making a model exchange easier	number of scientific software libraries exposes its
making a model exchange easier	functionality to Python via Python wrappers
Cost of designing, implementing, and maintaining a	A compiler/lexical parser/interpreter is an integral part of
domain-specific language as well as the tools required to	the programming language (c++, Python) with a robust
develop with it (IDE): a compiler/lexical parser/interpreter	error handling, universal grammar and massively tested
must be developed with all burden that comes with it (such	
as error handling, grammar ambiguities, hidden bugs etc)	
Cost of learning a new language vs. its limited	No learning of a new language required (everything can
applicability: users are required to master a new language	get done in a favourite programming language)
(yet another language grammar)	
Increased difficulty of integrating the DSL with other	Calling external functions/libraries is a natural and
components: calling external functions/libraries and	straightforward Interaction with other software is natural
interaction with other software is limited by the existence	and straightforward
of wrappers around a simulator engine (for instance some	
scripting languages like Python or javascript)	
Models usually cannot be created in the runtime/on the fly	Models can be created in the runtime/on the fly and easily
(or at least not easily) and cannot be modified in the	modified in the runtime
runtime	
Setting up a simulation (ie. the values of parameters	Setting up a simulation is done programmaticaly and the
values, initial conditions, initially active states) is	initial values can be obtained from some other software in
embedded in the language and it is typically difficult to do	a natural way (chaining several software calls is easy since
it on the fly or to obtain the values from some other	a large number of libraries make Python wrappers
software (for example to chain several software calls	available)
where outputs of previous calls represent inputs to the	
subsequent ones)	
Simulation operating procedures are not flexible;	Operating procedures are completely flexible (within the
manipulation of model parameters, variables, equations,	limits of a programming language itself) and a
simulation results etc is limited to only those operations	manipulation of model parameters, variables, equations,
provided by the language	simulation results etc can be done in any way which a user
provided by the language	cosiders suitable for his/her problem
Only the type of results provided by the	
Only the type of results provided by the	The results processing can be done in any way which a
language/simulator is available; custom processing is	user considers suitable(again within the limits of a
usually not possible or if a simulator does provide a way to	programming language itself)
build extensions it is limited to the functionality made	
available to them	

# 2.2 The Equation-Oriented approach

In general, three approaches to process modelling exist (1):

- Sequential Modular (SeqM) approach
- Simultaneous Modular (SimM) approach
- Equation-Oriented (EO) approach

<sup>&</sup>lt;sup>1</sup> Morton, W., Equation-Oriented Simulation and Optimization. *Proc. Indian Natl. Sci. Acad.* 2003, 317-357.

The pros & cons of the first two approaches are extensively studied in the literature. Under the **EO** approach we generate and gather together all equations and variables which constitute the model representing the process. The equations are solved simultaneously using a suitable mathematical algorithm (Morton, 2003 <sup>1</sup>). Equation-oriented simulation requires simultaneous solution of a set of differential algebraic equations (**DAE**) which itself requires a solution of a set of nonlinear algebraic equations (**NLAE**) and linear algebraic equations (**LAE**). The Newton's method or some variant of it is almost always used to solve problems described by NLAEs. A brief history of Equation-Oriented solvers and comparison of **SeqM** and **EO** approaches as well as descriptions of the simultaneous modular and equation-oriented methods can be found in Morton, 2003 (<sup>1</sup>). Also a good overview of the equation-oriented approach and its application in gPROMS is given by Barton & Pantelides (<sup>2</sup>, <sup>3</sup>, <sup>4</sup>).

DAE Tools use the Equation-Oriented approach to process modelling, and the following types of processes can be modelled:

- · Lumped and distributed
- Steady-state and dynamic

Problems can be formulated as linear, non-linear, and (partial) differential algebraic systems (of index 1). The most common problems are initial value problems of implicit form. Equations can be ordinary or discontinuous, where discontinuities are automatically handled by the framework. A good overview of discontinuous equations and a procedure for location of equation discontinuities is given by Park & Barton (5) and in Sundials IDA (used in DAE Tools).

The main characteristics of the Equation-oriented (acausal) approach:

• Equations are given in an implicit form (as a residual):

$$F(\dot{x}, x, y, p) = 0$$

where x and  $\dot{x}$  are state variables and their derivatives, y are degrees of freedom and p are parameters.

· Input-Output causality is not fixed

The benefits are:

- · Increased model re-use
- Support for different simulation scenarios (based on a single model) by specifying different degrees of freedom. For instance, an equation given in the following form:

$$x_1 + x_2 + x_3 = 0$$

can be used to determine either x1, x2 or x3 depending on what combination of variables is known:

$$x_1 = -x_2 - x_3$$

$$\vee$$

$$x_2 = -x_1 - x_3$$

$$\vee$$

$$x_3 = -x_1 - x_2$$

# 2.3 The Object-Oriented approach

The Object-Oriented approach to process modelling is adopted in **DAE Tools**. The main characteristics of such an approach are:

- · Everything is an object
- Models are classes derived from the base daeModel class

<sup>&</sup>lt;sup>2</sup> Pantelides, C. C., and P. I. Barton, Equation-oriented dynamic simulation current status and future perspectives, *Computers & Chemical Engineering*, vol. 17, no. Supplement 1, pp. 263 - 285, 1993.

<sup>&</sup>lt;sup>3</sup> Barton, P. I., and C. C. Pantelides, gPROMS - a Combined Discrete/Continuous Modelling Environment for Chemical Processing Systems, *Simulation Series*, vol. 25, no. 3, pp. 25-34, 1993.

<sup>&</sup>lt;sup>4</sup> Barton, P. I., and C. C. Pantelides, Modeling of combined discrete/continuous processes", *AIChE Journal*, vol. 40, pp. 966-979, 1994.

<sup>&</sup>lt;sup>5</sup> Park, T., and P. I. Barton, State event location in differential-algebraic models", *ACM Transactions on Modeling and Computer Simulation*, vol. 6, no. 2, New York, NY, USA, ACM, pp. 137-165, 1996.

- Basically all OO concepts supported by the target language (c++, Python) are allowed, except few exceptions: \* Multiple inheritance is supported \* Models can be parametrized (using templates in c++) \* Derived classes always inherit all declared parameters, variables, equations etc. (polymorphism achieved through virtual functions where the declaration takes place) \* All parameters, variables, equations etc. remain public
- · Hierarchical model decomposition

## 2.4 Programming language

**DAE Tools** core libraries are written in standard c++. However, Python programming language is used as the main modelling language. The main reason for use of Python is (as the authors say): "Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms" link.

And: "Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast" link. Also, please have a look on a comparison to the other languages. Based on the information available online, and according to the personal experience, the python programs are much shorter and take an order of magnitude less time to develop it. Initially I developed daePlotter module in c++; it took me about one month of part time coding. But, then I moved to python: reimplementing it in PyQt took me just two days (with several new features added), while the code size shrank from 24 cpp modules to four python modules only!

"Where Python code is typically 3-5 times shorter than equivalent Java code, it is often 5-10 times shorter than equivalent C++ code! Anecdotal evidence suggests that one Python programmer can finish in two months what two C++ programmers can't complete in a year. Python shines as a glue language, used to combine components written in C++" link. Obviously, not everything can be developed in python; a heavy c++ artillery is still necessary for highly complex projects.

# **ARCHITECTURE**

**DAE Tools** consists of several interdependent components:

- Model
- Simulation
- Optimization
- · DAE solver
- · LA solver
- NLP solver
- Log
- Data reporter
- Data receiver

The components are located in the following modules:

- pyCore module (Model and Log components)
- pyActivity module (Simulation and Optimization components)
- pyIDAS module (DAE solver component)
- pyDataReporting module (Data reporter and Data receiver components)
- pyUnits module
- Large number of third party linear equation solver modules (LA solver component): pySuperLU, pySuperLU\_MT, pyTrilinos
- Large number of third party NLP/MINLP solver modules (NLP solver component): pyIPOPT, pyBONMIN, pyNLOPT

An overview of **DAE Tools** components and their interdepedency is presented in the *DAE Tools architecture*.

# 3.1 pyCore module

pyCore module defines the key modelling concepts such as:

• Model

A model of the process is a simplified abstraction of real world process/phenomena describing its most important/driving elements and their interactions. In **DAE Tools** models are created by defining their parameters, distribution domains, variables, equations, and ports.

• Distribution domain

Domain is a general term used to define an array of different objects (parameters, variables, equations but models and ports as well).

Parameter

Parameter can be defined as a time invariant quantity that will not change during a simulation.

Variable

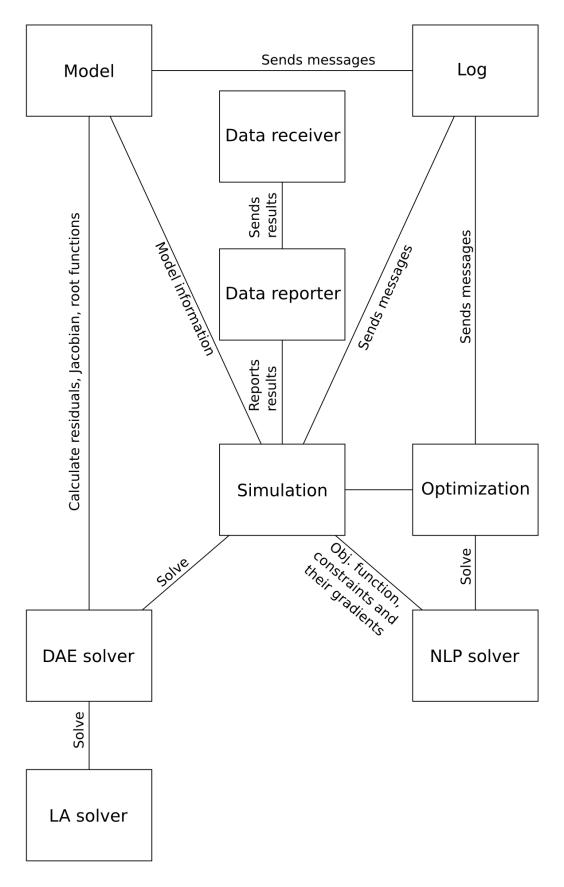


Figure 3.1: **DAE Tools** architecture

Variable can be defined as a time variant quantity, also called a *state variable*.

#### • Equation

Equation can be defined as an expression used to calculate a variable value, which can be created by performing basic mathematical operations (+, -, \*, /) and functions (such as sin, cos, tan, sqrt, log, ln, exp, pow, abs etc) on parameter and variable values (and time and partial derivatives as well).

#### · State transition network

State transition networks are used to model a special type of equations: discontinuous equation\*s. Discontinuous equations are equations that take different forms subject to certain conditions. They are composed of a finite number of \*states.

#### • State

States can be defined as a set of actions (in our case a set of equations) under current operating conditions. In addition, every state contains a set of state transitions which describe conditions when the state changes occur.

#### • OnEvent and OnCondition event handlers

#### Port

Ports are objects used to connect two model instances and exchange continuous information. Like models, they may contain domains, parameters and variables.

#### EventPort

Event ports are objects used to connect two model instances and exchange discrete information (events/messages).

#### Log

Log is defined as an object used to send messages from the various parts of **DAE Tools** framework (messages from solvers or simulation).

# 3.2 pyActivity module

#### • Simulation

Simulation of a process can be considered as the model run for certain input conditions. To define a simulation, several tasks are necessary such as: specifying information about domains and parameters, fixing the degrees of freedom by assigning values to certain variables, setting the initial conditions and many other (setting the initial guesses, absolute tolerances, etc).

#### • Optimization

Process optimization can be considered as a process adjustment so as to minimize or maximize a specified goal while satisfying imposed set of constraints. The most common goals are minimizing cost, maximizing throughput, and/or efficiency. In general there are three types of parameters that can be adjusted to affect optimal performance:

- Equipment optimization
- · Operating procedures
- · Control optimization

# 3.3 pyDataReporting module

#### · Data Reporter

Data reporter is defined as an object used to report the results of a simulation/optimization. They can either keep the results internally (and export them into a file, for instance) or send them via TCP/IP protocol to the **DAE Tools** plotter.

• Data Receiver

Data receiver can be defined as on object which duty is to receive the results from a data reporter. These data can be later plotted or processed in some other ways.

# 3.4 pyIDAS module

Contains an implementation of the Sundials IDAS DAE solver.

# 3.5 pyUnits module

Defines two key concepts:

- Unit (SI)
  - 7 fundamental dimensions (length, mass, time, electrical current, temperature, luminous intensity, amount of substance) \* Multiplier \* Offset
- Quantity
- Value
- Unit

### 3.6 NLP/MINLP modules

Contain implementations of various NLP/MINLP solvers:

- IPOPT in the pyIPOPTmodule
- NLOPT in the pyNLOPT module
- BONMIN in the pyBONMIN module

#### 3.7 LA solver modules

Contain implementations of various third party linear equation solvers:

- Trilinos Amesos in the pyTrilinos module
- Trilinos AztecOO in the pyTrilinos module
- SuperLU in the pySuperLU module
- SuperLU\_MT in the pySuperLU\_MT module
- Intel MKL in the pyTrilinos module

# **GETTING DAE TOOLS**

**DAE Tools** (pyDAE module) is installed in daetools folder within site-packages (or dist-packages) folder under python (tipically /usr/local/lib/pythonXY/Lib or C:\PythonX.Y\Lib). The structure of the folders is the following:

- daetools
  - code\_generators
  - dae\_plotter
  - dae\_simulator
  - docs
  - examples
  - pyDAE
  - solvers
  - unit\_tests

# 4.1 System requirements

Supported platforms:

- GNU/Linux (i686, x86\_64, arm)
- Windows (32/64 bit)
- MacOS (x86, x86\_64)

Supported python versions:

- 2.6 (some older GNU/Linux distributions only)
- 2.7

Supported numpy versions:

- GNU/Linux (1.5, 1.6, 1.7)
- Windows (1.6)
- MacOS (1.6)

Mandatory packages:

- Python (2.7.x): http://www.python.org
- Numpy (1.5.x, 1.6.x, 1.7.x): http://numpy.scipy.org
- Scipy: http://www.scipy.org
- Matplotlib: http://matplotlib.sourceforge.net
- pyQt4 (4.x): http://www.riverbankcomputing.co.uk/software/pyqt

Optional packages (3rd party linear solvers):

- Intel Pardiso (proprietary)
- AMD ACML (proprietary)

For more information on how to install packages please refer to the documentation for the specific library. By default all versions (GNU/Linux, Windows and MacOS) come with the Sundials dense LU and Lapack linear solvers, SuperLU, Trilinos Amesos (with built-in support for KLU, SuperLU and Lapack linear solvers), Trilinos AztecOO (with built-in support for Ifpack and ML preconditioners), NLOPT and IPOPT/BONMIN (with MUMPS linear solver and PORD ordering). Standalone SuperLU\_MT is available on GNU/Linux and MacOS versions only. Additional linear solvers: AMD ACML and Intel Pardiso must be downloaded separately and compiled from source since they are not free software.

## 4.2 Getting the packages

The instalation files can be downloaded from: https://sourceforge.net/projects/daetools/files

**Note**: From the version 1.2.1 **DAE Tools** use distutils to distribute python packages and extensions.

The naming convention of the installation files:

```
daetools-major.minor.build-platform-architecture-python_version.tar.gz
```

where major.minor.build represents the version (1.2.1 for instance), architecture could be i686, x86\_64 or universal, and python\_version can be py26, py27 etc. An example: daetools-1.2.1-gnu\_linux-x86\_64-py27.tar.gz is the version 1.2.1 for 64 bit GNU/Linux with python 2.7.

For the other platforms, architectures and python versions not listed in System requirements daetools must be compiled from the source. The source code can be downloaded either from the subversion tree or from the folder with a particular version (daetools-1.2.1-source.tar.gz for instance).

#### 4.3 Installation

#### 4.3.1 GNU/Linux

First install the mandatory packages: python 2.7, numpy 1.5/1.6/1.7, scipy, matplotlib and pyqt4.

In Debian GNU/Linux and derivatives use the Synaptic Package Manager or type the following commands:

```
sudo apt-get install python-numpy python-scipy python-matplotlib python-qt4 mayavi2
```

In Red Hat and derivatives use the package manager or type the following commands:

```
sudo yum install numpy scipy python-matplotlib PyQt4 Mayavi
```

Then unpack the downloaded archive, cd to the daetools-X.Y.Z folder and install **DAE Tools** by typing the following shell command:

```
sudo python setup.py install
```

#### 4.3.2 MacOS

First install the mandatory packages: **python 2.7**, **numpy 1.6**, **scipy**, **matplotlib** and **pyqt4**. As a starting point the following links can be used:

- Python 2.7: http://www.python.org/ftp/python/2.7.3/python-2.7.3-macosx10.6.dmg
- Numpy: http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/numpy-1.6.2-py2.7-python.org-macosx10.6.dmg/download
- Scipy: http://sourceforge.net/projects/scipy/files/scipy/0.10.1/scipy-0.10.1-py2.7-python.org-macosx10.6.dmg/download
- Matplotlib: http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.0/matplo
- PyQt4: http://www.riverbankcomputing.co.uk/static/Downloads/PyQt4downloadsection

Then unpack the downloaded archive, cd to the daetools-X.Y.Z folder and install **DAE Tools** by typing the following shell command:

```
sudo python setup.py install
```

#### 4.3.3 Windows

**DAE Tools** is compiled and tested on a 32-bit Windows XP and Windows 7. In order to use **DAE Tools** on 64-bit versions of Windows the 32-bit versions of python, pyqt, numpy and scipy packages should be installed. First install the mandatory packages: python 2.7, numpy 1.6, scipy, matplotlib and pyqt4. As a starting point the following links can be used:

- Python 2.7: http://www.python.org/ftp/python/2.7.3/python-2.7.3.msi
- Numpy: http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/numpy-1.6.2-win32-superpack-python2.7.exe/download
- Scipy: http://sourceforge.net/projects/scipy/files/scipy/0.10.1/scipy-0.10.1-win32-superpack-python2.7.exe/download
- Matplotlib: http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.0/matplotlib-1.1.0.win32-py2.7.exe/download
- PyQt4: http://www.riverbankcomputing.co.uk/static/Downloads/PyQt4downloadsection

To be able to create 3D plots you need to install Mayavi2 package. Alternatively you can install everything needed through Python(x,y). Finally, install **DAE Tools** by double clicking the file daetools\_x.x-x-win32\_py27.exe and follow the instructions. To uninstall use the uninstall program in Start -> All Programs -> DAE Tools -> Uninstall.

## 4.4 Compiling from source

To compile the **DAE Tools** the following is needed:

- Installed python, numpy, and scipy modules
- Compiled third party libraries and DAE/LA/NLP solvers: Sundials IDAS, Bonmin, NLopt, Trilinos, SuperLU, SuperLU\_MT, Blas/Lapack

All **DAE Tools** modules are developed using the QtCreator/QMake cross-platform integrated development environment. The source code can be downloaded from the SourceForge website or checked out from the DAE Tools subversion repository:

```
svn checkout svn://svn.code.sf.net/p/daetools/code daetools
```

#### 4.4.1 GNU/Linux and MacOS

#### The easy way

First, install all the necessary dependencies by executing install\_dependencies\_linux.sh shell script located in the trunk directory. It will check the OS you are running (currently Debian, Ubuntu, Linux Mint, CentOS and Fedora are supported but other can be easily added) and install all necessary packages needed for **DAE Tools** development.

```
# 'lsb_release' command might be missing on some GNU/Linux platforms
# and has to be installed before proceeding.
# On Debian based systems:
# sudo apt-get install lsb-release
# On red Hat based systems:
# sudo yum install redhat-lsb

cd daetools/trunk
sh install_dependencies_linux.sh
```

Then, compile the third party libraries by executing <code>compile\_libraries\_linux.sh</code> shell script located in the <code>trunk</code> directory. The script will download all necessary source archives from the <code>DAE Tools</code> SourceForge web-site, unpack them, apply changes and compile them. If all dependencies are installed there should not be problems compiling the libraries.

```
sh compile_libraries_linux.sh
```

**Note 1:** There is a bug in Sundials IDAS library. When compiling fails, go to the folder trunk/idas and change the line 24 (or somewhere around it) in the Makefile: top builddir = ``to `to builddir = ..

**Note 2:** There are known problems to compile the older bonmin and trilinos libraries using GNU GCC 4.6. This has been fixed in bonmin 1.5+ and trilinos 10.8+ versions. Therefore, either GCC 4.5 and below or the recent versions of bonmin/trilinos libraries should be used.

Finally, compile the **DAE Tools** libraries and python modules by executing compile\_linux.sh shell script located in the trunk directory. The script accepts one argument specifying projects that should be compiled. Any of the following is accepted: all, core, pydae, solvers, superlu, superlu\_mt, superlu\_cuda, cusp, trilinos, bonmin, ipopt, and nlopt. If all is specified the script will compile dae, superlu, superlu\_mt, trilinos, bonmin, ipopt, and nlopt projects.

```
sh compile_linux.sh all
# Or for instance:
# sh compile_linux.sh dae superlu nlopt
```

All python extensions should be placed in trunk/daetools-package/daetools/pyDAE and trunk/daetools-package/daetools/solvers folders. **DAE Tools** can be now installed by using the following commands:

```
cd daetools-package
sudo python setup.py install
```

#### From QtCreator IDE

DAE Tools can also be compiled from within QtCreator IDE. First install dependencies and compile third party libraries (as explained in *The easy way*) and then do the following:

- Do not do the shadow build. Uncheck it (for all projects) and build everything in the release folder
- Choose the right specification file for your platform (usually it is done automatically by the IDE, but double-check it):
- for GNU/Linux use -spec linux-g++
- for MacOS use -spec macx-g++
- Compile the dae project (you can add the additional Make argument jN to speed-up the compilation process, where N is the number of processors plus one; for instance on the quad-core machine you can use j5)
- Compile SuperLU\_SuperLU\_MT/SuperLU\_CUDA and Bonmin/Ipopt solvers. SuperLU/SuperLU\_MT/SuperLU\_CUDA and Bonmin/Ipopt share the same code and the same project file so some hacking is needed. Here are the instructions how to compile them:
- Compiling libcdaeBONMIN\_MINLPSolver.a and pyBONMIN.so:
  - Set CONFIG += BONMIN in BONMIN\_MINLPSolver.pro, run qmake and then compile
  - Set CONFIG += BONMIN in pyBONMIN.pro, run qmake and then compile
- Compiling libcdaeIPOPT\_NLPSolver.a and pyIPOPT.so:
  - Set CONFIG += IPOPT in BONMIN\_MINLPSolver.pro, run qmake and then compile
  - Set CONFIG += IPOPT in pyBONMIN.pro, run qmake and then compile
- Compiling libcdaeSuperLU\_LASolver.a and pySuperLU.so:
  - Set CONFIG += SuperLU in LA\_SuperLU.pro, run qmake and then compile
  - Set CONFIG += SuperLU in pySuperLU.pro, run qmake and then compile
- Compiling libcdaeSuperLU\_MT\_LASolver.a and pySuperLU\_MT.so:
  - Set CONFIG += SuperLU\_MT in LA\_SuperLU.pro, run qmake and then compile
  - Set CONFIG += SuperLU\_MT in pySuperLU.pro, run qmake and then compile
- $\bullet$  Compiling <code>libcdaeSuperLU\_CUDA\_LASolver.a</code> and <code>pySuperLU\_CUDA.so:</code>
  - Set CONFIG += SuperLU\_CUDA in LA\_SuperLU.pro, run qmake and then compile

- Set CONFIG += SuperLU\_CUDA in pySuperLU.pro, run qmake and then compile
- Compile the LA\_Trilinos\_Amesos project

#### 4.4.2 Windows

Necessary tools: QtCreator, Microsoft VC++ and G95 Fortran compiler (Mumps only).

**Note:** Compiling all third party libraries and **DAE Tools** projects requires a mental gymnastics impossible to describe by any human language so that the pre-compiled libraries are provided in the downloads section (windows libraries).

**DAE Tools** should be compiled from within QtCreator IDE:

- Unpack the downloaded archive bonmin-trilinos-idas-superlu-nlopt-mumps-g95-msvc-win32.zip into the daetools/trunk folder. All libraries are compiled with MS VC++ 2008 Express edition (the most likely other versions of MS VC++ will also work). Mumps Fortran 95 files are compiled with G95 Fortran compiler.
- Path to libf95.a and libgcc.a libraries should be set in dae.pri config file. For instance, if G95 is installed in c:\g95 set the G95\_LIBDIR variable to: G95\_LIBDIR = c:\g95\lib\gcc-lib\i686-pc-mingw32\4.1.2
- Follow the instructions for compiling **DAE Tools** described in *From QtCreator IDE* section above.

Note: superlu\_mt and superlu\_cuda cannot be compiled on Windows at the moment.

DAE Tools can be installed by using the following commands:

cd daetools-package
sudo python setup.py install

# **GETTING STARTED WITH DAE TOOLS**

This chapter gives the basic information about what is needed to develop a model of a process, how to simulate/optimize it and how to obtain and plot the results of a process simulation/optimization. In general, the simulation/optimization of a process consists of three tasks:

- 1. Modelling of a proces
- 2. Defining a simulation/optimization
- 3. Processing the results

# 5.1 Running tutorials

- 1. Start daePlotter:
  - GNU/Linux:

Run Applications/Programming/daePlotter from the system menu or execute the following shell command:

daeplotter

• MacOS:

Execute the following shell command:

daeplotter

· Windows:

Run Start/Programs/DAE Tools/daePlotter from the Start menu.

The daePlotter main window should appear (given in daePlotter main window.)



Figure 5.1: daePlotter main window.

- 2. Start DAE Tools Examples program to try some examples:
  - GNU/Linux:

Run Applications/Programming/DAE Tools Examples from the system menu or execute the following shell command:

daeexamples

• MacOS:

Execute the following shell command:

daeexamples

· Windows:

Run Start/Programs/DAE Tools/DAE Tools Examples from the Start menu.

The main window of DAE Tools Examples application is given in *DAE Tools Examples main window* while the output from the simulation run in *A typical optimization output from DAE Tools*. Users can select one of several tutorials, run them, and inspect their source code or model reports. Model reports open in a new window of the system's default web browser (however, only Mozilla Firefox is currently supported because of the MathML rendering issue).

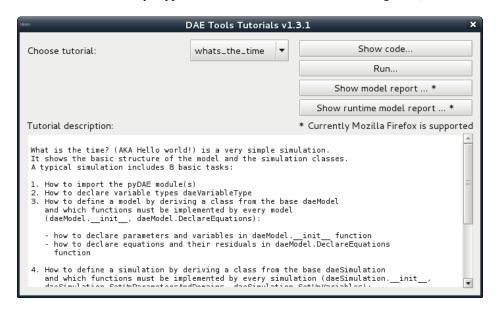


Figure 5.2: DAE Tools Examples main window

Tutorials can also be started from the shell:

```
cd /usr/local/lib/python2.7/dist-packages/daetools/examples
# Or in windows:
# cd C:\PythonX.Y\Lib\site-packages\daetools\examples
python tutorial1.py console
```

The sample output is given in *Shell output from the simulation*:

#### 5.2 Models

#### 5.2.1 Developing a model

In **DAE Tools** models are developed by deriving a new class from the base model class (daeModel). The process consists of two steps:

- 1. Declare all domains, parameters, variables, ports etc.:
  - In **pyDAE** declare and instantiate in the \_\_init\_\_() function
  - In cDAE declare as class data members and instantiate in the constructor
- 2. Declare equations and state transition networks in the DeclareEquations () function

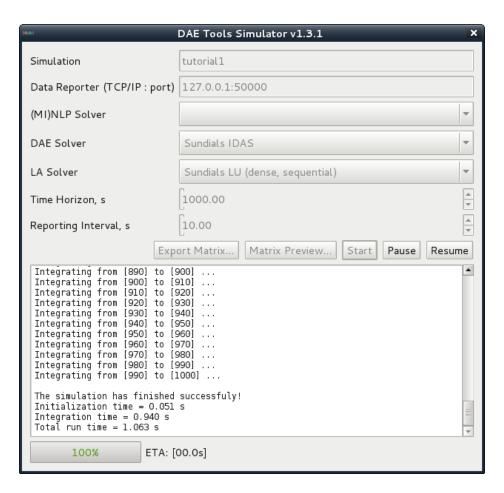


Figure 5.3: A typical optimization output from DAE Tools

An example model developed in **pyDAE** (using python programming language):

```
class myModel(daeModel):
           def __init__(self, name, parent = None, description = ""):
                       daeModel.__init__(self, name, parent, description)
                       # Declaration/instantiation of domains, parameters, variables, ports, etc:
                                              = daeParameter("m",
                                                                                                                                                                 self, "Mass of the copper plate")
                       self.m
                                                                                                                      kg,
                                                                                                                                                                        self, "Specific heat capacity of the plate
                       self.cp
                                                     = daeParameter("c_p",
                                                                                                                                 J/(kg*K),
                       \texttt{self.alpha} = \texttt{daeParameter("\α", W/((m**2)*K), self, "Heat transfer coefficient")}
                                                                                                                                                                          self, "Area of the plate")
                       self.A
                                            = daeParameter("A",
                                                                                                                                m**2,
                       self.Tsurr = daeParameter("T_surr", K,
                                                                                                                                                                          self, "Temperature of the surroundings")
                                                                                                                                                               self, "Power of the heater")
                       self.Qin
                                                      = daeVariable("Q_in", power_t,
                                                                                                                        temperature_t, self, "Temperature of the plate")
                       self.T
                                                     = daeVariable("T",
           def DeclareEquations(self):
                       # Declaration of equations and state transitions:
                       eq = self.CreateEquation("HeatBalance", "Integral heat balance equation")
                       eq.Residual = self.m() * self.cp() * self.T.dt() - self.Qin() + self.alpha() * self.A() * (self.alpha() * self.alpha() * sel
The same model developed in cDAE (using c++ programming language):
```

```
class myModel : public daeModel
public:
    // Declarations of domains, parameters, variables, ports, etc:
    daeParameter mass;
    daeParameter c_p;
    daeParameter alpha;
    daeParameter A;
    daeParameter T_surr;
    daeVariable Q_in;
```

5.2. Models 21

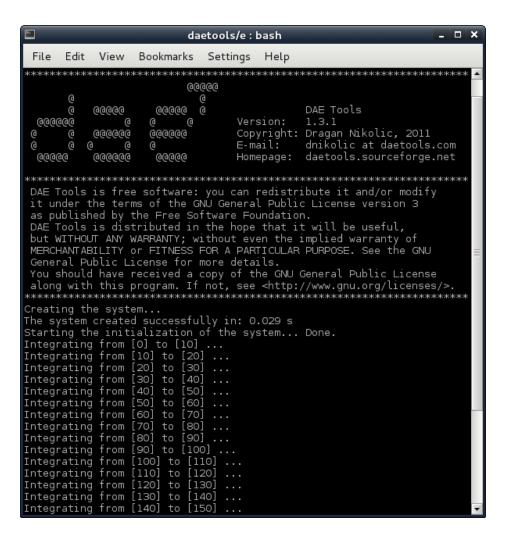


Figure 5.4: Shell output from the simulation

```
daeVariable T;
public:
   myModel(string strName, daeModel* pParent = NULL, string strDescription = "")
    : daeModel(strName, pParent, strDescription),
    // Instantiation of domains, parameters, variables, ports, etc:
                                   this, "Mass of the copper plate"),
   mass ("m",
                    kg,
                  J/(kg*K), this, "Specific heat capacity of the plate"),
         ("c_p",
    с_р
    alpha ("α", W/((m^2) * K), this, "Heat transfer coefficient"),
                                   this, "Area of the plate"),
         ("A",
                   m ^ 2,
   T_surr("T_surr", K,
                                    this, "Temperature of the surroundings"),
                                   this, "Power of the heater"),
    Q_in ("Q_in", power_t,
         ("T",
                    temperature_t, this, "Temperature of the plate")
   Т
    void DeclareEquations(void)
        // Declaration of equations and state transitions:
        daeEquation* eq = CreateEquation("HeatBalance", "Integral heat balance equation");
        eq->SetResidual( mass() \star c_p() \star T.dt() - Q_in() + alpha() \star A() \star (T() - T_surr()) );
};
```

More information about developing models can be found in *pyDAE User Guide* and pyCore.daeModel. Also, do not forget to have a look on *Tutorials*.

#### 5.3 Simulation

#### 5.3.1 Setting up a simulation

Definition of a simulation in **DAE Tools** requires the following steps:

- 1. Deriving a new simulation class from the base simulation class (daeSimulation)
- Specification of a model to be simulated
- Setting the values of parameters
- Fixing the degrees of freedom by assigning the values to certain variables
- Setting the initial conditions for differential variables
- Setting the other variables' information: initial guesses, absolute tolerances, etc
- Specifation of an operating procedure. It can be either a simple run for a specified period of time (default) or a complex one where various actions can be taken during the simulation
- 2. Specify DAE and LA solvers
- 3. Specify a data reporter and a data receiver, and connect them
- 4. Set a time horizon, reporting interval, etc
- 5. Do the initialization of the DAE system
- 6. Save model report and/or runtime model report (to inspect expanded equations etc)
- 7. Run the simulation

An example simulation developed in **pyDAE**:

```
class mySimulation(daeSimulation):
    def __init__(self):
        daeSimulation.__init__(self)

# Set the model to simulate:
    self.m = myModel("myModel", "Description")
```

5.3. Simulation 23

```
def SetUpParametersAndDomains(self):
        # Set the parameters values:
        self.m.cp.SetValue(385 * J/(kg*K))
        self.m.m.SetValue(1 * kg)
        self.m.alpha.SetValue(200 \star W/((m\star \star2) \starK))
        self.m.A.SetValue(0.1 * m**2)
        self.m.Tsurr.SetValue(283 * K)
    def SetUpVariables(self):
        # Set the degrees of freedom, initial conditions, initial guesses, etc.:
        self.m.Qin.AssignValue(1500 * W)
        self.m.T.SetInitialCondition(283 * K)
    def Run(self):
        # A custom operating procedure, if needed.
        # Here we use the default one:
        daeSimulation.Run(self)
The same simulation in cDAE:
class mySimulation : public daeSimulation
public:
    myModel m;
public:
    mySimulation(void) : m("myModel", "Description")
        // Set the model to simulate:
        SetModel (&m);
    }
public:
    void SetUpParametersAndDomains(void)
        // Set the parameters values:
        model.c_p.SetValue(385 * J/(kg*K));
        model.mass.SetValue(1 * kg);
        model.alpha.SetValue(200 \star W/((m<sup>2</sup>) \starK));
        model.A.SetValue(0.1 * (m^2));
        model.T_surr.SetValue(283 * K);
    }
    void SetUpVariables(void)
        // Set the degrees of freedom, initial conditions, initial guesses, etc.:
        model.Q_in.AssignValue(1500 * W);
        model.T.SetInitialCondition(283 * K);
    }
    void Run (void)
        // A custom operating procedure, if needed.
        // Here we use the default one:
        daeSimulation::Run();
    }
};
```

Simulations in **pyDAE** can be set-up to run in two modes:

1. From th PyQt4 graphical user interface (**pyDAE** only):

Here the default log, and data reporter objects will be used, while the user can choose DAE and LA solvers and specify time horizon and reporting interval.

```
# Import modules
import sys
from time import localtime, strftime
```

```
from PyQt4 import QtCore, QtGui
  # Create QtApplication object
  app = QtGui.QApplication(sys.argv)
  # Create simulation object
  sim = mySimulation()
  # Report ALL variables in the model
  sim.m.SetReportingOn(True)
  # Show the daeSimulator window to choose the other information needed for simulation
  simulator = daeSimulator(app, simulation=sim)
  simulator.show()
  # Execute applications main loop
  app.exec_()
2. From the shell:
  In pyDAE:
  # Import modules
  import sys
  from time import localtime, strftime
  # Create Log, Solver, DataReporter and Simulation object
              = daeStdOutLog()
              = daeIDAS()
  solver
  datareporter = daeTCPIPDataReporter()
  simulation = mySimulation()
  # Report ALL variables in the model
  simulation.m.SetReportingOn(True)
  # Set the time horizon (1000 seconds) and the reporting interval (10 seconds)
  simulation.SetReportingInterval(10)
  simulation.SetTimeHorizon(1000)
  # Connect data reporter (use the default TCP/IP connection settings: localhost and 50000 port)
  simName = simulation.m.Name + strftime(" [m.%Y %H:%M:%S]", localtime())
  if(datareporter.Connect("", simName) == False):
      sys.exit()
  # Initialize the simulation
  simulation. Initialize (solver, datareporter, log)
  # Solve at time = 0 (initialization)
  simulation.SolveInitial()
  # Run
  simulation.Run()
  # Clean up
  simulation.Finalize()
  In cDAE:
  // Create Log, Solver, DataReporter and Simulation object
  boost::scoped_ptr<daeSimulation_t>
pSimulation(new mySimulation());
  boost::scoped_ptr<daeDataReporter_t> pDataReporter(daeCreateTCPIPDataReporter());
  boost::scoped_ptr<daeIDASolver>
                                        pDAESolver(daeCreateIDASolver());
  boost::scoped_ptr<daeLog_t>
                                        pLog(daeCreateStdOutLog());
  // Report ALL variables in the model
  pSimulation->GetModel()->SetReportingOn(true);
  // Set the time horizon (1000 seconds) and the reporting interval (10 seconds)
```

5.3. Simulation 25

```
pSimulation->SetReportingInterval(10);
pSimulation->SetTimeHorizon(1000);

// Connect data reporter (use the default TCP/IP connection settings: localhost and 50000 port)
string strName = pSimulation->GetModel()->GetName();
if(!pDataReporter->Connect("", strName))
    return;

// Initialize the simulation
pSimulation->Initialize(pDAESolver.get(), pDataReporter.get(), pLog.get());

// Solve at time = 0 (initialization)
pSimulation->SolveInitial();

// Run
pSimulation->Run();

// Clean up
pSimulation->Finalize();
```

#### 5.3.2 Running a simulation

Simulations are started by executing the following shell commands:

```
cd "directory where simulation file is located"
python mySimulation.py
```

## 5.4 Optimization

#### 5.4.1 Setting up an optimization

To define an optimization problem it is first necessary to develop a model of the process and to define a simulation (as explained above). Having done these tasks (working model and simulation) the optimization in **DAE Tools** can be defined by specifying the objective function, optimization variables and optimization constraints. It is intentionally chosen to keep simulation and optimization tightly coupled. The optimization problem should be specified in the function <code>SetUpOptimization()</code>.

Definition of an optimization in **DAE Tools** requires the following steps:

- 1. Specify the objective function
- Objective function is defined by specifying its residual (similarly to specifying an equation residual); Internally the framework will create a new variable (V\_obj) and a new equation (F\_obj).
- 2. Specify optimization variables
- The optimization variables have to be already defined in the model and their values assigned in the simulation; they can be either non-distributed or distributed.
- Specify a type of optimization variable values. The variables can be continuous (floating point values in the given range), integer (set of integer values in the given range) or binary (integer value: 0 or 1).
- Specify the starting point (within the range)
- 3. Specify optimization constraints
- Two types of constraints exist in DAE Tools: equality and inequality constraints To define an equality constraint its residual and the value has to be specified; To define an inequality constraint its residual, the lower and upper bounds have to be specified; Internally the framework will create a new variable (V\_constraint[N]) and a new equation (F\_constraint[N]) for each defined constraint, where N is the ordinal number of the constraint.
- 4. Specify NLP/MINLP solver
- Currently BONMIN MINLP solver and IPOPT and NLOPT solvers are supported (the BONMIN solver internally uses IPOPT to solve NLP problems)

- 5. Specify DAE and LA solvers
- 6. Specify a data reporter and a data receiver, and connect them
- 7. Set a time horizon, reporting interval, etc
- 8. Set the options of the (MI)NLP solver
- 9. Initialize the optimization
- 10. Save model report and/or runtime model report (to inspect expanded equations etc)
- 11. Run the optimization

SetUpOptimization() function should be declared in the simulation class:

In **pyDAE**: .. code-block:: python

class mySimulation(daeSimulation): ...

def SetUpOptimization(self): # Declarations of the obj. function, opt. variables and constraints:

#### In cDAE:

Optimizations, like simulations can be set-up to run in two modes:

1. From the PyQt4 graphical user interface (pyDAE only)

Here the default log, and data reporter objects will be used, while the user can choose NLP, DAE and LA solvers and specify time horizon and reporting interval:

```
# Import modules
import sys
from time import localtime, strftime
from PyQt4 import QtCore, QtGui
# Create QtApplication object
app = QtGui.QApplication(sys.argv)
# Create simulation object
sim = mySimulation()
nlp = daeBONMIN()
# Report ALL variables in the model
sim.m.SetReportingOn(True)
# Show the daeSimulator window to choose the other information needed for optimization
simulator = daeSimulator(app, simulation=sim, nlpsolver=nlp)
simulator.show()
# Execute applications main loop
app.exec_()
```

2. From the shell:

#### In **pyDAE**:

5.4. Optimization 27

```
optimization = daeOptimization()
# Enable reporting of all variables
simulation.m.SetReportingOn(True)
# Set the time horizon and the reporting interval
simulation.ReportingInterval = 10
simulation.TimeHorizon = 100
# Connect data reporter
simName = simulation.m.Name + strftime(" [m.%Y %H:%M:%S]", localtime())
if (datareporter.Connect("", simName) == False):
    sys.exit()
# Initialize the opimization
optimization. Initialize (simulation, nlpsolver, daesolver, datareporter, log)
optimization.Run()
# Clean up
optimization.Finalize()
In cDAE:
// Create Log, NLPSolver, DAESolver, DataReporter, Simulation and Optimization objects
boost::scoped_ptr<daeSimulation_t>
                                          pSimulation (new mySimulation());
boost::scoped_ptr<daeDataReporter_t>
                                          pDataReporter(daeCreateTCPIPDataReporter());
boost::scoped_ptr<daeIDASolver>
                                          pDAESolver(daeCreateIDASolver());
boost::scoped_ptr<daeLog_t>
                                          pLog(daeCreateStdOutLog());
boost::scoped_ptr<daeNLPSolver_t>
                                          pNLPSolver(new daeBONMINSolver());
boost::scoped_ptr<daeOptimization_t>
                                          pOptimization(new daeOptimization());
// Report ALL variables in the model
pSimulation->GetModel()->SetReportingOn(true);
// Set the time horizon and the reporting interval
pSimulation->SetReportingInterval(10);
pSimulation->SetTimeHorizon(100);
// Connect data reporter
string strName = pSimulation->GetModel()->GetName();
if(!pDataReporter->Connect("", strName))
    return:
// Initialize the optimization
pOptimization->Initialize(pSimulation.get(),
                          pNLPSolver.get(),
                          pDAESolver.get(),
                          pDataReporter.get(),
                          pLog.get());
// Run
pOptimization.Run();
// Clean up
pOptimization.Finalize();
```

More information about simulation can be found in *pyDAE User Guide* and daeOptimization. Also, do not forget to have a look on *Tutorials*.

#### 5.4.2 Starting an optimization

Starting the optimization problems is analogous to running a simulation.

# 5.5 Processing the results

The simulation/optimization results can be easily plotted by using **DAE Plotter** application. It is possible to choose between 2D and 3D plots. After choosing a desired type, a **Choose variable** (given in *Choose variable dialog for a 2D plot*) dialog appears where a variable to be plotted can be selected and information about domains specified - some domains should be fixed while leaving another free by selecting  $\star$  from the list (to create a 2D plot one domain must remain free, while for a 3D plot two domains).

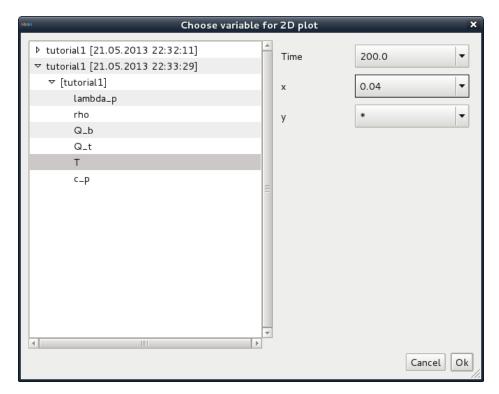


Figure 5.5: Choose variable dialog for a 2D plot

Typical 2D and 3D plots are given in Example 2D plot (produced by Matplotlib) and Example 3D plot (produced by Mayavi2).

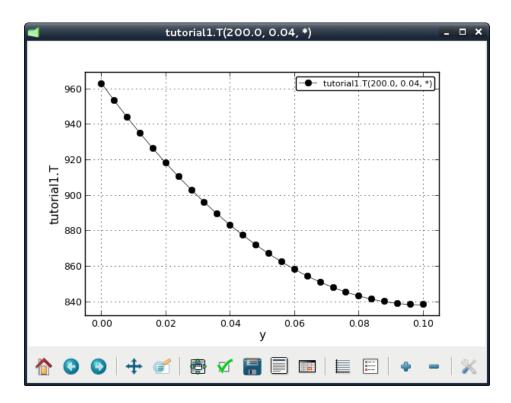


Figure 5.6: Example 2D plot (produced by Matplotlib)

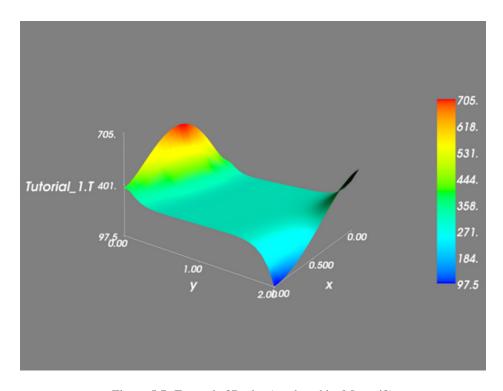


Figure 5.7: Example 3D plot (produced by Mayavi2)

**CHAPTER** 

SIX

# **PYDAE USER GUIDE**

6.1 The main concepts

# **PYDAE API REFERENCE**

# 7.1 Module pyCore

# 7.1.1 Overview

# 7.1.2 Key modelling concepts

...

# **Classes**

daeVariableType	
daeDomain	
daeParameter	
daeVariable	
daeModel	Base model class.
daeSTN	
daeIF	
daeEquation	
daeState	
daePort	
daeEventPort	
daePortConnection	
daeScalarExternalFunction	
daeVectorExternalFunction	
daeDomainIndex	
daeIndexRange	
daeArrayRange	
daeDEDI	
daeAction	
daeOnConditionActions	
daeOnEventActions	
daeOptimizationVariable	
daeObjectiveFunction	
daeOptimizationConstraint	
daeMeasuredVariable	
daeEquationExecutionInfo	

# ${f class}$ pyCore.daeVariableType

Bases: Boost.Python.instance

\_\_init\_\_ ((object)self, (str)name, (object)units, (float)lowerBound, (float)upperBound, (float)initialGuess, (float)absTolerance) → None

# AbsoluteTolerance

```
InitialGuess
      LowerBound
     Name
     Units
     UpperBound
class pyCore.daeObject
     Bases: Boost.Python.instance
      CanonicalName
     Description
      {\tt GetNameRelativeToParentModel}~((\textit{daeObject}) self~) \rightarrow {\tt str}
     GetStrippedName((daeObject)self) \rightarrow str
      \texttt{GetStrippedNameRelativeToParentModel} ((\textit{daeObject}) self) \rightarrow \text{str}
     Library
     Model
     Name
     Version
class pyCore.daeDomain
     Bases: pyCore.daeObject
     \_init\_((object)self, (str)name, (daeModel)parentModel, (object)units[, (str)description='']) \rightarrow None
           __init__( (object)self, (str)name, (daePort)parentPort, (object)units [, (str)description='']) -> None
      __getitem__((daeDomain)self, (int)index) \rightarrow adouble
       _call__((daeDomain)self, (int)index) \rightarrow adouble
     CreateArray((daeDomain)self, (int)noIntervals) \rightarrow None
      {\tt CreateDistributed} \ ((\textit{daeDomain}) \textit{self},
                                                                 (daeeDiscretizationMethod)discretizationMethod,
                                                                (int)numberOfIntervals,
                               (int)discretizationOrder,
                                                                                               (float)lowerBound,
                               (float)upperBound) \rightarrow None
     DiscretizationMethod
     DiscretizationOrder
     LowerBound
     NumberOfIntervals
     NumberOfPoints
     Points
      Type
     Units
     UpperBound
     npyPoints
class pyCore.daeParameter
      Bases: pyCore.daeObject
        \_init\_ ((object)self, (str)name, (object)units, (daePort)parentPort[, (str)description='[, (list)domains=[]
             []]) \rightarrow None
_init__( (object)self,
                                                                  (daeModel)parentModel [, (str)description="
                                     (str)name,
                                                  (object)units,
           (list)domains=[]]]) -> None
```

```
GetValue ((daeParameter)self[, (int)index1[, ...[, (int)index8]]]) \rightarrow float
           Gets the value of the parameter at the specified domain indexes. How many arguments index1, ...,
            index8 are used depends on the number of domains that the parameter is distributed on.
      GetQuantity ((daeParameter)self[, (int)index1[, ...[, (int)index8]]]) \rightarrow quantity
           Gets the value of the parameter at the specified domain indexes as the quantity object (with value and units).
            How many arguments index1, ..., index8 are used depends on the number of domains that the parameter
           is distributed on.
      \textbf{SetValue} ((\textit{daeParameter}) self \big[, (\textit{int}) \textit{index} 1 \big[, ... \big[, (\textit{int}) \textit{index} 8 \big] \big] \big], (\textit{float}) value) \rightarrow \textbf{None}
            Sets the value of the parameter at the specified domain indexes. How many arguments index1, ...,
            index8 are used depends on the number of domains that the parameter is distributed on.
      SetValue ((daeParameter)self[, (int)index1[, ...[, (int)index8]]], (quantity)value) \rightarrow None
            Sets the value of the parameter at the specified domain indexes. How many arguments index1, ...,
            index8 are used depends on the number of domains that the parameter is distributed on.
      SetValues ((daeParameter)self, (float)values) \rightarrow None
           Sets all values of the parameter.
      SetValues ((daeParameter)self, (quantity)values) \rightarrow None
           Sets all values of the parameter.
      array((daeParameter)self[, (object)index1[, ...[, (object)index8]]]) \rightarrow adouble_array
           Gets the array of parameter's values at the specified domain indexes (used to build equation residuals only). How
            many arguments index1, ..., index8 are used depends on the number of domains that the parameter is
            distributed on. Argument types can be one of the following:
               •daeIndexRange object
               •plain integer (to select a single index from a domain)
               python list (to select a list of indexes from a domain)
               •python slice (to select a range of indexes from a domain: start index, end index, step)
               •character ' *' (to select all points from a domain)
               •integer -1 (to select all points from a domain)
               •empty python list [] (to select all points from a domain)
      __call__((daeParameter)self[, (int)index1[, ...[, (int)index8]]]) \rightarrow adouble
            Gets the value of the parameter at the specified domain indexes (used to build equation residuals only). How many
            arguments index1, ..., index8 are used depends on the number of domains that the parameter is distributed
      \texttt{DistributeOnDomain} ((daeParameter)self, (daeDomain)domain) \rightarrow None
      Domains
      GetDomainsIndexesMap ((daeParameter)self, (int)indexBase) \rightarrow dict
      NumberOfPoints
      ReportingOn
      Units
      npyValues
class pyCore.daeVariable
      Bases: pyCore.daeObject
                                                                                              (daeModel)parentPort ,
                                                       (daeVariableType)variableType,
        init ((object)self,
                                      (str)name,
                   (str)description="([,(list)domains=[]]]) \rightarrow None
             _init__( (object)self, (str)name, (daeVariableType)variableType, (daePort)parentModel [, (str)description=" [,
           (list)domains=[]]]) -> None
      GetValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]]) \rightarrow float
           Gets the value of the variable at the specified domain indexes. How many arguments index1, ..., index8
            are used depends on the number of domains that the variable is distributed on.
```

 $\textbf{GetQuantity} ((\textit{daeVariable}) \textit{self} \big[, (\textit{int}) \textit{index} 1 \big[, ... \big[, (\textit{int}) \textit{index} 8 \big] \big] \big]) \rightarrow \text{quantity}$ 

```
Gets the value of the variable at the specified domain indexes as the quantity object (with value and units).
      How many arguments index1, ..., index8 are used depends on the number of domains that the variable
      is distributed on.
SetValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (float)value) \rightarrow None
      Sets the value of the variable at the specified domain indexes. How many arguments index1, ..., index8
      are used depends on the number of domains that the variable is distributed on.
SetValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (quantity)value) \rightarrow None
      Sets the value of the variable at the specified domain indexes. How many arguments index1, ..., index8
      are used depends on the number of domains that the variable is distributed on.
SetValues ((daeVariable)self, (float)values) \rightarrow None
      Sets all values of the variable.
SetValues ((daeVariable)self, (quantity)values) \rightarrow None
      Sets all values of the variable.
AssignValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]]], (float)value) <math>\rightarrow None
AssignValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (quantity)value) \rightarrow None
AssignValues ((daeVariable)self, (float)values) \rightarrow None
AssignValues ((daeVariable)self, (quantity)values) \rightarrow None
\textbf{ReAssignValue} \ ((\textit{daeVariable}) \textit{self} \big[, (\textit{int}) \textit{index} 1 \big[, ... \big[, (\textit{int}) \textit{index} 8 \big] \big] \big], (\textit{float}) \textit{value}) \ \rightarrow \ \text{None}
ReAssignValue ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (<math>quantity)value) \rightarrow None
ReAssignValues ((daeVariable)self, (float)values) \rightarrow None
ReAssignValues ((daeVariable)self, (quantity)values) \rightarrow None
SetInitialCondition ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (float)initialCondition) \rightarrow
SetInitialCondition ((daeVariable)self[, (int)index1[, ...[, (int)index8]]]], (quantity)initialCondition)
SetInitialConditions ((daeVariable)self, (float)initialConditions) <math>\rightarrow None
SetInitialConditions ((daeVariable)self, (quantity)initialConditions) <math>\rightarrow None
ReSetInitialCondition ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (float)initialCondition)
                                                                             ...[,
                                                             (int)index1,
                                                                                         (int)index8 ] ] ],
ReSetInitialCondition ((daeVariable)self |,
                                                                                                                (quan-
                                   tity)initialCondition) \rightarrow None
ReSetInitialConditions ((daeVariable)self, (float)initialConditions) <math>\rightarrow None
ReSetInitialConditions ((daeVariable)self, (quantity)initialConditions) \rightarrow None
SetInitialGuess ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (float)initialGuess) <math>\rightarrow None
SetInitialGuess ((daeVariable)self[, (int)index1[, ...[, (int)index8]]], (quantity)initialGuess) \rightarrow None
\textbf{SetInitialGuesses} ((\textit{daeVariable}) \textit{self}, (\textit{float}) \textit{initialGuesses}) \rightarrow \textbf{None}
SetInitialGuesses ((daeVariable)self, (quantity)initialGuesses) <math>\rightarrow None
\textbf{SetAbsoluteTolerances} \ ((\textit{daeVariable}) \textit{self}, (\textit{float}) tolerances) \ \rightarrow \textbf{None}
array((daeVariable)self[, (object)index1[, ...[, (object)index8]]]) \rightarrow adouble_array
      Gets the array of variable's values at the specified domain indexes (used to build equation residuals only). How
      many arguments index1, ..., index8 are used depends on the number of domains that the variable is
      distributed on. Argument types are the same as those described in pyCore.daeParameter.array()
d_{array}((daeVariable)self[, (object)index1[, ...[, (object)index8]]]) \rightarrow adouble_array
      Gets the array of partial derivatives at the specified domain indexes (used to build equation residuals only). How
      many arguments index1, ..., index8 are used depends on the number of domains that the variable is
      distributed on. Argument types are the same as those described in pyCore.daeParameter.array().
```

```
\texttt{d2\_array} \ ((\textit{daeVariable}) \textit{self} \big[, (\textit{object}) \textit{index1} \big[, ... \big[, (\textit{object}) \textit{index8} \big] \big] \big]) \ \rightarrow \ \text{adouble\_array}
            Gets the array of partial derivatives of the second order at the specified domain indexes (used to build
            equation residuals only). How many arguments index1, ..., index8 are used depends on the num-
           ber of domains that the variable is distributed on. Argument types are the same as those described in
           pyCore.daeParameter.array().
      dt_array((daeVariable)self[, (object)index1[, ...[, (object)index8]]]) \rightarrow adouble_array
            Gets the array of time derivatives at the specified domain indexes (used to build equation residuals only). How
            many arguments index1, ..., index8 are used depends on the number of domains that the variable is
            distributed on. Argument types are the same as those described in pyCore.daeParameter.array().
        \_\mathtt{call}\_\_((daeVariable)self[, (int)index1], ...[, (int)index8]]]) \rightarrow adouble
            Gets the value of the variable at the specified domain indexes (used to build equation residuals only). How many
            arguments index1, ..., index8 are used depends on the number of domains that the variable is distributed on.
      d((daeVariable)self, (daeDomain)domain[, (int)index1[, ...[, (int)index8]]]) \rightarrow adouble
            Gets the partial derivative of the variable at the specified domain indexes (used to build equation residuals only).
            How many arguments index1, ..., index8 are used depends on the number of domains that the variable is
            distributed on.
      d2 ((daeVariable)self, (daeDomain)domain[, (int)index1[, ...[, (int)index8]]]) \rightarrow adouble
            Gets the partial derivative of second order of the variable at the specified domain indexes (used to build equation
            residuals only). How many arguments index1, ..., index8 are used depends on the number of domains that
           the variable is distributed on.
      dt ((daeVariable)self[, (int)index1[, ...[, (int)index8]]]) \rightarrow adouble
            Gets the time derivative of the variable at the specified domain indexes (used to build equation residuals only).
            How many arguments index1, ..., index8 are used depends on the number of domains that the variable is
            distributed on.
      \texttt{DistributeOnDomain} ((daeVariable)self, (daeDomain)domain) \rightarrow None
      GetDomainsIndexesMap((daeVariable)self, (int)indexBase) \rightarrow dict
      NumberOfPoints
      OverallIndex
      ReportingOn
      VariableType
      npyIDs
      npyValues
class pyCore.daeModel
      Bases: pyCore.daeObject
      Base model class.
         init ((object)self, (str)name[, (daeModel)parentModel=0[, (str)description='']]) \rightarrow None:
           Constructor...
      ComponentArrays
            A list of arrays of components in the model.
      Components
            A list of components in the model.
      ConnectEventPorts ((daeModel) self, (daeEventPort) portFrom, (daeEventPort) portTo) \rightarrow None:
           Connects two event ports.
      ConnectPorts ((daeModel)self, (daePort)portFrom, (daePort)portTo) \rightarrow None:
            Connects two ports.
      CreateEquation ((daeModel)self, (str)name[, (str)description='\(^1\), (float)scaling=1.0]]) \rightarrow daeEquation
           Creates a new equation. Used to add equations to models or states in state transition networks
      DeclareEquations ((daeModel)self) \rightarrow None:
```

User-defined function where all model equations ans state transition networks are declared. Must be always implemented in derived classes.

DeclareEquations( (daeModel)self) -> None

#### Domains

A list of domains in the model.

#### **ELSE** $((daeModel)self) \rightarrow None:$

Adds the last state to a reversible state transition network.

 $\textbf{ELSE\_IF} ((\textit{daeModel}) \textit{self}, (\textit{daeCondition}) \textit{condition} \big[, (\textit{float}) \textit{eventTolerance} = 0.0 \big]) \rightarrow \textit{None} :$ 

Adds a new state to a reversible state transition network.

### **END\_IF** $((daeModel)self) \rightarrow None$ :

Finalises a reversible state transition network.

```
END_STN ((daeModel)self) \rightarrow None:
```

# Equations

A list of equations in the model.

### **EventPorts**

A list of event ports in the model.

```
 \begin{array}{ll} \textbf{Export} \; ((daeModel)self, & (str)content, \\ & text)modelExportContext) \; \to \; \text{None} \; : \end{array} \; (daeModelLanguage) language, \\ & (daeModelExportContext) \; \to \; \text{None} \; : \end{array}
```

 $\textbf{ExportObjects}. ((\textit{daeModel}) \textit{self}, (\textit{list}) \textit{objects}, (\textit{daeeModelLanguage}) \\ \rightarrow \textit{str}:$ 

 $\textbf{IF} \; ((\textit{daeModel}) \textit{self}, (\textit{daeCondition}) \textit{condition} \big[, (\textit{float}) \textit{eventTolerance} = 0.0 \, \big]) \; \rightarrow \; \text{None} \; :$ 

Creates a reversible state transition network and adds the first state.

#### InitialConditionMode

A mode used to calculate initial conditions ...

### IsModelDynamic

Boolean flag that determines whether the model is synamic or steady-state.

### ModelType

A type of the model ().

```
 \begin{array}{ll} \textbf{ON\_CONDITION} \ ((daeModel)self, & (daeCondition)condition[, & (list)switchToStates=[][, \\ & (list)setVariableValues=[][, & (list)triggerEvents=[][, \\ & (float)eventTolerance=0.0]]]]]]]) \rightarrow \textbf{None}: \\ \end{array}
```

# OnConditionActions

A list of OnCondition actions in the model.

# OnEventActions

A list of OnEvent actions in the model.

### Parameters

A list of parameters in the model.

### PortArrays

A list of arrays of ports in the model.

#### PortConnections

A list of port connections in the model.

#### Ports

A list of ports in the model.

**STATE** ((daeModel)self, (str)stateName)  $\rightarrow$  daeState :

38

```
STN ((daeModel)self, (str)stnName) \rightarrow daeSTN :
     STNs
           A list of state transition networks in the model.
     SWITCH_TO ((daeModel)self, (str)targetState, (daeCondition)condition[, (float)eventTolerance=0.0]) \rightarrow
     SaveModelReport ((daeModel)self, (str)xmlFilename) \rightarrow None:
     SaveRuntimeModelReport ((daeModel)self, (str)xmlFilename) \rightarrow None:
     SetReportingOn ((daeModel)self, (bool)reportingOn) \rightarrow None:
          Switches the reporting of the model variables/parameters to the data reporter on or off.
     Variables
           A list of variables in the model.
class pyCore.daeSTN
     Bases: pyCore.daeObject
     ActiveState
     States
class pyCore.daeIF
     Bases: pyCore.daeSTN
class pyCore.daeEquation
     Bases: pyCore.daeObject
     \textbf{DistributeOnDomain} \ ((dae Equation) self, \ (dae Domain) domain, \ (dae eDomain Bounds) domain Bounds) \ \rightarrow
           DistributeOnDomain( (daeEquation)self, (daeDomain)domain, (list)domainIndexes) -> daeDEDI
     DistributedEquationDomainInfos
     EquationExecutionInfos
     EquationType
     Residual
     Scaling
class pyCore.daeState
     Bases: pyCore.daeObject
     Equations
     NestedSTNs
     OnConditionActions
     OnEventActions
class pyCore.daePort
     Bases: pyCore.daeObject
       ullet initullet ((object)self, (str)name, (daeePortType)type, (daeModel)parentModeligl[, (str)description=''igr]) 	o
     Domains
     Export ((daePort)self, (str)content, (daeeModelLanguage)language, (daeModelExportContext)context) <math>\rightarrow
               None
     Parameters
     SetReportingOn ((daePort)self, (bool)reportingOn) \rightarrow None
     Type
     Variables
```

```
class pyCore.daeEventPort
      Bases: pyCore.daeObject
        \verb|_init__((object)self, (str) name, (daeePortType) type, (daeModel) parentModel[, (str) description='`|) <math>
ightarrow
     EventData
     Events
     ReceiveEvent ((daeEventPort)self, (float)data) \rightarrow None
     RecordEvents
     SendEvent ((daeEventPort)self, (float)data) \rightarrow None
      Type
class pyCore.daePortConnection
     Bases: pyCore.daeObject
     Equations
     PortFrom
     PortTo
class pyCore.daeScalarExternalFunction
     Bases: Boost.Python.instance
      \_init\_((object)self, (str)name, (daeModel)parentModel, (object)units, (dict)arguments) \rightarrow None
       \_call\_((daeScalarExternalFunction)self) \rightarrow adouble
      Calculate ((daeScalarExternalFunction)arg1, (tuple)self, (dict)values) \rightarrow object
           Calculate((daeScalarExternalFunction)arg1, (tuple)arg2, (dict)arg3) -> None
     Name
class pyCore.daeVectorExternalFunction
     Bases: Boost.Python.instance
        _init___((object)self,
                                 (str)name,
                                                (daeModel)parentModel,
                                                                            (object)units,
                                                                                             (int)numberOfXXX,
                  (dict)arguments) \rightarrow None
       __call__((daeVectorExternalFunction)self) \rightarrow adouble_array
      Calculate ((daeVectorExternalFunction)arg1, (tuple)self, (dict)values) \rightarrow list
           Calculate( (daeVectorExternalFunction)arg1, (tuple)arg2, (dict)arg3) -> None
     Name
class pyCore.daeDomainIndex
     Bases: Boost.Python.instance
       _{\tt init} ((object)self, (int)index) \rightarrow None
           __init__( (object)self, (daeDEDI)dedi) -> None
           __init__( (object)self, (daeDEDI)dedi, (int)increment) -> None
           __init__( (object)self, (daeDomainIndex)domainIndex) -> None
     DEDI
      Increment
      Index
      Type
class pyCore.daeIndexRange
     Bases: Boost.Python.instance
       \_init\_((object)self, (daeDomain)domain) \rightarrow None
           __init__( (object)arg1, (daeDomain)arg2, (list)arg3) -> object
           __init__( (object)self, (daeDomain)domain, (int)startIndex, (int)endIndex, (int)step) -> None
     Domain
```

```
EndIndex
     NoPoints
     StartIndex
     Step
     Type
class pyCore.daeArrayRange
     Bases: Boost.Python.instance
       \_init\_((object)self, (daeDomainIndex)domainIndex) \rightarrow None
           __init__( (object)self, (daeIndexRange)indexRange) -> None
     DomainIndex
     NoPoints
     Range
     Type
class pyCore.daeDEDI
     Bases: pyCore.daeObject
     ___init___()
          Raises an exception This class cannot be instantiated from Python
     \_\_call\_\_((daeDEDI)self) \rightarrow adouble
     Domain
     DomainBounds
     DomainPoints
class pyCore.daeAction
     Bases: pyCore.daeObject
      \underline{\hspace{0.5cm}} init \underline{\hspace{0.5cm}} ((object)self) \rightarrow None
     Execute ((daeAction)self) \rightarrow None
          Execute( (daeAction)arg1) -> None
     RuntimeNode
     STN
     SendEventPort
     SetupNode
     StateTo
     Type
     VariableWrapper
class pyCore.daeOnEventActions
     Bases: pyCore.daeObject
     Actions
     EventPort
     Execute ((daeOnEventActions)arg1) \rightarrow None
     UserDefinedActions
class pyCore.daeOnConditionActions
     Bases: pyCore.daeObject
     Actions
     Condition
     \textbf{Execute} ((\textit{daeOnConditionActions}) \textit{arg1}) \rightarrow \textit{None}
```

# UserDefinedActions class pyCore.daeOptimizationVariable Bases: pyCore.daeOptimizationVariable\_t $\underline{\hspace{1cm}}$ init $\underline{\hspace{1cm}}$ ((object)self) $\rightarrow$ None LowerBound Name StartingPoint Type UpperBound Value class pyCore.daeObjectiveFunction Bases: $pyCore.daeObjectiveFunction\_t$ $\underline{\hspace{0.5cm}}$ init $\underline{\hspace{0.5cm}}$ $((object)self) \rightarrow None$ Gradients Name Residual Value class pyCore.daeOptimizationConstraint Bases: $pyCore.daeOptimizationConstraint_t$ $\underline{\hspace{0.5cm}}$ init $\underline{\hspace{0.5cm}}$ ((object)self) $\rightarrow$ None Gradients Name Residual Type Value class pyCore.daeMeasuredVariable Bases: pyCore.daeMeasuredVariable\_t $\underline{\hspace{0.5cm}}$ init $\underline{\hspace{0.5cm}}$ ((object)self) $\rightarrow$ None Gradients Name Residual Value class pyCore.daeEquationExecutionInfo Bases: Boost.Python.instance EquationType Node VariableIndexes **Functions** d dt Time Continued on next page

Table 7.2 – continued from previous page

Constant	Constant( (object)value) -> adouble
Array	
Sum	
Product	
Integral	
Average	

```
pyCore.d((adouble)arg1, (daeDomain)ad) → adouble
pyCore.dt((adouble)ad) → adouble
pyCore.Time() → adouble

pyCore.Constant((float)value) → adouble
    Constant((object)value) → adouble
    pyCore.Array((list)values) → adouble_array
pyCore.Sum((adouble_array)adarray) → adouble
pyCore.Product((adouble_array)adarray) → adouble
pyCore.Integral((adouble_array)adarray) → adouble
pyCore.Average((adouble_array)adarray) → adouble
```

# 7.1.3 Autodifferentiation and equation evaluation tree support

#### **Classes**

adouble	Class adouble operates on values/derivatives of domains, parameters and variables.	
adouble_array	Class adouble_array operates on arrays of values/derivatives of domains, parameters and variables.	
daeCondition		

# class pyCore.adouble

Bases: Boost.Python.instance

```
\underline{\quad \text{init}}_{\underline{\quad}}((object)self[, \quad (float)value=0.0[, \quad (float)derivative=0.0[, \quad (bool)gatherInfo=False[, \quad (adNode)node=0]]]]]) \rightarrow \text{None}
```

### Derivative

Derivative

### GatherInfo

Internally used by the framework.

### Node

Contains the equation evaluation node.

### Value

Value

### class pyCore.adouble\_array

Bases: Boost.Python.instance

Class adouble\_array operates on arrays of values/derivatives of domains, parameters and variables. It supports basic mathematical operators (+, -, , /, \*). Operands can be instances of adouble\_array, adouble or float values.

 $\_$ init $\_$ ((object)self[, (bool)gatherInfo=False[, (adNodeArray)node=0]])  $\rightarrow$  None  $\_$ len $\_$ ((adouble\_array)self)  $\rightarrow$  int:

Returns the size of the adouble\_array object.

```
\_\_getitem\_\_((adouble\_array)self, (int)index) <math>\rightarrow adouble:
           Gets an adouble object at the specified index.
        \_setitem\_((adouble_array)self, (int)index, (adouble)value) \rightarrow None:
           Sets an adouble object at the specified index.
      GatherInfo
           Used internally by the framework.
     Node
           Contains the equation evaluation node.
     Resize ((adouble\_array)self, (int)newSize) \rightarrow None:
           Resizes the adouble_array object to the new size.
      items ((object)arg1) \rightarrow object:
           Returns an iterator over adouble items in adouble_array object.
class pyCore.daeCondition
      Bases: Boost.Python.instance
       \_or\_((daeCondition)self, (daeCondition)right) \rightarrow daeCondition
     Logical operator or
      __and__((daeCondition)self, (daeCondition)right) <math>\rightarrow daeCondition
     Logical operator and
     EventTolerance
     Expressions
     RuntimeNode
      SetupNode
```

# **Mathematical functions**

Ехр	Exp( (adouble_array)arg1) -> adouble_array
Log	Log( (adouble_array)arg1) -> adouble_array
Log10	Log10( (adouble_array)arg1) -> adouble_array
Sqrt	Sqrt( (adouble_array)arg1) -> adouble_array
Sin	Sin( (adouble_array)arg1) -> adouble_array
Cos	Cos( (adouble_array)arg1) -> adouble_array
Tan	Tan( (adouble_array)arg1) -> adouble_array
ASin	ASin( (adouble_array)arg1) -> adouble_array
ACos	ACos( (adouble_array)arg1) -> adouble_array
ATan	ATan( (adouble_array)arg1) -> adouble_array
Sinh	
Cosh	
Tanh	
ASinh	
ACosh	
ATanh	
ATan2	
Ceil	Ceil( (adouble_array)arg1) -> adouble_array
Floor	Floor( (adouble_array)arg1) -> adouble_array
Pow	Pow( (adouble)arg1, (adouble)arg2) -> adouble
Abs	Abs( (adouble_array)arg1) -> adouble_array
Min	Min( (float)arg1, (adouble)arg2) -> adouble
Max	Max( (float)arg1, (adouble)arg2) -> adouble

 $\label{eq:core_abs} \text{pyCore.} \ \textbf{Exp} \ ((adouble)arg1) \ \to \ adouble \\ \text{Exp} \ ((adouble\_array)arg1) \ -> \ adouble\_array$ 

```
pyCore.Log((adouble)arg1) \rightarrow adouble
      Log( (adouble_array)arg1) -> adouble_array
pyCore.Log10 ((adouble)arg1) \rightarrow adouble
      Log10( (adouble_array)arg1) -> adouble_array
pyCore. Sqrt ((adouble)arg1) \rightarrow adouble
      Sqrt( (adouble array)arg1) -> adouble array
pyCore.Sin((adouble)arg1) \rightarrow adouble
      Sin( (adouble_array)arg1) -> adouble_array
pyCore.Cos ((adouble)arg1) \rightarrow adouble
      Cos( (adouble_array)arg1) -> adouble_array
pyCore. Tan ((adouble)arg1) \rightarrow adouble
      Tan( (adouble_array)arg1) -> adouble_array
pyCore.ASin ((adouble)arg1) \rightarrow adouble
      ASin( (adouble_array)arg1) -> adouble_array
pyCore. ACos((adouble)arg1) \rightarrow adouble
      ACos( (adouble_array)arg1) -> adouble_array
pyCore.ATan ((adouble)arg1) \rightarrow adouble
      ATan( (adouble_array)arg1) -> adouble_array
pyCore. Sinh ((adouble)arg1) \rightarrow adouble
pyCore.Cosh((adouble)arg1) \rightarrow adouble
pyCore. Tanh ((adouble)arg1) \rightarrow adouble
pyCore.ASinh ((adouble)arg1) \rightarrow adouble
pyCore. ACosh((adouble)arg1) \rightarrow adouble
pyCore.ATanh ((adouble)arg1) \rightarrow adouble
pyCore.ATan2 ((adouble)arg1, (adouble)arg2) \rightarrow adouble
pyCore.Ceil ((adouble)arg1) \rightarrow adouble
      Ceil( (adouble_array)arg1) -> adouble_array
pyCore.Floor ((adouble)arg1) \rightarrow adouble
      Floor((adouble_array)arg1) -> adouble_array
pyCore.Pow((adouble)arg1, (float)arg2) \rightarrow adouble
      Pow( (adouble)arg1, (adouble)arg2) -> adouble
      Pow((float)arg1, (adouble)arg2) -> adouble
pyCore.Abs ((adouble)arg1) \rightarrow adouble
      Abs( (adouble_array)arg1) -> adouble_array
pyCore.Min ((adouble)arg1, (adouble)arg2) \rightarrow adouble
      Min((float)arg1, (adouble)arg2) -> adouble
      Min( (adouble)arg1, (float)arg2) -> adouble
      Min( (adouble_array)adarray) -> adouble
pyCore. Max ((adouble)arg1, (adouble)arg2) \rightarrow adouble
      Max( (float)arg1, (adouble)arg2) -> adouble
      Max( (adouble)arg1, (float)arg2) -> adouble
      Max( (adouble_array)adarray) -> adouble
```

# 7.1.4 Auxiliary classes

daeVariableWrapper

Continued on next page

# Table 7.5 – continued from previous page

daeConfig

```
class pyCore.daeVariableWrapper
      Bases: Boost.Python.instance
        \_init\_((object)self, (daeVariable)variable[, (str)name='']) <math>\rightarrow None
           __init__( (object)self, (adouble)ad [, (str)name='']) -> None
      DomainIndexes
      Name
      OverallIndex
      Value
      Variable
      VariableType
class pyCore.daeConfig
      Bases: Boost.Python.instance
      \_contains\_((daeConfig)self, (object)propertyPath) \rightarrow object
      __getitem__((daeConfig)self, (object)propertyPath) <math>\rightarrow object
       __setitem__((daeConfig)self, (object)propertyPath, (object)value) \rightarrow None
      GetBoolean ((daeConfig)self, (str)propertyPath[, (bool)defaultValue]) \rightarrow bool
      GetFloat((daeConfig)self, (str)propertyPath[, (float)defaultValue]) \rightarrow float
      GetInteger ((daeConfig)self, (str)propertyPath[, (int)defaultValue]) \rightarrow int
      GetString ((daeConfig)self, (str)propertyPath[, (str)defaultValue]) \rightarrow str
      Reload((daeConfig)self) \rightarrow None
      SetBoolean ((daeConfig)self, (str)propertyPath, (bool)value) \rightarrow None
      SetFloat ((daeConfig)self, (str)propertyPath, (float)value) \rightarrow None
      SetInteger ((daeConfig)self, (str)propertyPath, (int)value) \rightarrow None
      SetString ((daeConfig)self, (str)propertyPath, (str)value) \rightarrow None
      has\_key((daeConfig)self, (object)propertyPath) \rightarrow object
```

# 7.1.5 Auxiliary functions

daeGetConfig
daeVersion
daeVersionMajor
daeVersionMinor
daeVersionBuild

```
\label{eq:pycore.daeGetConfig} \begin{split} &\operatorname{pyCore.daeGetConfig}() \to \operatorname{object} \\ &\operatorname{pyCore.daeVersion}(\big[ (bool) includeBuild = False \big]) \to \operatorname{str} \\ &\operatorname{pyCore.daeVersionMajor}() \to \operatorname{int} \\ &\operatorname{pyCore.daeVersionBuild}() \to \operatorname{int} \end{split}
```

# 7.1.6 Enumerations

daeeDomainType
daeeParameterType
daeePortType
daeeDiscretizationMethod
daeeDomainBounds
daeeInitialConditionMode
daeeDomainIndexType
daeeRangeType
daeIndexRangeType
daeeOptimizationVariableType
daeeModelLanguage
daeeConstraintType
daeeUnaryFunctions
daeeBinaryFunctions
daeeSpecialUnaryFunctions
daeeLogicalUnaryOperator
daeeLogicalBinaryOperator
daeeConditionType
daeeActionType
daeeEquationType
daeeModelType

```
class pyCore.daeeDomainType
     Bases: Boost.Python.enum
     eArray = pyCore.daeeDomainType.eArray
     eDTUnknown = pyCore.daeeDomainType.eDTUnknown
     eDistributed = pyCore.daeeDomainType.eDistributed
class pyCore.daeeParameterType
    Bases: Boost.Python.enum
     eBool = pyCore.daeeParameterType.eBool
     eInteger = pyCore.daeeParameterType.eInteger
     ePTUnknown = pyCore.daeeParameterType.ePTUnknown
     eReal = pyCore.daeeParameterType.eReal
class pyCore.daeePortType
    Bases: Boost.Python.enum
     eInletPort = pyCore.daeePortType.eInletPort
     eOutletPort = pyCore.daeePortType.eOutletPort
     eUnknownPort = pyCore.daeePortType.eUnknownPort
class pyCore.daeeDiscretizationMethod
     Bases: Boost.Python.enum
     eBFDM = pyCore.daeeDiscretizationMethod.eBFDM
     eCFDM = pyCore.daeeDiscretizationMethod.eCFDM
     eCustomDM = pyCore.daeeDiscretizationMethod.eCustomDM
     eDMUnknown = pyCore.daeeDiscretizationMethod.eDMUnknown
     eFFDM = pyCore.daeeDiscretizationMethod.eFFDM
class pyCore.daeeDomainBounds
    Bases: Boost.Python.enum
     eClosedClosed = pyCore.daeeDomainBounds.eClosedClosed
     eClosedOpen = pyCore.daeeDomainBounds.eClosedOpen
```

```
eDBUnknown = pyCore.daeeDomainBounds.eDBUnknown
    eLowerBound = pyCore.daeeDomainBounds.eLowerBound
    eOpenClosed = pyCore.daeeDomainBounds.eOpenClosed
    eOpenOpen = pyCore.daeeDomainBounds.eOpenOpen
    eUpperBound = pyCore.daeeDomainBounds.eUpperBound
class pyCore.daeeInitialConditionMode
    Bases: Boost.Python.enum
    eAlgebraicValuesProvided = pyCore.daeeInitialConditionMode.eAlgebraicValuesProvided
    eDifferentialValuesProvided = pyCore.daeeInitialConditionMode.eDifferentialValuesProvided
    eICTUnknown = pyCore.daeeInitialConditionMode.eICTUnknown
    eQuasySteadyState = pyCore.daeeInitialConditionMode.eQuasySteadyState
class pyCore.daeeDomainIndexType
    Bases: Boost.Python.enum
    eConstantIndex = pyCore.daeeDomainIndexType.eConstantIndex
    eDITUnknown = pyCore.daeeDomainIndexType.eDITUnknown
    eDomainIterator = pyCore.daeeDomainIndexType.eDomainIterator
    eIncrementedDomainIterator = pyCore.daeeDomainIndexType.eIncrementedDomainIterator
class pyCore.daeeRangeType
    Bases: Boost.Python.enum
    eRaTUnknown = pyCore.daeeRangeType.eRaTUnknown
    eRange = pyCore.daeeRangeType.eRange
    eRangeDomainIndex = pyCore.daeeRangeType.eRangeDomainIndex
class pyCore.daeIndexRangeType
    Bases: Boost.Python.enum
    eAllPointsInDomain = pyCore.daeIndexRangeType.eAllPointsInDomain
    eCustomRange = pyCore.daeIndexRangeType.eCustomRange
    eIRTUnknown = pyCore.daeIndexRangeType.eIRTUnknown
    eRangeOfIndexes = pyCore.daeIndexRangeType.eRangeOfIndexes
class pyCore.daeeOptimizationVariableType
    Bases: Boost.Python.enum
    eBinaryVariable = pyCore.daeeOptimizationVariableType.eBinaryVariable
    eContinuousVariable = pyCore.daeeOptimizationVariableType.eContinuousVariable
    eIntegerVariable = pyCore.daeeOptimizationVariableType.eIntegerVariable
class pyCore.daeeModelLanguage
    Bases: Boost.Python.enum
    eCDAE = pyCore.daeeModelLanguage.eCDAE
    eMLNone = pyCore.daeeModelLanguage.eMLNone
    ePYDAE = pyCore.daeeModelLanguage.ePYDAE
class pyCore.daeeConstraintType
    Bases: Boost.Python.enum
    eEqualityConstraint = pyCore.daeeConstraintType.eEqualityConstraint
    eInequalityConstraint = pyCore.daeeConstraintType.eInequalityConstraint
class pyCore.daeeUnaryFunctions
    Bases: Boost.Python.enum
```

```
eAbs = pyCore.daeeUnaryFunctions.eAbs
     eArcCos = pyCore.daeeUnaryFunctions.eArcCos
     eArcSin = pyCore.daeeUnaryFunctions.eArcSin
     eArcTan = pyCore.daeeUnaryFunctions.eArcTan
     eCeil = pyCore.daeeUnaryFunctions.eCeil
     eCos = pyCore.daeeUnaryFunctions.eCos
     eExp = pyCore.daeeUnaryFunctions.eExp
     eFloor = pyCore.daeeUnaryFunctions.eFloor
     eLn = pyCore.daeeUnaryFunctions.eLn
     eLog = pyCore.daeeUnaryFunctions.eLog
     eSign = pyCore.daeeUnaryFunctions.eSign
     eSin = pyCore.daeeUnaryFunctions.eSin
     eSqrt = pyCore.daeeUnaryFunctions.eSqrt
     eTan = pyCore.daeeUnaryFunctions.eTan
     eUFUnknown = pyCore.daeeUnaryFunctions.eUFUnknown
class pyCore.daeeBinaryFunctions
     Bases: Boost.Python.enum
     eBFUnknown = pyCore.daeeBinaryFunctions.eBFUnknown
     eDivide = pyCore.daeeBinaryFunctions.eDivide
     eMax = pyCore.daeeBinaryFunctions.eMax
     eMin = pyCore.daeeBinaryFunctions.eMin
     eMinus = pyCore.daeeBinaryFunctions.eMinus
     eMulti = pyCore.daeeBinaryFunctions.eMulti
     ePlus = pyCore.daeeBinaryFunctions.ePlus
     ePower = pyCore.daeeBinaryFunctions.ePower
class pyCore.daeeSpecialUnaryFunctions
     Bases: Boost.Python.enum
     eAverage = pyCore.daeeSpecialUnaryFunctions.eAverage
     eMaxInArray = pyCore.daeeSpecialUnaryFunctions.eMaxInArray
     eMinInArray = pyCore.daeeSpecialUnaryFunctions.eMinInArray
     eProduct = pyCore.daeeSpecialUnaryFunctions.eProduct
     eSUFUnknown = pyCore.daeeSpecialUnaryFunctions.eSUFUnknown
     eSum = pyCore.daeeSpecialUnaryFunctions.eSum
class pyCore.daeeLogicalUnaryOperator
     Bases: Boost.Python.enum
     eNot = pyCore.daeeLogicalUnaryOperator.eNot
     eUOUnknown = pyCore.daeeLogicalUnaryOperator.eUOUnknown
class pyCore.daeeLogicalBinaryOperator
     Bases: Boost.Python.enum
     eAnd = pyCore.daeeLogicalBinaryOperator.eAnd
     eBOUnknown = pyCore.daeeLogicalBinaryOperator.eBOUnknown
     eOr = pyCore.daeeLogicalBinaryOperator.eOr
```

```
class pyCore.daeeConditionType
     Bases: Boost.Python.enum
     eCTUnknown = pyCore.daeeConditionType.eCTUnknown
     eEQ = pyCore.daeeConditionType.eEQ
     eGT = pyCore.daeeConditionType.eGT
     eGTEQ = pyCore.daeeConditionType.eGTEQ
     eLT = pyCore.daeeConditionType.eLT
     eLTEQ = pyCore.daeeConditionType.eLTEQ
     eNotEQ = pyCore.daeeConditionType.eNotEQ
class pyCore.daeeActionType
     Bases: Boost.Python.enum
     eChangeState = pyCore.daeeActionType.eChangeState
     eReAssignOrReInitializeVariable = pyCore.daeeActionType.eReAssignOrReInitializeVariable
     eSendEvent = pyCore.daeeActionType.eSendEvent
     eUnknownAction = pyCore.daeeActionType.eUnknownAction
     eUserDefinedAction = pyCore.daeeActionType.eUserDefinedAction
class pyCore.daeeEquationType
    Bases: Boost.Python.enum
     eAlgebraic = pyCore.daeeEquationType.eAlgebraic
     eETUnknown = pyCore.daeeEquationType.eETUnknown
     eExplicitODE = pyCore.daeeEquationType.eExplicitODE
     eImplicitODE = pyCore.daeeEquationType.eImplicitODE
class pyCore.daeeModelType
     Bases: Boost.Python.enum
     eDAE = pyCore.daeeModelType.eDAE
     eMTUnknown = pyCore.daeeModelType.eMTUnknown
     eODE = pyCore.daeeModelType.eODE
     eSteadyState = pyCore.daeeModelType.eSteadyState
```

### 7.1.7 Global constants

cnAlgebraic	int(x[, base]) -> integer
cnDifferential	int(x[, base]) -> integer
cnAssigned	int(x[, base]) -> integer

```
pyCore.cnAlgebraic = 0
int(x[, base]) -> integer
```

Convert a string or number to an integer, if possible. A floating point argument will be truncated towards zero (this does not include a string representation of a floating point number!) When converting a string, use the optional base. It is an error to supply a base when converting a non-string. If base is zero, the proper base is guessed based on the string content. If the argument is outside the integer range a long object will be returned instead.

```
pyCore.cnDifferential = 1
int(x[, base]) -> integer
```

Convert a string or number to an integer, if possible. A floating point argument will be truncated towards zero (this does not include a string representation of a floating point number!) When converting a string, use the optional base. It is an error to supply a base when converting a non-string. If base is zero, the proper base is guessed based on the

string content. If the argument is outside the integer range a long object will be returned instead.

```
pyCore.cnAssigned = 2 int(x[, base]) -> integer
```

Convert a string or number to an integer, if possible. A floating point argument will be truncated towards zero (this does not include a string representation of a floating point number!) When converting a string, use the optional base. It is an error to supply a base when converting a non-string. If base is zero, the proper base is guessed based on the string content. If the argument is outside the integer range a long object will be returned instead.

# 7.2 Message logs

## 7.2.1 Overview

### Classes

```
daeLog_t
daeBaseLog
daeDelegateLog
daeFileLog
daeStdOutLog
daeTCPIPLog
daeTCPIPLogServer
```

```
class pyCore.daeLog_t
      Bases: Boost.Python.instance
          init ()
            Raises an exception This class cannot be instantiated from Python
      DecreaseIndent ((daeLog\_t)self, (int)offset) \rightarrow None
            DecreaseIndent( (daeLog_t)arg1, (int)arg2) -> None
      ETA
      Enabled
      IncreaseIndent((daeLog\_t)self,(int)offset) \rightarrow None
            IncreaseIndent( (daeLog_t)arg1, (int)arg2) -> None
      Indent
      IndentString
      JoinMessages ((daeLog\_t)self[, (str)delimiter='n']) \rightarrow str
            JoinMessages( (daeLog_t)arg1, (str)arg2) -> None
      \textbf{Message} ((\textit{daeLog\_t}) \textit{self}, (\textit{str}) \textit{message}, (\textit{int}) \textit{severity}) \rightarrow \textit{None}
            Message( (daeLog_t)arg1, (str)arg2, (int)arg3) -> None
      PercentageDone
      PrintProgress
      Progress
class pyCore.daeBaseLog
      Bases: pyCore.daeLog_t
      \underline{\hspace{1cm}}init\underline{\hspace{1cm}} ((object)self) \rightarrow None
      DecreaseIndent ((daeBaseLog)self, (int)offset) \rightarrow None
      IncreaseIndent ((daeBaseLog)self, (int)offset) \rightarrow None
      Message ((daeBaseLog)self, (str)message, (int)severity) \rightarrow None
            Message( (daeBaseLog)self, (str)message, (int)severity) -> None
```

7.2. Message logs 51

```
SetProgress ((daeBaseLog)self, (float)progress) \rightarrow None
           SetProgress( (daeBaseLog)self, (float)progress) -> None
class pyCore.daeDelegateLog
     Bases: pyCore.daeBaseLog
      \_init\_((object)self) \rightarrow None
     AddLog ((daeDelegateLog)self, (daeLog_t)log) \rightarrow None
     Logs
     Message ((daeDelegateLog)self, (str)message, (int)severity) \rightarrow None
           Message((daeDelegateLog)self, (str)message, (int)severity) -> None
class pyCore.daeFileLog
      Bases: pyCore.daeBaseLog
      \_init\_((object)self, (str)filename) \rightarrow None
     Message ((daeFileLog)self, (str)message, (int)severity) \rightarrow None
           Message( (daeFileLog)self, (str)message, (int)severity) -> None
class pyCore.daeStdOutLog
     Bases: pyCore.daeBaseLog
     \_init\_((object)self) \rightarrow None
     Message ((daeStdOutLog)self, (str)message, (int)severity) \rightarrow None
           Message( (daeStdOutLog)self, (str)message, (int)severity) -> None
class pyCore.daeTCPIPLog
     Bases: pyCore.daeBaseLog
     \_init\_((object)self) \rightarrow None
      Connect ((daeTCPIPLog)self, (str)tcpipAddress, (int)port) \rightarrow bool
     Disconnect((daeTCPIPLog)self) \rightarrow bool
      IsConnected((daeTCPIPLog)self) \rightarrow bool
     Message ((daeTCPIPLog)self, (str)message, (int)severity) \rightarrow None
           Message( (daeTCPIPLog)self, (str)message, (int)severity) -> None
class pyCore.daeTCPIPLogServer
      Bases: Boost.Python.instance
      \_init\_((object)self, (int)port) \rightarrow None
     MessageReceived ((daeTCPIPLogServer)self, (str)message) \rightarrow None
           MessageReceived( (daeTCPIPLogServer)self, (str)message) -> None
     Port
      Start((daeTCPIPLogServer)self) \rightarrow None
      \textbf{Stop} ((\textit{daeTCPIPLogServer}) self) \rightarrow \text{None}
Extra logs
{\bf class} \; {\tt daetools.pyDAE.logs.daePythonStdOutLog}
     Bases: pyCore.daeStdOutLog
     Message (message, severity)
class daetools.dae_simulator.simulator.daeTextEditLog(TextEdit, ProgressBar, ProgressLabel,
                                                                          App)
      Bases: pyCore.daeBaseLog
     Message (message, severity)
      SetProgress (progress)
```

# 7.3 Module pyActivity

# 7.3.1 Overview

# 7.3.2 Classes

daeSimulation
daeOptimization

```
class pyActivity.daeSimulation
      Bases: pyActivity.daeSimulation_t
      Initialization methods
      \underline{\hspace{0.5cm}} init \underline{\hspace{0.5cm}} ((object)self) \rightarrow None
                                                                                                               (object)log[,
      Initialize ((daeSimulation)self,
                                                     (object)daeSolver,
                                                                                (object)dataReporter,
                        (bool)calculateSensitivities=False ) \rightarrow None
      SolveInitial((daeSimulation)self) \rightarrow None
      model
      Model
      DAESolver
      Log
      DataReporter
      AbsoluteTolerances
      RelativeTolerance
      TotalNumberOfVariables
      NumberOfEquations
      Loading/storing the initialization data
      \textbf{LoadInitializationValues} ((\textit{daeSimulation}) \textit{self}, (\textit{str}) \textit{filename}) \rightarrow \textit{None}
      StoreInitializationValues ((daeSimulation)self, (str)filename) \rightarrow None
      InitialValues
      InitialDerivatives
      Clean up methods
      CleanUpSetupData((daeSimulation)self) \rightarrow None
      \textbf{Finalize} ((\textit{daeSimulation}) \textit{self}) \ \rightarrow None
      Simulation setup methods
      \texttt{SetUpParametersAndDomains}((daeSimulation)self) \rightarrow \texttt{None}
```

 $\textbf{SetUpVariables} ((\textit{daeSimulation}) self) \rightarrow \textbf{None}$ 

```
Optimization setup methods
SetUpOptimization ((daeSimulation)self) \rightarrow None
CreateInequalityConstraint ((daeSimulation)self, (str)description) \rightarrow object
CreateEqualityConstraint ((daeSimulation)self, (str)description) \rightarrow object
SetContinuousOptimizationVariable ((daeSimulation)self, (object)variable, (float)lowerBound,
                                                   (float)upperBound, (float)defaultValue) \rightarrow object
      SetContinuousOptimizationVariable( (daeSimulation)self, (object)ad, (float)lowerBound, (float)upperBound,
      (float)defaultValue) -> object
SetIntegerOptimizationVariable ((daeSimulation)self,
                                                                        (object)variable,
                                                                                               (int)lowerBound,
                                               (int)upperBound, (int)defaultValue) \rightarrow object
      SetIntegerOptimizationVariable(
                                         (daeSimulation)self,
                                                                 (object)ad,
                                                                                (int)lowerBound,
                                                                                                     (int)upperBound,
      (int)defaultValue) -> object
SetBinaryOptimizationVariable ((daeSimulation)self, (object)variable, (bool)defaultValue) \rightarrow ob-
      SetBinaryOptimizationVariable((daeSimulation)self, (object)ad, (bool)defaultValue) -> object
OptimizationVariables
Constraints
ObjectiveFunction
Parameter estimation setup methods
\textbf{SetUpParameterEstimation} \ ((\textit{daeSimulation}) self) \ \rightarrow \ None
SetMeasuredVariable ((daeSimulation)self, (object)variable) \rightarrow object
     SetMeasuredVariable((daeSimulation)self, (object)ad) -> object
\textbf{SetInputVariable} \ ((\textit{daeSimulation}) \textit{self}, (\textit{object}) \textit{variable}) \ \rightarrow \text{object}
     SetInputVariable( (daeSimulation)self, (object)ad) -> object
SetModelParameter ((daeSimulation)self,
                                                  (object)variable,
                                                                       (float)lowerBound,
                                                                                             (float)upperBound,
                           (float)defaultValue) \rightarrow object
      SetModelParameter((daeSimulation)self, (object)ad, (float)lowerBound, (float)upperBound, (float)defaultValue)
      -> object
InputVariables
MeasuredVariables
ModelParameters
Parameter estimation setup methods
SetUpSensitivityAnalysis ((daeSimulation)self) \rightarrow None
Operating procedures methods
Run ((daeSimulation)self) \rightarrow None
ReRun ((daeSimulation)self) \rightarrow None
Pause ((daeSimulation)self) \rightarrow None
Resume ((daeSimulation)self) \rightarrow None
ActivityAction
\textbf{Integrate} \ ((dae Simulation) self, (dae e Stop Criterion) stop Criterion \ [\ , (bool) report Data Around Discontinuities = True) \ ]
               \rightarrow float
IntegrateForTimeInterval ((daeSimulation)self,
                                                                                            (float)timeInterval,
                                     (bool)reportDataAroundDiscontinuities=True ) \rightarrow float
```

```
IntegrateUntilTime ((daeSimulation)self,
                                                           (float)time,
                                                                              (daeeStopCriterion)stopCriterion,
                                (bool)reportDataAroundDiscontinuities=True ) \rightarrow float
     Reinitialize ((daeSimulation)self) \rightarrow None
     Reset ((daeSimulation)self) \rightarrow None
      CurrentTime
      TimeHorizon
     ReportingInterval
     NextReportingTime
     ReportingTimes
      Data reporting methods
     RegisterData ((daeSimulation)self, (str)iteration) \rightarrow None
     ReportData ((daeSimulation)self, (float)currentTime) \rightarrow None
      Various information
      IndexMappings
      InitialConditionMode
      SimulationMode
     VariableTypes
      LastSatisfiedCondition
class pyActivity.daeOptimization
      Bases: pyActivity.daeOptimization_t
      \underline{\hspace{1cm}}init\underline{\hspace{1cm}} ((object)self) \rightarrow None
      Initialize ((daeOptimization)self, (daeSimulation_t)simulation, (object)nlpSolver, (object)daeSolver, (ob-
                     ject)dataReporter, (object)log) \rightarrow None
     Run ((daeOptimization)self) \rightarrow None
     Finalize ((daeOptimization)self) \rightarrow None
7.3.3 Enumerations
                                     daeeStopCriterion
                                     daeeActivityAction
                                     daeeSimulationMode
```

# class pyActivity.daeeStopCriterion

Bases: Boost.Python.enum

 ${\tt eDoNotStopAtDiscontinuity = pyActivity. daeeStopCriterion. eDoNotStopAtDiscontinuity}$ 

eStopAtModelDiscontinuity = pyActivity.daeeStopCriterion.eStopAtModelDiscontinuity

class pyActivity.daeeActivityAction

Bases: Boost.Python.enum

eAAUnknown = pyActivity.daeeActivityAction.eAAUnknown

ePauseActivity = pyActivity.daeeActivityAction.ePauseActivity

eRunActivity = pyActivity.daeeActivityAction.eRunActivity

### class pyActivity.daeeSimulationMode

Bases: Boost.Python.enum

eOptimization = pyActivity.daeeSimulationMode.eOptimization

eParameterEstimation = pyActivity.daeeSimulationMode.eParameterEstimation

eSimulation = pyActivity.daeeSimulationMode.eSimulation

# 7.4 Module pyDataReporting

# 7.4.1 Overview

# 7.4.2 DataReporter classes

daeDataReporter_t
daeDataReporterLocal
daeNoOpDataReporter
daeDataReporterFile
daeTEXTFileDataReporter
daeBlackHoleDataReporter
daeDelegateDataReporter

### class pyDataReporting.daeDataReporter\_t

Bases: Boost.Python.instance

**Connect**  $((daeDataReporter\_t)self, (str)connectionString, (str)processName) <math>\rightarrow$  bool

 $\textbf{Disconnect} ((daeDataReporter\_t)self) \rightarrow bool$ 

 $\textbf{IsConnected} \left( (\textit{daeDataReporter\_t}) self \right) \ \rightarrow \textbf{bool}$ 

 $StartRegistration((daeDataReporter\_t)self) \rightarrow bool$ 

 $RegisterDomain((daeDataReporter\_t)self, (daeDataReporterDomain)domain) \rightarrow bool$ 

 $RegisterVariable ((daeDataReporter\_t)self, (daeDataReporterVariable) variable) \rightarrow bool$ 

**EndRegistration**  $((daeDataReporter\_t)self) \rightarrow bool$ 

 $StartNewResultSet((daeDataReporter\_t)self, (Float)time) \rightarrow bool$ 

**SendVariable**  $((daeDataReporter\_t)self, (daeDataReporterVariableValue)variableValue) <math>\rightarrow$  bool

 $\textbf{EndOfData} ((\textit{daeDataReporter\_t}) self) \ \rightarrow \textbf{bool}$ 

# Data reporters that do not send data to a data receiver and keep data locally (local data reporters)

# class pyDataReporting.daeDataReporterLocal

 $Bases: \verb"pyDataReporting.daeDataReporter_t"$ 

**Process** 

dictDomains

dictVariables

# class pyDataReporting.daeNoOpDataReporter

Bases: pyDataReporting.daeDataReporterLocal

class pyDataReporting.daeDataReporterFile

Bases: pyDataReporting.daeDataReporterLocal

 $WriteDataToFile ((daeDataReporterFile)self) \rightarrow None$ 

class pyDataReporting.daeTEXTFileDataReporter

Bases: pyDataReporting.daeDataReporterFile

 $\textbf{WriteDataToFile} ((\textit{daeTEXTFileDataReporter}) self) \rightarrow None$ 

# Third-party local data reporters

```
daePlotDataReporter()
daeMatlabMATFileDataReporter()
```

class daetools.pyDAE.data\_reporters.daePlotDataReporter

Bases: pyDataReporting.daeDataReporterLocal

```
Plot (*args, **kwargs)
args can be either:
```

- 1.Instances of daeVariable, or
- 2.Lists of daeVariable instances, or
- 3.A mixture of both.

Each arg will get its own subplot. The subplots are all automatically arranged such that the resulting figure is as square-like as possible. You can however override the shape by supplying figRows and figCols as keyword args.

# Basic Example:

```
# Create Log, Solver, DataReporter and Simulation object
log = daePythonStdOutLog()
daesolve = daeIDAS()
from daetools.pyDAE.data_reporters import daePlotDataReporter
datareporter = daePlotDataReporter()
simulation = simTutorial()
simulation.m.SetReportingOn(True)
simulation.ReportingInterval = 20
simulation.TimeHorizon = 500
simName = simulation.m.Name + strftime(" [%d.%m.%Y %H:%M:%S]", localtime())
if(datareporter.Connect("", simName) == False):
    sys.exit()
simulation. Initialize (daesolver, datareporter, log)
simulation.m.SaveModelReport(simulation.m.Name + ".xml")
simulation.m.SaveRuntimeModelReport(simulation.m.Name + "-rt.xml")
simulation.SolveInitial()
simulation.Run()
simulation.Finalize()
datareporter.Plot(
    simulation.m.Ci,
                                           # Subplot 1
    [simulation.m.L, simulation.m.event], # Subplot 2 (2 sets)
    simulation.m.Vp,
                                            # Subplot 3
    [simulation.m.L, simulation.m.Vp]
                                           # Subplot 4 (2 sets)
    )
```

class daetools.pyDAE.data\_reporters.daeMatlabMATFileDataReporter

Bases: pyDataReporting.daeDataReporterLocal

```
WriteDataToFile()
```

# Data reporters that do send data to a data receiver (remote data reporters)

```
class pyDataReporting.daeDataReporterRemote
    Bases: pyDataReporting.daeDataReporter_t
```

```
SendMessage ((daeDataReporterRemote)self, (str)message) \rightarrow bool
```

### class pyDataReporting.daeTCPIPDataReporter

Bases: pyDataReporting.daeDataReporterRemote

**SendMessage**  $((daeTCPIPDataReporter)self, (str)message) \rightarrow bool$ 

### Special-purpose data reporters

### class pyDataReporting.daeBlackHoleDataReporter

Bases: pyDataReporting.daeDataReporter\_t

Data reporter that does not process any data and all function calls simply return True. Could be used when no results from the simulation are needed.

### class pyDataReporting.daeDelegateDataReporter

Bases: pyDataReporting.daeDataReporter\_t

A container-like data reporter, which does not process any data but forwards (delegates) all function calls (Disconnect(), IsConnected(), StartRegistration(), RegisterDomain(), RegisterVariable(), EndRegistration(), StartNewResultSet(), SendVariable(), EndOfData()) to data reporters in the containing list of data reporters. Data reporters can be added by using the AddDataReporter(). The list of containing data reporters is in the DataReporters attribute.

 $\label{local_connection} \textbf{Connect}((\textit{daeDataReporter\_t}) \textit{self}, (\textit{str}) \textit{connectionString}, (\textit{str}) \textit{processName}) \rightarrow \textbf{Boolean} \\ \textbf{Does nothing. Always returns True}.$ 

 $\textbf{AddDataReporter}((daeDelegateDataReporter) self, (daeDataReporter\_t) dataReporter) \rightarrow \textbf{None}$ 

DataReporters

### **DataReporter data-containers**

daeDataReporterDomain
daeDataReporterVariable
daeDataReporterVariableValue

# class pyDataReporting.daeDataReporterDomain

```
Bases: Boost.Python.instance
```

```
\_getitem\_((daeDataReporterDomain)self, (int)index) <math>\rightarrow float
```

 $\_$ **setitem** $\_((daeDataReporterDomain)self, (int)index, (float)value) <math>\rightarrow$  None

Name

NumberOfPoints

Points

Type

# class pyDataReporting.daeDataReporterVariable

Bases: Boost.Python.instance

 $\textbf{AddDomain} \ ((\textit{daeDataReporterVariable}) \textit{self}, (\textit{str}) \textit{domainName}) \ \rightarrow \ \textit{None}$ 

Domains

Name

NumberOfDomains

NumberOfPoints

# ${\bf class}~{\tt pyDataReporting.daeDataReporterVariableValue}$

Bases: Boost.Python.instance

**\_\_getitem\_\_**( $(daeDataReporterVariableValue)self, (int)index) <math>\rightarrow$  float

```
\underline{\hspace{1.5cm}} \textbf{setitem}\underline{\hspace{1.5cm}} ((\textit{daeDataReporterVariableValue}) \textit{self, (int)index, (float)value}) \ \rightarrow \ None
```

Name

NumberOfPoints

Values

### 7.4.3 DataReceiver classes

```
daeDataReceiver_t
daeTCPIPDataReceiver
daeTCPIPDataReceiverServer
```

```
class pyDataReporting.daeDataReceiver_t
```

Bases: Boost.Python.instance

 $Start((daeDataReceiver\_t)self) \rightarrow bool$ 

**Stop**  $((daeDataReceiver\_t)self) \rightarrow bool$ 

Process

### class pyDataReporting.daeTCPIPDataReceiver

Bases: pyDataReporting.daeDataReceiver\_t

 $\texttt{Start}((daeTCPIPDataReceiver)self) \rightarrow bool$ 

**Stop**  $((daeTCPIPDataReceiver)self) \rightarrow bool$ 

**Process** 

### class pyDataReporting.daeTCPIPDataReceiverServer

Bases: Boost.Python.instance

DataReceivers

 $\textbf{IsConnected}((\textit{daeTCPIPDataReceiverServer}) self) \rightarrow bool$ 

 $\texttt{Start}((daeTCPIPDataReceiverServer)self) \rightarrow \texttt{None}$ 

 $Stop((daeTCPIPDataReceiverServer)self) \rightarrow None$ 

### **DataReceiver data-containers**

daeDataReceiverDomain
daeDataReceiverVariable
daeDataReceiverVariableValue
daeDataReceiverProcess

### class pyDataReporting.daeDataReceiverDomain

Bases: Boost.Python.instance

```
__getitem__ ((daeDataReceiverDomain)self, (int)index) \rightarrow float
```

 $\_$ **setitem** $\_$ ((daeDataReceiverDomain)self, (int)index, (float)value)  $\rightarrow$  None

Name

NumberOfPoints

Points

Type

# ${\bf class}~{\tt pyDataReporting.daeDataReceiverVariable}$

Bases: Boost.Python.instance

 $\textbf{AddDomain} ((\textit{daeDataReceiverVariable}) self, (\textit{daeDataReceiverDomain}) domain) \rightarrow \text{None}$ 

```
AddVariableValue ((daeDataReceiverVariable)self, (daeDataReceiverVariableValue) variableValue)
                            None
     Domains
     Name
     NumberOfPoints
     TimeValues
     Values
class pyDataReporting.daeDataReceiverVariableValue
     Bases: Boost.Python.instance
       \_getitem\_ ((daeDataReceiverVariableValue)self, (int)index) \rightarrow float
      \_setitem\_ ((daeDataReceiverVariableValue)self, (int)index, (float)value) \rightarrow None
     Time
class pyDataReporting.daeDataReceiverProcess
     Bases: Boost.Python.instance
     Domains
     FindVariable ((daeDataReceiverProcess)self, (str)variableName) \rightarrow daeDataReceiverVariable
     Name
     \textbf{RegisterDomain} \ ((\textit{daeDataReceiverProcess}) self, \ (\textit{daeDataReceiverDomain}) \\ \textit{domain}) \rightarrow \textit{None}
     RegisterVariable ((daeDataReceiverProcess)self, (daeDataReceiverVariable) \rightarrow None
     Variables
     dictDomains
     dictVariables
7.5 Module pyIDAS
7.5.1 Overview
7.5.2 Classes
                                   daeDAESolver_t
                                   daeIDAS
class pyIDAS.daeDAESolver_t
     Bases: Boost.Python.instance
     InitialConditionMode
     Log
     Name
     NumberOfVariables
     RelativeTolerance
class pyIDAS.daeIDAS
     Bases: pyIDAS.daeDAESolver_t
     SaveMatrixAsXPM ((daeIDAS)self, (str)xpmFilename) \rightarrow None
     SetLASolver ((daeIDAS)self, (daeeIDALASolverType)laSolverType) <math>\rightarrow None
          SetLASolver( (daeIDAS)self, (object)laSolver) -> None
```

# 7.5.3 Enumerations

daeeIDALASolverType

### class pyIDAS.daeeIDALASolverType

```
Bases: Boost.Python.enum
```

 $\verb"eSundialsGMRES" = pyIDAS. dae eIDALAS olver Type. eSundials GMRES$ 

eSundialsLU = pyIDAS.daeeIDALASolverType.eSundialsLU

eSundialsLapack = pyIDAS.daeeIDALASolverType.eSundialsLapack

eThirdParty = pyIDAS.daeeIDALASolverType.eThirdParty

# 7.6 Module pyUnits

# 7.6.1 Overview

# 7.6.2 Classes

```
base_unit
unit
quantity
```

```
{f class} pyUnits.base_unit
```

```
Bases: Boost.Python.instance
                    __init___((object)arg1) \rightarrow None
                                __init__( (object)arg1, (float)arg2, (dict)arg3) -> object
                 \underline{\hspace{0.5cm}} mul\underline{\hspace{0.5cm}} ((base_unit)arg1, (base_unit)arg2) \rightarrow object
                               __mul__( (base_unit)arg1, (float)arg2) -> object
                 \_\_div\_\_((base\_unit)arg1, (base\_unit)arg2) \rightarrow object
                                __div__( (base_unit)arg1, (float)arg2) -> object
                 __pow__ ((base_unit)arg1, (float)arg2) \rightarrow object
                \underline{\hspace{0.5cm}} eq \underline{\hspace{0.5cm}} ((base_unit)arg1, (base_unit)arg2) \rightarrow object
                 \underline{\hspace{0.5cm}} \underline{\hspace{0.5cm}}
                С
                 Ι
                 L
               М
                N
                 0
                 Т
                multiplier
class pyUnits.unit
                Bases: Boost.Python.instance
                 \_init\_((object)arg1) \rightarrow None
                                __init__( (object)arg1, (dict)arg2) -> object
                    _{\underline{\mathbf{mul}}_{\underline{\mathbf{u}}}}((unit)arg1,(unit)arg2) \rightarrow \text{object}
```

\_\_mul\_\_( (unit)arg1, (float)arg2) -> object

7.6. Module pyUnits 61

```
___div__ ((unit)arg1, (unit)arg2) \rightarrow object
                    __div__( (unit)arg1, (float)arg2) -> object
             _pow__ ((unit)arg1, (float)arg2) \rightarrow object
          \_eq\_((unit)arg1, (unit)arg2) <math>\rightarrow object
           \underline{\hspace{0.1cm}} \underline{\hspace{0.1cm}}
          baseUnit
          unitDictionary
class pyUnits.quantity
          Bases: Boost.Python.instance
          \underline{\hspace{0.5cm}} init\underline{\hspace{0.5cm}} ((object)arg1) \rightarrow None
                   __init__( (object)self, (float)value, (unit)unit) -> None
            \underline{\hspace{0.1cm}} neg \underline{\hspace{0.1cm}} ((quantity)arg1) \rightarrow object
           __pos__ ((quantity)arg1) \rightarrow object
             _add_ ((quantity)arg1, (quantity)arg2) \rightarrow object
                    __add__( (quantity)arg1, (float)arg2) -> object
           __sub__ ((quantity)arg1, (quantity)arg2) \rightarrow object
                   __sub__( (quantity)arg1, (float)arg2) -> object
          \underline{\hspace{1cm}} mul\underline{\hspace{1cm}} ((quantity)arg1, (quantity)arg2) \rightarrow object
                   __mul__( (quantity)arg1, (unit)arg2) -> object
                    __mul__( (quantity)arg1, (float)arg2) -> object
          \_\_div\_\_((quantity)arg1, (quantity)arg2) \rightarrow object
                   __div__( (quantity)arg1, (unit)arg2) -> object
                    __div__( (quantity)arg1, (float)arg2) -> object
           __pow__ ((quantity)arg1, (quantity)arg2) \rightarrow object
                    __pow__( (quantity)arg1, (float)arg2) -> object
             _eq_ ((quantity)arg1, (quantity)arg2) \rightarrow object
                   __eq__( (quantity)arg1, (float)arg2) -> object
                    __eq__( (quantity)arg1, (float)arg2) -> object
             _{\tt ne}_{\tt ((quantity)arg1, (quantity)arg2)} \rightarrow {\sf object}
                   __ne__( (quantity)arg1, (float)arg2) -> object
                    __ne__( (quantity)arg1, (float)arg2) -> object
          \__{1t}_{(quantity)arg1, (quantity)arg2)} \rightarrow object
                   __lt__( (quantity)arg1, (float)arg2) -> object
                    __lt__( (quantity)arg1, (float)arg2) -> object
            __le__ ((quantity)arg1, (quantity)arg2) \rightarrow object
                   __le__( (quantity)arg1, (float)arg2) -> object
                    __le__( (quantity)arg1, (float)arg2) -> object
          \__{\tt gt} ((quantity)arg1, (quantity)arg2) \rightarrow object
                    __gt__( (quantity)arg1, (float)arg2) -> object
                    __gt__( (quantity)arg1, (float)arg2) -> object
          __ge__((quantity)arg1, (quantity)arg2) \rightarrow object
                   __ge__( (quantity)arg1, (float)arg2) -> object
                    __ge__( (quantity)arg1, (float)arg2) -> object
          scaleTo((quantity)self, (object)referrer) \rightarrow quantity
          units
          value
```

valueInSIUnits

# 7.7 Variable types

# 7.7.1 Overview

variable\_types.time\_t = daeVariableType(name=time\_t, units=[s], lowerBound=0, upperBound=1e+20, initialGuess=0, al variable\_types.length\_t = daeVariableType(name=length\_t, units=[m], lowerBound=0, upperBound=100000, initialGue variable\_types.area\_t = daeVariableType(name=area\_t, units=[m^2], lowerBound=0, upperBound=100000, initialGuess= variable\_types.volume\_t = daeVariableType(name=volume\_t, units=[m^3], lowerBound=0, upperBound=100000, initial( variable\_types.velocity\_t = daeVariableType(name=velocity\_t, units=[m/s], lowerBound=-1e+10, upperBound=1e+10, variable\_types.pressure\_t = daeVariableType(name=pressure\_t, units=[Pa], lowerBound=100, upperBound=1e+10, init variable\_types.temperature\_t = daeVariableType(name=temperature\_t, units=[K], lowerBound=0, upperBound=10000 variable\_types.fraction\_t = daeVariableType(name=fraction\_t, units=[1], lowerBound=-1e-10, upperBound=1.1, initial variable\_types.no\_t = daeVariableType(name=no\_t, units=[1], lowerBound=-1e+20, upperBound=1e+20, initialGuess=0, a variable\_types.moles\_t = daeVariableType(name=moles\_t, units=[mol], lowerBound=0, upperBound=1e+20, initialGuess variable\_types.molar\_flux\_t = daeVariableType(name=molar\_flux\_t, units=[mol/m^2], lowerBound=-1e+20, upperBou variable\_types.molar\_concentration\_t = daeVariableType(name=molar\_concentration\_t, units=[mol/m^3], lowerBo variable\_types.molar\_flowrate\_t = daeVariableType(name=molar\_flowrate\_t, units=[mol/s], lowerBound=-1e+10, upj variable\_types.heat\_t = daeVariableType(name=heat\_t, units=[J], lowerBound=-1e+20, upperBound=1e+20, initialGuess variable\_types.heat\_flux\_t = daeVariableType(name=heat\_flux\_t, units=[W/m^2], lowerBound=-1e+20, upperBound= variable\_types.heat\_transfer\_coefficient\_t = daeVariableType(name=heat\_transfer\_coefficient\_t, units=[W/(Kir variable\_types.power\_t = daeVariableType(name=power\_t, units=[W], lowerBound=-1e+20, upperBound=1e+20, initialC variable\_types.specific\_heat\_capacity\_t = daeVariableType(name=specific\_heat\_capacity\_t, units=[J/(K kg)], low variable\_types.density\_t = daeVariableType(name=density\_t, units=[kg/m^3], lowerBound=0, upperBound=1e+20, init variable\_types.specific\_heat\_conductivity\_t = daeVariableType(name=specific\_heat\_conductivity\_t, units=[W/( variable\_types.dynamic\_viscosity\_t = daeVariableType(name=dynamic\_viscosity\_t, units=[Pa s], lowerBound=0, up variable\_types.diffusivity\_t = daeVariableType(name=diffusivity\_t, units=[m^2/s], lowerBound=0, upperBound=100 variable\_types.amount\_adsorbed\_t = daeVariableType(name=amount\_adsorbed\_t, units=[mol/kg], lowerBound=-1e+2

# 7.8 Third party solvers

# 7.8.1 Linear solvers

 ${f class}$  pyCore. ${f daeIDALASolver\_t}$ 

Name

**SaveAsXPM** ( $(daeIDALASolver\_t)self$ ,  $(str)xpmFilename) \rightarrow int$ 

### **SuperLU**

### Instantiation function

 $\texttt{pySuperLU.daeCreateSuperLUSolver()} \rightarrow daeIDALASolver\_t$ 

7.7. Variable types 63

#### Classes

```
class pySuperLU.daeSuperLU_Solver
    Bases: pySuperLU.daeIDALASolver_t
    Options
    SaveAsMatrixMarketFile ((daeSuperLU_Solver)self, (str)filename, (str)matrixName, (str)description)
class pySuperLU.superlu_options_t
    Bases: Boost.Python.instance
    ColPerm
    DiagPivotThresh
    PrintStat
    RowPerm
Enumerations
class pySuperLU.IterRefine_t
    Bases: Boost.Python.enum
    DOUBLE = pySuperLU.IterRefine_t.DOUBLE
    EXTRA = pySuperLU.IterRefine_t.EXTRA
    NOREFINE = pySuperLU.IterRefine t.NOREFINE
    SINGLE = pySuperLU.IterRefine_t.SINGLE
class pySuperLU.rowperm_t
    Bases: Boost.Python.enum
    LargeDiag = pySuperLU.rowperm_t.LargeDiag
    MY_PERMR = pySuperLU.rowperm_t.MY_PERMR
    NOROWPERM = pySuperLU.rowperm_t.NOROWPERM
class pySuperLU.yes_no_t
    Bases: Boost.Python.enum
    NO = pySuperLU.yes_no_t.NO
    YES = pySuperLU.yes_no_t.YES
class pySuperLU.colperm_t
    Bases: Boost.Python.enum
    COLAMD = pySuperLU.colperm_t.COLAMD
    METIS_AT_PLUS_A = pySuperLU.colperm_t.METIS_AT_PLUS_A
    MMD_ATA = pySuperLU.colperm_t.MMD_ATA
    MMD_AT_PLUS_A = pySuperLU.colperm_t.MMD_AT_PLUS_A
    NATURAL = pySuperLU.colperm_t.NATURAL
SuperLU MT
Instantiation function
\texttt{pySuperLU\_MT.daeCreateSuperLUSolver()} \rightarrow daeIDALASolver\_t
```

#### **Classes**

```
class pySuperLU_MT.daeSuperLU_MT_Solver
     Bases: pySuperLU_MT.daeIDALASolver_t
     Options
     SaveAsMatrixMarketFile ((daeSuperLU_MT_Solver)self,
                                                                (str)filename,
                                                                                 (str)matrixName,
                                (str)description) \rightarrow int
class pySuperLU_MT.superlumt_options_t
     Bases: Boost.Python.instance
     ColPerm
    PrintStat
    diag_pivot_thresh
     drop_tol
    nprocs
    panel size
     relax
Enumerations
class pySuperLU_MT.yes_no_t
     Bases: Boost.Python.enum
    NO = pySuperLU_MT.yes_no_t.NO
     YES = pySuperLU_MT.yes_no_t.YES
class pySuperLU_MT.colperm_t
     Bases: Boost.Python.enum
     COLAMD = pySuperLU_MT.colperm_t.COLAMD
    METIS_AT_PLUS_A = pySuperLU_MT.colperm_t.METIS_AT_PLUS_A
    MMD_ATA = pySuperLU_MT.colperm_t.MMD_ATA
    \label{eq:mmd_at_plus_a} \mathbf{MMD\_AT\_PLUS\_A} = pySuperLU\_MT.colperm\_t.MMD\_AT\_PLUS\_A
    NATURAL = pySuperLU_MT.colperm_t.NATURAL
Trilinos
Instantiation function
pyTrilinos.daeTrilinosSupportedSolvers() \rightarrow list
pyTrilinos.daeCreateTrilinosSolver((str)solverName, (str)preconditionerName) \rightarrow daeIDALA-
                                          Solver_t
Classes
class pyTrilinos.daeTrilinosSolver
     Bases: pyTrilinos.daeIDALASolver_t
    AmesosOptions
    Aztec000ptions
     IfpackOptions
     MLOptions
```

```
NumIters
     PreconditionerName
     PrintPreconditionerInfo((daeTrilinosSolver)self) \rightarrow None
      SaveAsMatrixMarketFile ((daeTrilinosSolver)self, (str)filename, (str)matrixName, (str)description) \rightarrow
      Tolerance
class pyTrilinos. Teuchos Parameter List
      Bases: Boost.Python.instance
     Print((TeuchosParameterList)self) \rightarrow None
      get\_bool((TeuchosParameterList)self,(str)name) \rightarrow bool
      get_float((TeuchosParameterList)self,(str)name) \rightarrow float
      get\_int((TeuchosParameterList)self,(str)name) \rightarrow int
      \texttt{get\_string}((TeuchosParameterList)self, (str)name) \rightarrow \mathsf{str}
      set\_bool((TeuchosParameterList)self,(str)name,(bool)value) \rightarrow None
      set_float((TeuchosParameterList)self,(str)name,(float)value) \rightarrow None
      set_int((TeuchosParameterList)self,(str)name,(int)value) \rightarrow None
      set\_string((TeuchosParameterList)self,(str)name,(str)value) \rightarrow None
7.8.2 Optimization solvers
class pyCore.daeIDALASolver_t
     Name
      Initialize((daeIDALASolver_t)self, (daeSimulation_t)simulation, (daeDAESolver_t)daeSolver, (dae-
                     DataReporter\_t)dataReporter, (daeLog\_t)log) \rightarrow None
      Solve ((daeIDALASolver\_t)self) \rightarrow None
class pyIPOPT.daeIPOPT
      Bases: pyIPOPT.daeNLPSolver_t
     ClearOptions ((daeIPOPT)self) \rightarrow None
     LoadOptionsFile ((daeIPOPT)self, (str)optionsFilename) \rightarrow None
     PrintOptions ((daeIPOPT)self) \rightarrow None
     PrintUserOptions ((daeIPOPT)self) \rightarrow None
      SetOption ((daeIPOPT)self, (str)name, (str)value) \rightarrow None
           SetOption( (daeIPOPT)self, (str)name, (float)value) -> None
           SetOption( (daeIPOPT)self, (str)name, (int)value) -> None
class pyBONMIN.daeBONMIN
      Bases: pyBONMIN.daeNLPSolver_t
      ClearOptions ((daeBONMIN)self) \rightarrow None
     LoadOptionsFile ((daeBONMIN)self, (str)optionsFilename) \rightarrow None
     PrintOptions ((daeBONMIN)self) \rightarrow None
     PrintUserOptions ((daeBONMIN)self) \rightarrow None
      SetOption ((daeBONMIN)self, (str)name, (str)value) \rightarrow None
           SetOption( (daeBONMIN)self, (str)name, (float)value) -> None
           SetOption( (daeBONMIN)self, (str)name, (int)value) -> None
```

```
class pyNLOPT.daeNLOPT
    Bases: pyNLOPT.daeNLPSolver_t
    PrintOptions ((daeNLOPT)self) → None
    ftol_abs
    ftol_rel
    xtol_abs
    xtol_rel
```

#### 7.8.3 Parameter estimation solvers

```
class daetools.solvers.minpack.daeMinpackLeastSq

Finalize()
Initialize(simulation, daesolver, datareporter, log, **kwargs)
Run()
getConfidenceCoefficient(confidence)
getConfidenceEllipsoid(x_param_index, y_param_index, **kwargs)
getFit_Dyn(measured_variable_index, experiment_index, **kwargs)
getFit_SS(input_variable_index, measured_variable_index, **kwargs)
```

## 7.9 Code generators

## 7.9.1 Auxiliary classes

```
{\bf class} \; {\tt daetools.code\_generators.analyzer.daeCodeGeneratorAnalyzer}
     Bases: object
     analyzeModel (model)
     analyzePort (port)
     analyzeSimulation (simulation)
class daetools.code_generators.formatter.daeExpressionFormatter
     Bases: object
     flattenIdentifier (identifier)
     formatDomain (domainCanonicalName, index, value)
     formatIdentifier (identifier)
     formatNumpyArray(arr)
     formatParameter (parameterCanonicalName, domainIndexes, value)
     formatQuantity (quantity)
     formatRuntimeConditionNode (node)
     formatRuntimeNode (node)
     formatTimeDerivative (variableCanonicalName, domainIndexes, overallIndex, order)
     formatUnits (units)
     formatVariable (variableCanonicalName, domainIndexes, overallIndex)
```

7.9. Code generators 67

#### 7.9.2 Modelica

```
class daetools.code_generators.modelica.daeModelicaExpressionFormatter
    Bases: daetools.code_generators.formatter.daeExpressionFormatter
    formatNumpyArray (arr)
    formatQuantity (quantity)
    formatUnits (units)
class daetools.code_generators.modelica.daeCodeGenerator_Modelica (simulation=None)
    Bases: object
    generateModel (model, filename=None)
    generatePort (port, filename=None)
    generateSimulation (simulation, filename=None)
7.9.3 ANSI C
class daetools.code_generators.ansi_c.daeANSICExpressionFormatter
    Bases: daetools.code_generators.formatter.daeExpressionFormatter
    formatNumpyArray (arr)
    formatQuantity (quantity)
class daetools.code_generators.ansi_c.daeCodeGenerator_ANSI_C (simulation=None)
    Bases: object
    generateSimulation (simulation, **kwargs)
7.9.4 Functional Mockup Interface (FMI)
class daetools.code_generators.fmi.daeCodeGenerator_FMI
    Bases: daetools.code_generators.fmi_xml_support.fmiModelDescription
    generateSimulation (simulation, **kwargs)
7.10 DAE Tools Plotter
```

#### 7.10.1 Overview

#### 7.10.2 Classes

```
class daetools.dae_plotter.plotter.daeMainWindow(tcpipServer)
    Bases: PyQt4.QtGui.QMainWindow
    plot2D (updateInterval=0)
    slotAbout()
    slotAnimatedPlot2D()
    slotDocumentation()
    slotPlot2D()
    slotPlot3D()
daetools.dae_plotter.plotter.daeStartPlotter(port=0)
class daetools.dae_plotter.plot2d.dae2DPlot (parent, tcpipServer, updateInterval=0)
    Bases: PyQt4.QtGui.QDialog
    addLine (xAxisLabel, yAxisLabel, xPoints, yPoints, domains)
```

```
closeEvent (event)
    newCurve()
    plotDefaults = [<daetools.dae_plotter.plot2d.daePlot2dDefaults instance at 0x53ac8c0>, <daetools.dae_plotter.plot2d.d
     reformatPlot()
     slotExportCSV()
     slotProperties()
     slotRemoveLine()
    slotToggleGrid()
     slotToggleLegend()
     slotViewTabularData()
    updateCurves()
class daetools.dae_plotter.plot2d.daePlot2dDefaults(color='black',
                                                                                  linewidth=0.5,
                                                            linestyle='solid', marker='o', mark-
                                                            ersize=6,
                                                                        markerfacecolor='black',
                                                            markeredgecolor='black')
class daetools.dae_plotter.mayavi_plot3d.daeMayavi3DPlot(tcpipServer)
    newSurface()
\verb|daetools.dae_plotter.plot_options.col2hex| (color)
     Convert matplotlib color to hex
daetools.dae_plotter.plot_options.figure_edit(canvas, parent=None)
    Edit matplotlib figure options
daetools.dae_plotter.plot_options.surface_edit(canvas, parent=None)
    Edit matplotlib figure options
```

#### 7.11 DAE Tools Simulator

#### 7.11.1 Overview

#### 7.11.2 Classes

```
class daetools.dae_simulator.simulator.daeSimulator(app, **kwargs)
    Bases: PyQt4.QtGui.QDialog
    done(status)
    laAmdACML = 10
    laAmesos_Klu = 3
    laAmesos_Lapack = 6
    laAmesos_Superlu = 4
    laAmesos_Umfpack = 5
    laAztecOO = 7
    laCUSP = 14
    laIntelMKL = 9
    laIntelPardiso = 8
    laLapack = 11
    laMagmaLapack = 12
```

```
laSundialsLU = 0
laSuperLU = 1
laSuperLU_CUDA = 13
laSuperLU_MT = 2
nlpBONMIN = 2
nlpIPOPT = 0
nlpNLOPT = 1
showMessage (msg)
slotExportSparseMatrixAsMatrixMarketFormat()
slotOpenSparseMatrixImage()
slotPause()
slotResume()
slotRun()
```

## **TUTORIALS**

**Note**: Currently only Mozilla Firefox and Opera 12+ browsers are supported for viewing model reports (MathML rendering issue).

## 8.1 What's the time? (AKA: Hello world!)

#### **Description**

What is the time? (AKA Hello world!) is a very simple simulation. It shows the basic structure of the model and the simulation classes. A typical simulation includes 8 basic tasks:

- 1. How to import the pyDAE module(s)
- 2. How to declare variable types daeVariableType
- 3. How to define a model by deriving a class from the base daeModel and which functions must be implemented by every model (daeModel.\_\_init\_\_, daeModel.DeclareEquations):
  - how to declare parameters and variables in daeModel.\_\_init\_\_ function
  - how to declare equations and their residuals in daeModel.DeclareEquations function
- 4. How to define a simulation by deriving a class from the base daeSimulation and which functions must be implemented by every simulation (daeSimulation.\_\_init\_\_, daeSimulation.SetUpParametersAndDomains, daeSimulation.SetUpVariables):
  - how to set specify a model to be used in simulation in daeSimulation.\_\_init\_\_ function
  - how to set values of parameters in daeSimulation.SetUpParametersAndDomains function
  - how to set initial conditions in daeSimulation.SetUpVariables' function
- 5. How to create auxiliary objects required for the simulation (DAE solver, data reporter and logging objects)
- 6. How to set simulation's additional settings
- 7. How to connect a data reporter
- 8. How to run a simulation

#### Files

Model report	whats_the_time.xml
Runtime model report	whats_the_time-rt.xml
Source code	whats_the_time.py

### 8.2 Tutorial 1

#### **Description**

This tutorial introduces several new concepts:

- Distribution domains
- Distributed parameters, variables and equations
- · Boundary and initial conditions

In this example we model a simple heat conduction problem: a conduction through a very thin, rectangular copper plate.

This example should be sufficiently complex to describe all basic DAE Tools features. For this problem, we need a two-dimensional Cartesian grid in X and Y axis (here, for simplicity, divided into 10 x 10 segments):

Points 'B' at the bottom edge of the plate (for y = 0), and the points 'T' at the top edge of the plate (for y = Ly) represent the points where the heat is applied.

The plate is considered insulated at the left (x = 0) and the right edges (x = Lx) of the plate (points 'L' and 'R'). To model this type of problem, we have to write a heat balance equation for all interior points except the left, right, top and bottom edges, where we need to define the Neumann type boundary conditions.

In this problem we have to define the following domains:

- x: X axis domain, length Lx = 0.1 m
- y: Y axis domain, length Ly = 0.1 m

the following parameters:

- ro: copper density, 8960 kg/m3
- cp: copper specific heat capacity, 385 J/(kgK)
- k: copper heat conductivity, 401 W/(mK)
- Qb: heat flux at the bottom edge of the plate, 1E6 W/m2 (or 100 W/cm2)
- Qt: heat flux at the top edge of the plate, here set to 0 W/m2

and the following variable:

• T: the temperature of the plate, K (distributed on x and y domains)

Also, we need to write the following 5 equations:

1. Heat balance:

```
ro * cp * dT(x,y) / dt = k * (d2T(x,y) / dx2 + d2T(x,y) / dy2); for all x in: (0, Lx), for all y in: (0, Ly)
```

2. Boundary conditions for the bottom edge:

72 Chapter 8. Tutorials

$$-k * dT(x,y) / dy = Qin;$$
 for all x in: [0, Lx], and  $y = 0$ 

3. Boundary conditions for the top edge:

```
-k * dT(x,y) / dy = Qin; for all x in: [0, Lx], and y = Ly
```

4. Boundary conditions for the left edge:

$$\label{eq:dt_dt_dt} \begin{array}{lll} dT\left(x,y\right) \ / \ dx \ = \ 0; & \mbox{for all y in: (0, Ly),} \\ & \mbox{and } x \ = \ 0 \end{array}$$

5. Boundary conditions for the right edge:

```
dT(x,y) / dx = 0; for all y in: (0, Ly),
and x = Ln
```

#### **Files**

Model report	tutorial1.xml
Runtime model report	tutorial1-rt.xml
Source code	tutorial1.py

## 8.3 Tutorial 2

#### **Description**

In this example we use the same conduction problem as in the tutorial 1.

Here we introduce:

- Arrays (discrete distribution domains)
- Distributed parameters
- Number of degrees of freedom and how to fix it
- Initial guess of the variables

#### Files

Model report	tutorial2.xml
Runtime model report	tutorial2-rt.xml
Source code	tutorial2.py

## 8.4 Tutorial 3

#### **Description**

In this example we use the same conduction problem as in the tutorial 1.

Here we introduce:

- · Arrays of variable values
- Functions that operate on arrays of values
- Functions that create constants and arrays of constant values (Constant and Array)

8.3. Tutorial 2 73

• Non-uniform domain grids

#### **Files**

Model report	tutorial3.xml
Runtime model report	tutorial3-rt.xml
Source code	tutorial3.py

## 8.5 Tutorial 4

#### **Description**

In this example we model a very simmple conduction problem where a piece of copper (a plate) is at one side exposed to the source of heat and at the other to the surroundings.

Here we introduce:

• Discontinuous equations (symmetrical state transition networks: daeIF statements)

Here we have a very simple heat balance:

```
ro * cp * dT/dt - Qin = h * A * (T - Tsurr)
```

The process starts at the temperature of the metal of 283K. The metal is allowed to warm up for 200 seconds and then the heat source is removed and the metal cools down slowly to the ambient temperature.

#### **Files**

Model report	tutorial4.xml
Runtime model report	tutorial4-rt.xml
Source code	tutorial4.py

## 8.6 Tutorial 5

#### **Description**

In this example we use the same conduction problem as in the tutorial 4.

Here we introduce:

• Discontinuous equations (non-symmetrical state transition networks: daeSTN statements)

Again we have a piece of copper (a plate) is at one side exposed to the source of heat and at the other to the surroundings. The process starts at the temperature of 283K. The metal is allowed to warm up, and then its temperature is kept in the interval [320 - 340] for at 350 seconds. After 350s the heat source is removed and the metal cools down slowly again to the ambient temperature.

#### **Files**

Model report	tutorial5.xml
Runtime model report	tutorial5-rt.xml
Source code	tutorial5.py

74 Chapter 8. Tutorials

## 8.7 Tutorial 6

#### **Description**

This is the simple port demo.

Here we introduce:

- Ports
- Port connections
- Units (instances of other models)

A simple port type 'portSimple' is defined which contains only one variable 't'. Two models 'modPortIn' and 'modPortOut' are defined, each having one port of type 'portSimple'. The wrapper model 'modTutorial' instantiate these two models as its units and connects them by connecting their ports.

#### **Files**

Model report	tutorial6.xml
Runtime model report	tutorial6-rt.xml
Source code	tutorial6.py

## 8.8 Tutorial 7

#### **Description**

In this example we use the same conduction problem as in the tutorial 1.

Here we introduce:

- · Custom operating procedures
- · Resetting of degrees of freedom
- Resetting of initial conditions

Here the heat flux at the bottom edge is defined as a variable. In the simulation its value will be fixed and manipulated in the custom operating procedure.

#### **Files**

Model report	tutorial7.xml
Runtime model report	tutorial7-rt.xml
Source code	tutorial7.py

## 8.9 Tutorial 8

#### **Description**

In this example we use a similar problem as in the tutorial 5.

Here we introduce:

- Writting Matlab .MAT files (using daeMatlabMATFileDataReporter)
- · Custom data reporters

8.7. Tutorial 6 75

Some time it is not enough to send the result to daePlotter but it is desirable to export them in certain format for use in other programs. Here we show how the custom data reporter can be created. In this example the data reporter simply, after the simulation is finished, save the results into a plain text file. Obviously, the data can be exported to any format. Also some numpy functions that operate on numpy arrays can be used as well. In addition, a new type of data reporters (daeDelegateDataReporter) is presented. It has the same interface and the functionality like all data reporters. However, it does not do any data processing itself but calls the corresponding functions of data reporters which are added to it by using the function AddDataReporter. This way it is possible, at the same time, to send the results to the daePlotter and save them into a file (or process them in some other ways).

#### **Files**

Model report	tutorial8.xml
Runtime model report	tutorial8-rt.xml
Source code	tutorial8.py

## 8.10 Tutorial 9

#### **Description**

In this example we use the same conduction problem as in the tutorial 1.

Here we introduce:

• Third party linear equations solvers

Currently there are 3rd party linear equations solvers:

- SuperLU: sequential sparse direct solver defined in pySuperLU module (BSD licence)
- SuperLU\_MT: multi-threaded sparse direct solver defined in pySuperLU\_MT module (BSD licence)
- Trilinos: sequential sparse direct/iterative solver defined in pyTrilinos module (GNU Lesser GPL)
- IntelPardiso: multi-threaded sparse direct solver defined in pyIntelPardiso module (proprietary)

#### **Files**

Model report	tutorial9.xml
Runtime model report	tutorial9-rt.xml
Source code	tutorial9.py

#### 8.11 Tutorial 10

#### **Description**

In this example we use the same conduction problem as in the tutorial 1.

Here we introduce:

- daeModel functions d() and dt() which calculate time- or partial-derivative of an expression
- · Initialization files
- Domains which bounds depend on parameter values
- How to evaluate integrals of a function

#### **Files**

Model report	tutorial10.xml
Runtime model report	tutorial10-rt.xml
Source code	tutorial10.py

## 8.12 Tutorial 11

#### **Description**

This tutorial shows the use of Trilinos group of solvers: Amesos and AztecOO iterative linear equation solvers with different preconditioners (built-in AztecOO, Ifpack or ML) and corresponding linear solver options.

#### **ACHTUNG, ACHTUNG!!**

Iterative solvers are not fully working yet and this example is given just as a showcase and for preconditioner options experimenting purposes.

#### **Files**

Model report	tutorial11.xml
Runtime model report	tutorial11-rt.xml
Source code	tutorial11.py

## 8.13 Tutorial 12

#### **Description**

As of the version 1.1.1 the main linear algebraic equations solver is superLU.

It comes in three variaants:

• Sequential: superlu

• Multithreaded (OpenMP/posix threads): superlu\_MT

• CUDA GPU: superlu CUDA

The first two are available in daetools with the addition of a new port: superlu\_CUDA that works on computers with NVidia CUDA enabled video cards. However, the later is still in an early stage of the development.

In this example, usage and available options of superlu and superlu\_MT are explored.

#### **Files**

Model report	tutorial12.xml
Runtime model report	tutorial12-rt.xml
Source code	tutorial12.py

## **8.14 Tutorial 13**

#### **Description**

In this example we use the same problem as in the tutorial 5.

8.12. Tutorial 11 77

Here we introduce:

- The event ports
- ON\_CONDITION() function showing the new types of actions that can be executed during state transitions
- ON\_EVENT() function showing the new types of actions that can be executed when an event is triggered
- User defined actions

#### **Files**

Model report	tutorial13.xml
Runtime model report	tutorial13-rt.xml
Source code	tutorial13.py

## 8.15 Tutorial 14

#### **Description**

In this example we use the same conduction problem as in the tutorial 5.

Here we introduce the external functions concept that can handle and evaluate functions in external libraries. Here we use daeScalarExternalFunction class derived external function object to calculate the power.

A support for external functions is still experimental and the goal is to support certain software components such as thermodynamic property packages etc.

#### **Files**

Model report	tutorial14.xml
Runtime model report	tutorial14-rt.xml
Source code	tutorial14.py

## 8.16 Tutorial 15

#### **Description**

In this example we use the same problem as in the tutorial 4.

Here we introduce:

· Nested state transitions

#### **Files**

Model report	tutorial15.xml
Runtime model report	tutorial15-rt.xml
Source code	tutorial15.py

78 Chapter 8. Tutorials

## 8.17 Tutorial 16

#### **Description**

In this example we use the same conduction problem as in the tutorial 4.

Here we introduce:

• Interactive operating procedures

#### **Files**

Model report	tutorial16.xml
Runtime model report	tutorial16-rt.xml
Source code	tutorial16.py

## 8.18 Tutorial 17

#### **Description**

In this example we use the same conduction problem as in the tutorial 4.

Here we introduce:

• TCPIP Log and TCPIPLogServer

#### **Files**

Model report	tutorial17.xml
Runtime model report	tutorial17-rt.xml
Source code	tutorial17.py

## 8.19 Optimization tutorial 1

#### **Description**

This tutorial introduces IPOPT NLP solver, its setup and options.

#### **Files**

Model report	opt_tutorial1.xml
Runtime model report	opt_tutorial1-rt.xml
Source code	opt_tutorial1.py

## 8.20 Optimization tutorial 2

#### **Description**

This tutorial introduces Bonmin MINLP solver, its setup and options.

8.17. Tutorial 16 79

#### **Files**

Model report	opt_tutorial2.xml
Runtime model report	opt_tutorial2-rt.xml
Source code	opt_tutorial2.py

## 8.21 Optimization tutorial 3

#### **Description**

This tutorial introduces NLOPT NLP solver, its setup and options.

#### **Files**

Model report	opt_tutorial3.xml
Runtime model report	opt_tutorial3-rt.xml
Source code	opt_tutorial3.py

## 8.22 Optimization tutorial 4

#### **Description**

This tutorial shows the interoperability between DAE Tools and 3rd party optimization software (scipy.optimize) used to minimize the Rosenbrock function.

DAE Tools simulation is used to calculate the objective function and its gradients, while scipy.optimize.fmin function (Nelder-Mead Simplex algorithm) to find the minimum of the Rosenbrock function.

#### Files

Model report	opt_tutorial4.xml
Runtime model report	opt_tutorial4-rt.xml
Source code	opt_tutorial4.py

## 8.23 Optimization tutorial 5

#### **Description**

This tutorial shows the interoperability between DAE Tools and 3rd party optimization software (scipy.optimize) used to fit the simple function with experimental data.

DAE Tools simulation object is used to calculate the objective function and its gradients, while scipy.optimize.leastsq function (a wrapper around MINPACK's lmdif and lmder) implementing Levenberg-Marquardt algorithm is used to estimate the parameters.

80 Chapter 8. Tutorials

## **Files**

Model report	opt_tutorial5.xml
Runtime model report	opt_tutorial5-rt.xml
Source code	opt_tutorial5.py

## 8.24 Optimization tutorial 6

## **Description**

 $dae Minpack Least Sq\ module\ test.$ 

## **Files**

Model report	opt_tutorial6.xml
Runtime model report	opt_tutorial6-rt.xml
Source code	opt_tutorial6.py

82 Chapter 8. Tutorials

CHAPTER

**NINE** 

# **INDICES AND TABLES**

- genindex
- modindex
- search

## **PYTHON MODULE INDEX**

```
d
daetools.code_generators.analyzer,67
daetools.code_generators.ansi_c,68
daetools.code generators.fmi,68
daetools.code_generators.formatter,67
daetools.code_generators.modelica,68
daetools.dae_plotter.mayavi_plot3d,69
daetools.dae_plotter.plot2d,68
daetools.dae_plotter.plot_options,69
daetools.dae_plotter.plotter,68
daetools.dae_simulator.simulator,69
daetools.examples.opt_tutorial1,79
daetools.examples.opt_tutorial2,79
daetools.examples.opt_tutorial3,80
daetools.examples.opt_tutorial4,80
daetools.examples.opt_tutorial5,80
daetools.examples.opt_tutorial6,81
daetools.examples.tutorial1,72
daetools.examples.tutorial10,76
daetools.examples.tutorial11,77
daetools.examples.tutorial12,77
daetools.examples.tutorial13,77
daetools.examples.tutorial14,78
daetools.examples.tutorial15,78
daetools.examples.tutorial16,79
daetools.examples.tutorial17,79
daetools.examples.tutorial2,73
daetools.examples.tutorial3,73
daetools.examples.tutorial4,74
daetools.examples.tutorial5,74
daetools.examples.tutorial6,75
daetools.examples.tutorial7,75
daetools.examples.tutorial8,75
daetools.examples.tutorial9,76
daetools.examples.whats the time, 71
daetools.pyDAE.data_reporters, 57
daetools.pyDAE.variable_types,63
pyActivity, 53
pyCore, 33
pyDataReporting, 56
pyIDAS, 60
pyUnits, 61
S
solvers.bonmin.pyBONMIN,66
solvers.ipopt.pyIPOPT, 66
```

solvers.nlopt.pyNLOPT, 66

solvers.superlu.pySuperLU, 63
solvers.superlu\_mt.pySuperLU\_MT, 64
solvers.trilinos.pyTrilinos, 65

86 Python Module Index

# **INDEX**

Symbols	init() (pyCore.daeOptimizationConstraint method), 42
add() (pyUnits.quantity method), 62	init() (pyCore.daeOptimizationVariable method), 42
and() (pyCore.daeCondition method), 44	init() (pyCore.daeParameter method), 34
call() (pyCore.daeDEDI method), 41	init() (pyCore.daePort method), 39
call() (pyCore.daeDomain method), 34	init() (pyCore.daeScalarExternalFunction method), 40
call() (pyCore.daeParameter method), 35	init() (pyCore.daeStdOutLog method), 52
call() (pyCore.daeScalarExternalFunction method), 40	init() (pyCore.daeTCPIPLog method), 52
call() (pyCore.daeVariable method), 37	init() (pyCore.daeTCPIPLogServer method), 52
call() (pyCore.dae Variable incuriog), 37call() (pyCore.dae VectorExternalFunction method), 40	init() (pyCore.daeVariable method), 35
contains() (pyCore.daeConfig method), 46	init() (pyCore.daeVariableType method), 33
div() (pyUnits.base_unit method), 61	init() (pyCore.daeVariableWrapper method), 46
div() (pyUnits.quantity method), 62	init() (pyCore.daeVectorExternalFunction method), 40
div() (pyUnits.quantity inctiod), 62 div() (pyUnits.unit method), 61	init() (pyUnits.base_unit method), 61
eq() (pyUnits.base_unit method), 61	init() (pyUnits.quantity method), 62
	init() (pyUnits.unit method), 61
eq() (pyUnits.quantity method), 62	le() (pyUnits.quantity method), 62
eq() (pyUnits.unit method), 62	len() (pyCore.adouble_array method), 43
ge() (pyUnits.quantity method), 62	lt() (pyUnits.quantity method), 62
getitem() (pyCore.adouble_array method), 43	mul() (pyUnits.base_unit method), 61
getitem() (pyCore.daeConfig method), 46	mul() (pyUnits.quantity method), 62
getitem() (pyCore.daeDomain method), 34	mul() (pyUnits.unit method), 61
getitem() (pyDataReporting.daeDataReceiverDomain	ne() (pyUnits.base_unit method), 61
method), 59	ne() (pyUnits.quantity method), 62
getitem() (pyDataReport-	ne() (pyUnits.unit method), 62
ing.daeDataReceiverVariableValue method),	neg() (pyUnits.quantity method), 62
60	or() (pyCore.daeCondition method), 44
getitem() (pyDataReporting.daeDataReporterDomain	pos() (pyUnits.quantity method), 62
method), 58	pos() (pyUnits.base_unit method), 61
getitem() (pyDataReport-	pow() (pyUnits.quantity method), 62
ing.daeDataReporterVariableValue method),	pow() (pyUnits.unit method), 62
58	setitem() (pyCore.adouble_array method), 44
gt() (pyUnits.quantity method), 62	setitem() (pyCore.daeConfig method), 46
init() (pyActivity.daeOptimization method), 55	setitem() (pyDataReporting.daeDataReceiverDomain
init() (pyActivity.daeSimulation method), 53	method), 59
init() (pyCore.adouble method), 43	setitem() (pyDataReport-
init() (pyCore.adouble_array method), 43	ing.daeDataReceiverVariableValue method),
init() (pyCore.daeAction method), 41	60
init() (pyCore.daeArrayRange method), 41	setitem() (pyDataReporting.daeDataReporterDomain
init() (pyCore.daeBaseLog method), 51	method), 58
init() (pyCore.daeDEDI method), 41	setitem() (pyDataReport-
init() (pyCore.daeDelegateLog method), 52	ing.daeDataReporterVariableValue method),
init() (pyCore.daeDomain method), 34	58
init() (pyCore.daeDomainIndex method), 40	sub() (pyUnits.quantity method), 62
init() (pyCore.daeEventPort method), 40	sub() (py Omts.quantity method), 62
init() (pyCore.daeFileLog method), 52	A
init() (pyCore.daeIndexRange method), 40	
init() (pyCore.daeLog_t method), 51	Abs() (in module pyCore), 45
init() (pyCore.daeMeasuredVariable method), 42	AbsoluteTolerance (pyCore.daeVariableType attribute), 33
init() (pyCore.daeModel method), 37	AbsoluteTolerances (pyActivity.daeSimulation attribute), 53
init() (pyCore.daeObjectiveFunction method), 42	ACos() (in module pyCore), 45

ACosh() (in module pyCore), 45	COLAMD (pySuperLU_MT.colperm_t attribute), 65
Actions (pyCore.daeOnConditionActions attribute), 41	ColPerm (pySuperLU.superlu_options_t attribute), 64
Actions (pyCore.daeOnEventActions attribute), 41	ColPerm (pySuperLU_MT.superlumt_options_t attribute),
ActiveState (pyCore.daeSTN attribute), 39	65
ActivityAction (pyActivity.daeSimulation attribute), 54	colperm_t (class in pySuperLU), 64
AddDataReporter() (pyDataReport-	colperm_t (class in pySuperLU_MT), 65
ing.daeDelegateDataReporter method), 58	ComponentArrays (pyCore.daeModel attribute), 37
AddDomain() (pyDataReporting.daeDataReceiverVariable	Components (pyCore.daeModel attribute), 37
method), 59	Condition (pyCore.daeOnConditionActions attribute), 41
AddDomain() (pyDataReporting.daeDataReporterVariable	Connect() (pyCore.daeTCPIPLog method), 52
method), 58	Connect() (pyDataReporting.daeDataReporter_t method), 56
$addLine() \   (daetools.dae\_plotter.plot2d.dae2DPlot \   method),$	Connect() (pyDataReporting.daeDelegateDataReporter
68	method), 58
AddLog() (pyCore.daeDelegateLog method), 52	ConnectEventPorts() (pyCore.daeModel method), 37
AddVariableValue() (pyDataReport-	ConnectPorts() (pyCore.daeModel method), 37
ing.daeDataReceiverVariable method), 59	Constant() (in module pyCore), 43
adouble (class in pyCore), 43	Constraints (pyActivity.daeSimulation attribute), 54
adouble_array (class in pyCore), 43	Cos() (in module pyCore), 45
AmesosOptions (pyTrilinos.daeTrilinosSolver attribute), 65	Cosh() (in module pyCore), 45
amount_adsorbed_t (daetools.pyDAE.variable_types at-	CreateArray() (pyCore.daeDomain method), 34
tribute), 63	CreateDistributed() (pyCore.daeDomain method), 34
$analyze Model () \ (dae tools.code\_generators.analyzer.dae Code Generators) \ (dae tools.code\_generators) \ (dae tools.code\_$	
method), 67	method), 54
$analyzePort()\ (daetools.code\_generators.analyzer.daeCodeGenerators.daeCodeGenerat$	
method), 67	CreateInequalityConstraint() (pyActivity.daeSimulation
$analyze Simulation()  (dae tools.code\_generators.analyzer.dae College analyzer.dae analyzer.dae College analyzer.dae analyz$	
method), 67	CurrentTime (pyActivity.daeSimulation attribute), 55
area_t (daetools.pyDAE.variable_types attribute), 63	D
Array() (in module pyCore), 43	_
array() (pyCore.daeParameter method), 35	d() (in module pyCore), 43
array() (pyCore.daeVariable method), 36	d() (pyCore.daeVariable method), 37
ASin() (in module pyCore), 45	d2() (pyCore.daeVariable method), 37
ASinh() (in module pyCore), 45	d2_array() (pyCore.daeVariable method), 36
AssignValue() (pyCore.daeVariable method), 36	d_array() (pyCore.daeVariable method), 36
AssignValues() (pyCore.daeVariable method), 36	dae2DPlot (class in daetools.dae_plotter.plot2d), 68
ATan() (in module pyCore), 45	daeAction (class in pyCore), 41
ATan2() (in module pyCore), 45	daeANSICExpressionFormatter (class in dae-
ATanh() (in module pyCore), 45	tools.code_generators.ansi_c), 68
Average() (in module pyCore), 43	daeArrayRange (class in pyCore), 41
AztecOOOptions (pyTrilinos.daeTrilinosSolver attribute), 65	daeBaseLog (class in pyCore), 51
В	daeBlackHoleDataReporter (class in pyDataReporting), 58
	daeBONMIN (class in pyBONMIN), 66
base_unit (class in pyUnits), 61	daeCodeGenerator_ANSI_C (class in dae-
baseUnit (pyUnits.unit attribute), 62	tools.code_generators.ansi_c), 68
C	daeCodeGenerator_FMI (class in dae-
	tools.code_generators.fmi), 68
C (pyUnits.base_unit attribute), 61	daeCodeGenerator_Modelica (class in dae-
Calculate() (pyCore.daeScalarExternalFunction method), 40	tools.code_generators.modelica), 68
Calculate() (pyCore.daeVectorExternalFunction method), 40	daeCodeGeneratorAnalyzer (class in dae-
CanonicalName (pyCore.daeObject attribute), 34	tools.code_generators.analyzer), 67
Ceil() (in module pyCore), 45	daeCondition (class in pyCore), 44
CleanUpSetupData() (pyActivity.daeSimulation method), 53	daeConfig (class in pyCore), 46
ClearOptions() (pyBONMIN.daeBONMIN method), 66	daeCreateSuperLUSolver() (in module pySuperLU), 63
ClearOptions() (pyIPOPT.daeIPOPT method), 66	daeCreateSuperLUSolver() (in module pySuperLU_MT), 64
closeEvent() (daetools.dae_plotter.plot2d.dae2DPlot	daeCreateTrilinosSolver() (in module pyTrilinos), 65
method), 68	daeDAESolver_t (class in pyIDAS), 60
cnAlgebraic (in module pyCore), 50	daeDataReceiver_t (class in pyDataReporting), 59
cnAssigned (in module pyCore), 51	daeDataReceiverDomain (class in pyDataReporting), 59 daeDataReceiverProcess (class in pyDataReporting), 60
cnDifferential (in module pyCore), 50	daeDataReceiverVariable (class in pyDataReporting), 59
col2hex() (in module daetools.dae_plotter.plot_options), 69	daeDataReceiverVariableValue (class in pyDataReporting), 39
COLAMD (pySuperLU.colperm_t attribute), 64	60
	00

daeDataReporter_t (class in pyDataReporting), 56	daeNoOpDataReporter (class in pyDataReporting), 56
daeDataReporterDomain (class in pyDataReporting), 58	daeObject (class in pyCore), 34
daeDataReporterFile (class in pyDataReporting), 56	daeObjectiveFunction (class in pyCore), 42
daeDataReporterLocal (class in pyDataReporting), 56	daeOnConditionActions (class in pyCore), 41
daeDataReporterRemote (class in pyDataReporting), 57	daeOnEventActions (class in pyCore), 41
daeDataReporterVariable (class in pyDataReporting), 58	daeOptimization (class in pyActivity), 55
daeDataReporterVariableValue (class in pyDataReporting),	daeOptimizationConstraint (class in pyCore), 42
58	daeOptimizationVariable (class in pyCore), 42
daeDEDI (class in pyCore), 41	daeParameter (class in pyCore), 34
daeDelegateDataReporter (class in pyDataReporting), 58	daePlot2dDefaults (class in daetools.dae_plotter.plot2d), 69
daeDelegateLog (class in pyCore), 52	daePlotDataReporter (class in dae-
daeDomain (class in pyCore), 34	tools.pyDAE.data_reporters), 57
daeDomainIndex (class in pyCore), 40	daePort (class in pyCore), 39
daeeActionType (class in pyCore), 50	daePortConnection (class in pyCore), 40
daeeActivityAction (class in pyActivity), 55	daePythonStdOutLog (class in daetools.pyDAE.logs), 52
daeeBinaryFunctions (class in pyCore), 49	daeScalarExternalFunction (class in pyCore), 40
daeeConditionType (class in pyCore), 49	daeSimulation (class in pyActivity), 53
daeeConstraintType (class in pyCore), 48	daeSimulator (class in daetools.dae_simulator.simulator), 69
daeeDiscretizationMethod (class in pyCore), 47	DAESolver (pyActivity.daeSimulation attribute), 53
daeeDomainBounds (class in pyCore), 47	daeStartPlotter() (in module daetools.dae_plotter.plotter), 68
daeeDomainIndexType (class in pyCore), 48	daeState (class in pyCore), 39
daeeDomainType (class in pyCore), 47	daeStdOutLog (class in pyCore), 52
daeeEquationType (class in pyCore), 50	daeSTN (class in pyCore), 39
daeeIDALASolverType (class in pyIDAS), 61	daeSuperLU_MT_Solver (class in pySuperLU_MT), 65
daeeInitialConditionMode (class in pyCore), 48	daeSuperLU_Solver (class in pySuperLU), 64
daeeLogicalBinaryOperator (class in pyCore), 49	daeTCPIPDataReceiver (class in pyDataReporting), 59
daeeLogicalUnaryOperator (class in pyCore), 49	daeTCPIPDataReceiverServer (class in pyDataReporting),
daeeModelLanguage (class in pyCore), 48	59
daeeModelType (class in pyCore), 50	daeTCPIPDataReporter (class in pyDataReporting), 58
daeeOptimizationVariableType (class in pyCore), 48	daeTCPIPLog (class in pyCore), 52
daeeParameterType (class in pyCore), 47	daeTCPIPLogServer (class in pyCore), 52
daeePortType (class in pyCore), 47	daeTextEditLog (class in daetools.dae_simulator.simulator),
daeEquation (class in pyCore), 39	52
daeEquationExecutionInfo (class in pyCore), 42	daeTEXTFileDataReporter (class in pyDataReporting), 56
daeeRangeType (class in pyCore), 48	daetools.code_generators.analyzer (module), 67
daeeSimulationMode (class in pyActivity), 55	daetools.code_generators.ansi_c (module), 68
daeeSpecialUnaryFunctions (class in pyCore), 49	daetools.code_generators.fmi (module), 68
daeeStopCriterion (class in pyActivity), 55	daetools.code_generators.formatter (module), 67
daeeUnaryFunctions (class in pyTetrivity), 35 daeeUnaryFunctions (class in pyCore), 48	daetools.code_generators.modelica (module), 68
daeEventPort (class in pyCore), 39	daetools.dae_plotter.mayavi_plot3d (module), 69
daeExpressionFormatter (class in dae-	daetools.dae_plotter.plot2d (module), 68
tools.code_generators.formatter), 67	daetools.dae_plotter.plot_options (module), 69
daeFileLog (class in pyCore), 52	daetools.dae_plotter.plotter (module), 68
daeGetConfig() (in module pyCore), 46	daetools.dae_simulator.simulator (module), 69
daeIDALASolver_t (class in pyCore), 63, 66	daetools.examples.opt_tutorial1 (module), 79
daeIDAS (class in pyIDAS), 60	daetools.examples.opt_tutorial2 (module), 79
daeIF (class in pyCore), 39	daetools.examples.opt_tutorial2 (module), 80
daeIndexRange (class in pyCore), 40	daetools.examples.opt_tutorial3 (module), 80
daeIndexRangeType (class in pyCore), 48	daetools.examples.opt_tutorial4 (module), 80
daeIPOPT (class in pyIPOPT), 66	daetools.examples.opt_tutorial6 (module), 80
daeLog_t (class in pyrOr 1), 60 daeLog_t (class in pyCore), 51	daetools.examples.tutorial (module), 72
daeMainWindow (class in daetools.dae_plotter.plotter), 68	daetools.examples.tutorial10 (module), 76
<u> •</u>	daetools.examples.tutorial11 (module), 77
tools.pyDAE.data_reporters), 57	daetools.examples.tutorial12 (module), 77
daeMayavi3DPlot (class in dae-	daetools.examples.tutorial13 (module), 77
tools.dae_plotter.mayavi_plot3d), 69	daetools.examples.tutorial14 (module), 78
daeMeasuredVariable (class in pyCore), 42	daetools.examples.tutorial15 (module), 78
daeMinpackLeastSq (class in daetools.solvers.minpack), 67	daetools.examples.tutorial16 (module), 79
daeModel (class in pyCore), 37	daetools.examples.tutorial17 (module), 79
daeModelicaExpressionFormatter (class in dae-	daetools.examples.tutorial2 (module), 73
tools.code_generators.modelica), 68 daeNLOPT (class in pyNLOPT), 66	daetools.examples.tutorial3 (module), 73 daetools.examples.tutorial4 (module), 74
UACINEATE E CLIASS III DVINEATE LE CO	UACIONIS, EXAMIDIES, LUIOHAIA UMOOMIET. 74

daetools.examples.tutorial5 (module), 74	Domains (pyCore.daePort attribute), 39
daetools.examples.tutorial6 (module), 75	Domains (pyCore.daeVariable attribute), 37
daetools.examples.tutorial7 (module), 75	Domains (pyDataReporting.daeDataReceiverProcess at-
daetools.examples.tutorial8 (module), 75	tribute), 60
daetools.examples.tutorial9 (module), 76	Domains (pyDataReporting.daeDataReceiverVariable
daetools.examples.whats_the_time (module), 71	attribute), 60
daetools.pyDAE.data_reporters (module), 57	Domains (pyDataReporting.daeDataReporterVariable
daetools.pyDAE.variable_types (module), 63	attribute), 58
daeTrilinosSolver (class in pyTrilinos), 65	done() (daetools.dae_simulator.simulator.daeSimulator
daeTrilinosSupportedSolvers() (in module pyTrilinos), 65	method), 69
daeVariable (class in pyCore), 35	DOUBLE (pySuperLU.IterRefine_t attribute), 64
daeVariableType (class in pyCore), 33	drop_tol (pySuperLU_MT.superlumt_options_t attribute), 65
dae Variable Wrapper (class in pyCore), 46	dt() (in module pyCore), 43
daeVectorExternalFunction (class in pyCore), 40	dt() (pyCore.daeVariable method), 37
daeVersion() (in module pyCore), 46	dt_array() (pyCore.daeVariable method), 37
daeVersionBuild() (in module pyCore), 46	dynamic_viscosity_t (daetools.pyDAE.variable_types
dae VersionMajor() (in module pyCore), 46	attribute), 63
dae VersionMinor() (in module pyCore), 46	
DataReceivers (pyDataReport-	E
ing.daeTCPIPDataReceiverServer attribute),	a A A Unknown (ny A ctivity does A ctivity A ction attribute) 55
59	eAAUnknown (pyActivity.daeeActivityAction attribute), 55 eAbs (pyCore.daeeUnaryFunctions attribute), 48
DataReporter (pyActivity.daeSimulation attribute), 53	
DataReporters (pyDataReporting.daeDelegateDataReporter	eAlgebraic (pyCore.daeeEquationType attribute), 50
attribute), 58	eAlgebraicValuesProvided (py-
DeclareEquations() (pyCore.daeModel method), 37	Core.daeeInitialConditionMode attribute), 48
DecreaseIndent() (pyCore.daeBaseLog method), 51	eAllPointsInDomain (pyCore.daeIndexRangeType attribute),
	48
DecreaseIndent() (pyCore.daeLog_t method), 51	eAnd (pyCore.daeeLogicalBinaryOperator attribute), 49
DEDI (pyCore.daeDomainIndex attribute), 40	eArcCos (pyCore.daeeUnaryFunctions attribute), 49
density_t (daetools.pyDAE.variable_types attribute), 63	eArcSin (pyCore.daeeUnaryFunctions attribute), 49
Derivative (pyCore.adouble attribute), 43	eArcTan (pyCore.daeeUnaryFunctions attribute), 49
Description (pyCore.daeObject attribute), 34	eArray (pyCore.daeeDomainType attribute), 47
diag_pivot_thresh (pySuperLU_MT.superlumt_options_t at-	eAverage (pyCore.daeeSpecialUnaryFunctions attribute), 49
tribute), 65	eBFDM (pyCore.daeeDiscretizationMethod attribute), 47
DiagPivotThresh (pySuperLU.superlu_options_t attribute),	eBFUnknown (pyCore.daeeBinaryFunctions attribute), 49
64	eBinaryVariable (pyCore.daeeOptimizationVariableType at-
dictDomains (pyDataReporting.daeDataReceiverProcess at-	tribute), 48
tribute), 60	eBool (pyCore.daeeParameterType attribute), 47
dictDomains (pyDataReporting.daeDataReporterLocal at-	eBOUnknown (pyCore.daeeLogicalBinaryOperator at-
tribute), 56	tribute), 49
dictVariables (pyDataReporting.daeDataReceiverProcess at-	eCDAE (pyCore.daeeModelLanguage attribute), 48
tribute), 60	eCeil (pyCore.daeeUnaryFunctions attribute), 49
dictVariables (pyDataReporting.daeDataReporterLocal at-	eCFDM (pyCore.daeeDiscretizationMethod attribute), 47
tribute), 56	eChangeState (pyCore.daeeActionType attribute), 50
diffusivity_t (daetools.pyDAE.variable_types attribute), 63	eClosedClosed (pyCore.daeeDomainBounds attribute), 47
Disconnect() (pyCore.daeTCPIPLog method), 52	eClosedOpen (pyCore.daeeDomainBounds attribute), 47
Disconnect() (pyDataReporting.daeDataReporter_t method),	eConstantIndex (pyCore.daeeDomainIndexType attribute),
56	48
DiscretizationMethod (pyCore.daeDomain attribute), 34	eContinuousVariable (pyCore.daeeOptimizationVariableType
DiscretizationOrder (pyCore.daeDomain attribute), 34	attribute), 48
DistributedEquationDomainInfos (pyCore.daeEquation at-	eCos (pyCore.daeeUnaryFunctions attribute), 49
tribute), 39	eCTUnknown (pyCore.daeeConditionType attribute), 50
DistributeOnDomain() (pyCore.daeEquation method), 39	eCustomDM (pyCore.daeeDiscretizationMethod attribute),
DistributeOnDomain() (pyCore.daeParameter method), 35	47
DistributeOnDomain() (pyCore.daeVariable method), 37	eCustomRange (pyCore.daeIndexRangeType attribute), 48
Domain (pyCore.daeDEDI attribute), 41	eDAE (pyCore.daeeModelType attribute), 50
Domain (pyCore.daeIndexRange attribute), 40	eDBUnknown (pyCore.daeeDomainBounds attribute), 47
DomainBounds (pyCore.daeDEDI attribute), 41	eDifferentialValuesProvided (py-
DomainIndex (pyCore.daeArrayRange attribute), 41	Core.daeeInitialConditionMode attribute), 48
DomainIndexes (pyCore.daeVariableWrapper attribute), 46	eDistributed (pyCore.daeeDomainType attribute), 47
DomainPoints (pyCore.daeDEDI attribute), 41	eDITUnknown (pyCore.daeeDomainIndexType attribute), 48
Domains (pyCore.daeModel attribute), 38	eDivide (pyCore.daeeBinaryFunctions attribute), 49
Domains (pyCore.daeParameter attribute), 35	Divide (p) core. duceDinary i unerions auriodic), 49
· · · //	

eDMUnknown (pyCore.daeeDiscretizationMethod at-	eOutletPort (pyCore.daeePortType attribute), 47
tribute), 47	eParameterEstimation (pyActivity.daeeSimulationMode at
eDomainIterator (pyCore.daeeDomainIndexType attribute),	tribute), 56
48	ePauseActivity (pyActivity.daeeActivityAction attribute), 55
eDoNotStopAtDiscontinuity (pyActivity.daeeStopCriterion	ePlus (pyCore.daeeBinaryFunctions attribute), 49
attribute), 55	ePower (pyCore.daeeBinaryFunctions attribute), 49
eDTUnknown (pyCore.daeeDomainType attribute), 47	eProduct (pyCore.daeeSpecialUnaryFunctions attribute), 49
eEQ (pyCore.daeeConditionType attribute), 50	ePTUnknown (pyCore.daeeParameterType attribute), 47
eEqualityConstraint (pyCore.daeeConstraintType attribute),	ePYDAE (pyCore.daeeModelLanguage attribute), 48
48	eQuasySteadyState (pyCore.daeeInitialConditionMode at
eETUnknown (pyCore.daeeEquationType attribute), 50	tribute), 48
eExp (pyCore.daeeUnaryFunctions attribute), 49	EquationExecutionInfos (pyCore.daeEquation attribute), 39
eExplicitODE (pyCore.daeeEquationType attribute), 50	Equations (pyCore.daeModel attribute), 38
eFFDM (pyCore.daeeDiscretizationMethod attribute), 47	Equations (pyCore.daePortConnection attribute), 40
eFloor (pyCore.daeeUnaryFunctions attribute), 49	Equations (pyCore.daeState attribute), 39
eGT (pyCore.daeeConditionType attribute), 50	EquationType (pyCore.daeEquation attribute), 39
eGTEQ (pyCore.daeeConditionType attribute), 50	EquationType (pyCore.daeEquationExecutionInfo attribute)
eICTUnknown (pyCore.daeeInitialConditionMode at-	42
tribute), 48	eRange (pyCore.daeeRangeType attribute), 48
eImplicitODE (pyCore.daeeEquationType attribute), 50	eRangeDomainIndex (pyCore.daeeRangeType attribute), 48
eIncrementedDomainIterator (py-	eRangeOfIndexes (pyCore.daeIndexRangeType attribute), 48
Core.daeeDomainIndexType attribute), 48	eRaTUnknown (pyCore.daeeRangeType attribute), 48
eInequalityConstraint (pyCore.daeeConstraintType at-	eReal (pyCore.daeeParameterType attribute), 47
tribute), 48	eReAssignOrReInitializeVariable (pyCore.daeeActionType
eInletPort (pyCore.daeePortType attribute), 47	attribute), 50
eInteger (pyCore.daeeParameterType attribute), 47	eRunActivity (pyActivity.daeeActivityAction attribute), 55
eIntegerVariable (pyCore.daeeOptimizationVariableType at-	eSendEvent (pyCore.daeeActionType attribute), 50
tribute), 48	eSign (pyCore.daeeUnaryFunctions attribute), 49
eIRTUnknown (pyCore.daeIndexRangeType attribute), 48	eSimulation (pyActivity.daeeSimulationMode attribute), 56
eLn (pyCore.daeeUnaryFunctions attribute), 49	eSin (pyCore.daeeUnaryFunctions attribute), 49
eLog (pyCore.daeeUnaryFunctions attribute), 49	eSqrt (pyCore.daeeUnaryFunctions attribute), 49
eLowerBound (pyCore.daeeDomainBounds attribute), 48	eSteadyState (pyCore.daeeModelType attribute), 50
ELSE() (pyCore.daeModel method), 38	eStopAtModelDiscontinuity (pyActivity.daeeStopCriterior
ELSE_IF() (pyCore.daeModel method), 38	attribute), 55
eLT (pyCore.daeeConditionType attribute), 50	eSUFUnknown (pyCore.daeeSpecialUnaryFunctions at
eLTEQ (pyCore.daeeConditionType attribute), 50	tribute), 49
eMax (pyCore.daeeBinaryFunctions attribute), 49	eSum (pyCore.daeeSpecialUnaryFunctions attribute), 49
eMaxInArray (pyCore.daeeSpecialUnaryFunctions at-	eSundialsGMRES (pyIDAS.daeeIDALASolverType at
	4.7
tribute), 49	tribute), 61
eMin (pyCore.daeeBinaryFunctions attribute), 49	eSundialsLapack (pyIDAS.daeeIDALASolverType at
eMinInArray (pyCore.daeeSpecialUnaryFunctions attribute),	tribute), 61
49	eSundialsLU (pyIDAS.daeeIDALASolverType attribute), 61
eMinus (pyCore.daeeBinaryFunctions attribute), 49	ETA (pyCore.daeLog_t attribute), 51
eMLNone (pyCore.daeeModelLanguage attribute), 48	eTan (pyCore.daeeUnaryFunctions attribute), 49
eMTUnknown (pyCore.daeeModelType attribute), 50	eThirdParty (pyIDAS.daeeIDALASolverType attribute), 61
eMulti (pyCore.daeeBinaryFunctions attribute), 49	eUFUnknown (pyCore.daeeUnaryFunctions attribute), 49
Enabled (pyCore.daeLog_t attribute), 51	eUnknownAction (pyCore.daeeActionType attribute), 50
	**
END_IF() (pyCore.daeModel method), 38	eUnknownPort (pyCore.daeePortType attribute), 47
END_STN() (pyCore.daeModel method), 38	eUOUnknown (pyCore.daeeLogicalUnaryOperator at-
EndIndex (pyCore.daeIndexRange attribute), 40	tribute), 49
EndOfData() (pyDataReporting.daeDataReporter_t method),	eUpperBound (pyCore.daeeDomainBounds attribute), 48
56	eUserDefinedAction (pyCore.daeeActionType attribute), 50
EndRegistration() (pyDataReporting.daeDataReporter_t	EventData (pyCore.daeEventPort attribute), 40
method), 56	EventPort (pyCore.daeOnEventActions attribute), 41
eNot (pyCore.daeeLogicalUnaryOperator attribute), 49	EventPorts (pyCore.daeModel attribute), 38
eNotEQ (pyCore.daeeConditionType attribute), 50	Events (pyCore.daeEventPort attribute), 40
eODE (pyCore.daeeModelType attribute), 50	EventTolerance (pyCore.daeCondition attribute), 44
eOpenClosed (pyCore.daeeDomainBounds attribute), 48	Execute() (pyCore.daeAction method), 41
eOpenOpen (pyCore.daeeDomainBounds attribute), 48	Execute() (pyCore.daeOnConditionActions method), 41
eOptimization (pyActivity.daeeSimulationMode attribute),	Execute() (pyCore.daeOnEventActions method), 41
56	Exp() (in module pyCore), 44
eOr (pyCore.daeeLogicalBinaryOperator attribute), 49	Export() (pyCore.daeModel method), 38

Export() (pyCore.daePort method), 39	generatePort() (daetools.code_generators.modelica.daeCodeGenerator_M
ExportObjects() (pyCore.daeModel method), 38	method), 68
Expressions (pyCore.daeCondition attribute), 44	generateSimulation() (dae-
EXTRA (pySuperLU.IterRefine_t attribute), 64	tools.code_generators.ansi_c.daeCodeGenerator_ANSI_C
	method), 68
F	generateSimulation() (dae-
figure_edit() (in module daetools.dae_plotter.plot_options),	tools.code_generators.fmi.daeCodeGenerator_FMI
69	method), 68
Finalize() (daetools.solvers.minpack.daeMinpackLeastSq	generateSimulation() (dae-
method), 67	$tools.code\_generators.modelica.dae Code Generator\_Modelica$
Finalize() (pyActivity.daeOptimization method), 55	method), 68
Finalize() (pyActivity.daeSimulation method), 53	get_bool() (pyTrilinos.TeuchosParameterList method), 66
FindVariable() (pyDataReporting.daeDataReceiverProcess	get_float() (pyTrilinos.TeuchosParameterList method), 66
method), 60	get_int() (pyTrilinos.TeuchosParameterList method), 66
$flattenIdentifier()  (dae tools.code\_generators.formatter.dae Expension and the contraction of the contra$	
method), 67	GetBoolean() (pyCore.daeConfig method), 46
Floor() (in module pyCore), 45	getConfidenceCoefficient() (dae-
$formatDomain()  (dae tools.code\_generators.formatter.dae Expression (Albert Control of Control o$	
method), 67	method), 67
$formatIdentifier()  (dae tools.code\_generators.formatter.dae Exp$	
method), 67	tools.solvers.minpack.daeMinpackLeastSq
formatNumpyArray() (dae-	method), 67
	FormainsIndexesMap() (pyCore.daeParameter method),
method), 68	35
formatNumpyArray() (dae-	
	hagetFit_Dyn() (daetools.solvers.minpack.daeMinpackLeastSq
method), 67	method), 67
formatNumpyArray() (dae-	
tools.code_generators.modelica.daeModelicaExpres	
method), 68	GetFloat() (pyCore.daeConfig method), 46
formatParameter() (daetools.code_generators.formatter.daeEx	
method), 67	GetNameRelativeToParentModel() (pyCore.daeObject
formatQuantity() (daetools.code_generators.ansi_c.daeANSIC	
method), 68 formatQuantity() (daetools.code_generators.formatter.daeExpr	GetQuantity() (pyCore.daeParameter method), 35
	GetString() (pyCore.daeConfig method), 46
method), 67 formatQuantity() (daetools.code_generators.modelica.daeMod	
method), 68	GetStrippedNameRelativeToParentModel() (py-
formatRuntimeConditionNode() (dae-	Core.daeObject method), 34
tools.code_generators.formatter.daeExpressionForm	
method), 67	GetValue() (pyCore.daeVariable method), 35
formatRuntimeNode() (dae-	Gradients (pyCore.daeMeasuredVariable attribute), 42
	hattendients (pyCore.daeObjectiveFunction attribute), 42
method), 67	Gradients (pyCore.daeOptimizationConstraint attribute), 42
formatTimeDerivative() (dae-	1,
tools.code_generators.formatter.daeExpressionForm	atter
method), 67	has_key() (pyCore.daeConfig method), 46
formatUnits() (daetools.code_generators.formatter.daeExpress	
method), 67	heat_t (daetools.pyDAE.variable_types attribute), 63
formatUnits() (daetools.code_generators.modelica.daeModelic	
method), 68	attribute), 63
formatVariable() (daetools.code_generators.formatter.daeExpr	
method), 67	
fraction_t (daetools.pyDAE.variable_types attribute), 63	I (pyUnits.base_unit attribute), 61
ftol_abs (pyNLOPT.daeNLOPT attribute), 67	ID (pyCore.daeObject attribute), 34
ftol_rel (pyNLOPT.daeNLOPT attribute), 67	IF() (pyCore.daeModel method), 38
_	IfpackOptions (pyTrilinos.daeTrilinosSolver attribute), 65
G	IncreaseIndent() (pyCore.daeBaseLog method), 51
GatherInfo (pyCore.adouble attribute), 43	IncreaseIndent() (pyCore.daeLog_t method), 51
GatherInfo (pyCore.adouble_array attribute), 44	Increment (pyCore.daeDomainIndex attribute), 40
generateModel() (daetools.code_generators.modelica.daeCode	**
method) 68	IndentString (nyCore daeLog t attribute) 51

Index (pyCore.daeDomainIndex attribute), 40	laSundialsLU (daetools.dae_simulator.simulator.daeSimulator
IndexMappings (pyActivity.daeSimulation attribute), 55	attribute), 69
InitialConditionMode (pyActivity.daeSimulation attribute),	laSuperLU (daetools.dae_simulator.simulator.daeSimulator
55	attribute), 70
InitialConditionMode (pyCore.daeModel attribute), 38	laSuperLU_CUDA (daetools.dae_simulator.simulator.daeSimulator
InitialConditionMode (pyIDAS.daeDAESolver_t attribute),	attribute), 70
60	laSuperLU_MT (daetools.dae_simulator.simulator.daeSimulator
InitialDerivatives (pyActivity.daeSimulation attribute), 53	attribute), 70
InitialGuess (pyCore.daeVariableType attribute), 33	length_t (daetools.pyDAE.variable_types attribute), 63
Initialize() (daetools.solvers.minpack.daeMinpackLeastSq	Library (pyCore.daeObject attribute), 34
method), 67	LoadInitializationValues() (pyActivity.daeSimulation
Initialize() (pyActivity.daeOptimization method), 55	method), 53
Initialize() (pyActivity.daeSimulation method), 53	LoadOptionsFile() (pyBONMIN.daeBONMIN method), 66
Initialize() (pyCore.daeIDALASolver_t method), 66	LoadOptionsFile() (pyIPOPT.daeIPOPT method), 66
Initial Values (py Activity. dae Simulation attribute), 53	Log (pyActivity.daeSimulation attribute), 53
InputVariables (pyActivity.daeSimulation attribute), 54	Log (pyIDAS.daeDAESolver_t attribute), 60
Integral() (in module pyCore), 43	Log() (in module pyCore), 44
Integrate() (pyActivity.daeSimulation method), 54	Log10() (in module pyCore), 45
	Logs (pyCore.daeDelegateLog attribute), 52
method), 54	LowerBound (pyCore.daeDomain attribute), 34
IntegrateUntilTime() (pyActivity.daeSimulation method), 55	LowerBound (pyCore.daeOptimizationVariable attribute), 42
IsConnected() (pyCore.daeTCPIPLog method), 52	LowerBound (pyCore.daeVariableType attribute), 34
IsConnected() (pyDataReporting.daeDataReporter_t	M
method), 56 IsConnected() (pyDataReport-	
IsConnected() (pyDataReport- ing.daeTCPIPDataReceiverServer method),	m (pyActivity.daeSimulation attribute), 53
59	M (pyUnits.base_unit attribute), 61
IsModelDynamic (pyCore.daeModel attribute), 38	Max() (in module pyCore), 45
items() (pyCore.adouble_array method), 44	MeasuredVariables (pyActivity.daeSimulation attribute), 54
IterRefine_t (class in pySuperLU), 64	Message() (daetools.dae_simulator.simulator.daeTextEditLog
nerkenne_t (class in pysuperLO), 04	method), 52
J	Message() (daetools.pyDAE.logs.daePythonStdOutLog
	method), 52
JoinMessages() (pyCore.daeLog_t method), 51	Message() (pyCore.daeBaseLog method), 51
L	Message() (pyCore.daeDelegateLog method), 52
	Message() (pyCore.daeFileLog method), 52 Message() (pyCore.daeLog_t method), 51
L (pyUnits.base_unit attribute), 61	Message() (pyCore.daeLog_t method), 51  Message() (pyCore.daeStdOutLog method), 52
$laAmdACML (dae tools. dae\_simulator. simulator. dae Simulator) \\$	Message() (pyCore.daeStdOutLog method), 52 Message() (pyCore.daeTCPIPLog method), 52
attribute), 69	
laAmesos_Klu (daetools.dae_simulator.simulator.daeSimulator	52
attribute), 69 laAmesos_Lapack (daetools.dae_simulator.simulator.daeSimul	
naAmesos_Lapack (daetoois.dae_simulator.simulator.daeSimul	METIS_AT_PLUS_A (pySuperLU_MT.colperm_t at-
attribute), 69 laAmesos_Superlu (daetools.dae_simulator.simulator.daeSimu	
attribute), 69	Min() (in module pyCore), 45
laAmesos_Umfpack (daetools.dae_simulator.simulator.daeSim	MLQptions (pvTrilinos.daeTrilinosSolver attribute), 65
attribute), 69	MMD_AT_PLUS_A (pySuperLU.colperm_t attribute), 64
laAztecOO (daetools.dae_simulator.simulator.daeSimulator	MMD_AT_PLUS_A (pySuperLU_MT.colperm_t attribute),
attribute), 69	65
laCUSP (daetools.dae_simulator.simulator.daeSimulator at-	MMD_ATA (pySuperLU.colperm_t attribute), 64
tribute), 69	MMD_ATA (pySuperLU_MT.colperm_t attribute), 65
laIntelMKL (daetools.dae_simulator.simulator.daeSimulator	Model (pyActivity.daeSimulation attribute), 53
attribute), 69	model (pyActivity.daeSimulation attribute), 53
laIntelPardiso (daetools.dae_simulator.simulator.daeSimulator	Model (pyCore.daeObject attribute), 34
attribute), 69	ModelParameters (pyActivity.daeSimulation attribute), 54
laLapack (daetools.dae_simulator.simulator.daeSimulator at-	ModelType (pyCore.daeModel attribute), 38
tribute), 69	molar_concentration_t (daetools.pyDAE.variable_types at-
laMagmaLapack (daetools.dae_simulator.simulator.daeSimulat	
attribute), 69	molar_flowrate_t (daetools.pyDAE.variable_types attribute),
LargeDiag (pySuperLU.rowperm_t attribute), 64	63
LastSatisfiedCondition (pyActivity.daeSimulation attribute),	molar_flux_t (daetools.pyDAE.variable_types attribute), 63
55	moles_t (daetools.pyDAE.variable_types attribute), 63
	multiplier (pyUnits.base_unit attribute), 61

MY_PERMR (pySuperLU.rowperm_t attribute), 64	NumberOfPoints (pyCore.daeParameter attribute), 35
N	NumberOfPoints (pyCore.daeVariable attribute), 37 NumberOfPoints (pyDataReport-
N (pyUnits.base_unit attribute), 61	ing.daeDataReceiverDomain attribute), 59
Name (pyCore.daeIDALASolver_t attribute), 63, 66	NumberOfPoints (pyDataReport-
Name (pyCore.daeMeasuredVariable attribute), 42	ing.daeDataReceiverVariable attribute), 60
Name (pyCore.daeObject attribute), 34	NumberOfPoints (pyDataReport-
Name (pyCore.daeObjectiveFunction attribute), 42	ing.daeDataReporterDomain attribute), 58 NumberOfPoints (pyDataReport-
Name (pyCore.daeOptimizationConstraint attribute), 42	¥ 7 1
Name (pyCore.daeOptimizationVariable attribute), 42	ing.daeDataReporterVariable attribute), 58 NumberOfPoints (pyDataReport-
Name (pyCore.daeScalarExternalFunction attribute), 40	ing.daeDataReporterVariableValue attribute),
Name (pyCore.daeVariableType attribute), 34	59
Name (pyCore.daeVariableWrapper attribute), 46	NumberOfVariables (pyIDAS.daeDAESolver_t attribute), 60
Name (pyCore.daeVectorExternalFunction attribute), 40 Name (pyDataReporting.daeDataReceiverDomain attribute),	NumIters (pyTrilinos.daeTrilinosSolver attribute), 65
Name (pyDatakeporting.daeDatakecerverDomain attribute),	Trainiers (py rimnosiale rimnosociver altricate), 65
Name (pyDataReporting.daeDataReceiverProcess attribute),	0
60	O (pyUnits.base_unit attribute), 61
Name (pyDataReporting.daeDataReceiverVariable attribute),	ObjectiveFunction (pyActivity.daeSimulation attribute), 54
60	ON_CONDITION() (pyCore.daeModel method), 38
Name (pyDataReporting.daeDataReporterDomain attribute),	ON_EVENT() (pyCore.daeModel method), 38
58	OnConditionActions (pyCore.daeModel attribute), 38
Name (pyDataReporting.daeDataReporterVariable attribute),	OnConditionActions (pyCore.daeState attribute), 39
58	OnEventActions (pyCore.daeModel attribute), 38
Name (pyDataReporting.daeDataReporterVariableValue at-	OnEventActions (pyCore.daeState attribute), 39
tribute), 59	OptimizationVariables (pyActivity.daeSimulation attribute),
Name (pyIDAS.daeDAESolver_t attribute), 60	54
NATURAL (pySuperLU.colperm_t attribute), 64	Options (pySuperLU.daeSuperLU_Solver attribute), 64
NATURAL (pySuperLU_MT.colperm_t attribute), 65	Options (pySuperLU_MT.daeSuperLU_MT_Solver at-
NestedSTNs (pyCore.daeState attribute), 39	tribute), 65
newCurve() (daetools.dae_plotter.plot2d.dae2DPlot	OverallIndex (pyCore.daeVariable attribute), 37
method), 69	OverallIndex (pyCore.daeVariableWrapper attribute), 46
newSurface() (daetools.dae_plotter.mayavi_plot3d.daeMayavii	3DPlot D
method), 69 NextReportingTime (pyActivity.daeSimulation attribute), 55	
nlpBONMIN (daetools.dae_simulator.simulator.daeSimulator	panel_size (pySuperLU_MT.superlumt_options_t attribute),
attribute), 70	65
nlpIPOPT (daetools.dae_simulator.simulator.daeSimulator	Parameters (pyCore.daeModel attribute), 38
attribute), 70	Parameters (pyCore.daePort attribute), 39 Pause() (pyActivity.daeSimulation method), 54
nlpNLOPT (daetools.dae_simulator.simulator.daeSimulator	PercentageDone (pyCore.daeLog_t attribute), 51
attribute), 70	Plot() (daetools.pyDAE.data_reporters.daePlotDataReporter
NO (pySuperLU.yes_no_t attribute), 64	method), 57
NO (pySuperLU_MT.yes_no_t attribute), 65	plot2D() (daetools.dae_plotter.plotter.daeMainWindow
no_t (daetools.pyDAE.variable_types attribute), 63	method), 68
Node (pyCore.adouble attribute), 43	plotDefaults (daetools.dae_plotter.plot2d.dae2DPlot at-
Node (pyCore.adouble_array attribute), 44	tribute), 69
Node (pyCore.daeEquationExecutionInfo attribute), 42	Points (pyCore.daeDomain attribute), 34
NoPoints (pyCore.daeArrayRange attribute), 41	Points (pyDataReporting.daeDataReceiverDomain attribute),
NoPoints (pyCore.daeIndexRange attribute), 41	59
NOREFINE (pySuperLU.IterRefine_t attribute), 64	Points (pyDataReporting.daeDataReporterDomain attribute),
NOROWPERM (pySuperLU.rowperm_t attribute), 64	58
nprocs (pySuperLU_MT.superlumt_options_t attribute), 65	Port (pyCore.daeTCPIPLogServer attribute), 52
npyIDs (pyCore.daeVariable attribute), 37	PortArrays (pyCore.daeModel attribute), 38
npyPoints (pyCore.daeDomain attribute), 34	PortConnections (pyCore.daeModel attribute), 38
npy Values (pyCore.daeParameter attribute), 35	PortFrom (pyCore.daePortConnection attribute), 40
npyValues (pyCore.daeVariable attribute), 37 NumberOfDomains (pyDataReport-	Ports (pyCore.daeModel attribute), 38
ing.daeDataReporterVariable attribute), 58	PortTo (pyCore.daePortConnection attribute), 40
NumberOfEquations (pyActivity.daeSimulation attribute),	Pow() (in module pyCore), 45
53	power_t (daetools.pyDAE.variable_types attribute), 63
NumberOfIntervals (pyCore.daeDomain attribute), 34	PreconditionerName (pyTrilinos.daeTrilinosSolver attribute), 66
NumberOfPoints (pyCore.daeDomain attribute), 34	uibuic), 00

pressure_t (daetools.pyDAE.variable_types attribute), 63	Residual (pyCore.daeMeasuredVariable attribute), 42
Print() (pyTrilinos.TeuchosParameterList method), 66	Residual (pyCore.daeObjectiveFunction attribute), 42
PrintOptions() (pyBONMIN.daeBONMIN method), 66	Residual (pyCore.daeOptimizationConstraint attribute), 42
PrintOptions() (pyIPOPT.daeIPOPT method), 66	Resize() (pyCore.adouble_array method), 44
PrintOptions() (pyNLOPT.daeNLOPT method), 67	Resume() (pyActivity.daeSimulation method), 54
PrintPreconditionerInfo() (pyTrilinos.daeTrilinosSolver	RowPerm (pySuperLU.superlu_options_t attribute), 64
method), 66	rowperm_t (class in pySuperLU), 64
PrintProgress (pyCore.daeLog_t attribute), 51	Run() (daetools.solvers.minpack.daeMinpackLeastSq
PrintStat (pySuperLU.superlu_options_t attribute), 64	method), 67
PrintStat (pySuperLU_MT.superlumt_options_t attribute),	Run() (pyActivity.daeOptimization method), 55
65	Run() (pyActivity.daeSimulation method), 54
PrintUserOptions() (pyBONMIN.daeBONMIN method), 66	RuntimeNode (pyCore.daeAction attribute), 41
PrintUserOptions() (pyIPOPT.daeIPOPT method), 66	RuntimeNode (pyCore.daeCondition attribute), 44
Process (pyDataReporting.daeDataReceiver_t attribute), 59	remainer tode (py core.due condition attribute),
Process (pyDataReporting.daeDataReporterLocal attribute),	S
56	
Process (pyDataReporting.daeTCPIPDataReceiver attribute),	SaveAsMatrixMarketFile() (pySu-
59	perLU.daeSuperLU_Solver method), 64
Product() (in module pyCore), 43	SaveAsMatrixMarketFile() (pySu-
Progress (pyCore.daeLog_t attribute), 51	perLU_MT.daeSuperLU_MT_Solver method),
pyActivity (module), 53	65
pyCore (module), 33	SaveAsMatrixMarketFile() (pyTrilinos.daeTrilinosSolver
pyDataReporting (module), 56	method), 66
pyIDAS (module), 60	SaveAsXPM() (pyCore.daeIDALASolver_t method), 63
	SaveMatrixAsXPM() (pyIDAS.daeIDAS method), 60
pyUnits (module), 61	SaveModelReport() (pyCore.daeModel method), 39
Q	SaveRuntimeModelReport() (pyCore.daeModel method), 39
·	scaleTo() (pyUnits.quantity method), 62
quantity (class in pyUnits), 62	Scaling (pyCore.daeEquation attribute), 39
П	SendEvent() (pyCore.daeEventPort method), 40
R	SendEventPort (pyCore.daeAction attribute), 41
Range (pyCore.daeArrayRange attribute), 41	SendMessage() (pyDataReporting.daeDataReporterRemote
ReAssignValue() (pyCore.daeVariable method), 36	method), 57
ReAssignValues() (pyCore.daeVariable method), 36	SendMessage() (pyDataReporting.daeTCPIPDataReporter
ReceiveEvent() (pyCore.daeEventPort method), 40	method), 58
RecordEvents (pyCore.daeEventPort attribute), 40	SendVariable() (pyDataReporting.daeDataReporter_t
reformatPlot() (daetools.dae_plotter.plot2d.dae2DPlot	method), 56
method), 69	set_bool() (pyTrilinos.TeuchosParameterList method), 66
RegisterData() (pyActivity.daeSimulation method), 55	set_float() (pyTrilinos.TeuchosParameterList method), 66
RegisterDomain() (pyDataReport-	set_int() (pyTrilinos.TeuchosParameterList method), 66
ing.daeDataReceiverProcess method), 60	set_string() (pyTrilinos.TeuchosParameterList method), 66
RegisterDomain() (pyDataReporting.daeDataReporter_t	SetAbsoluteTolerances() (pyCore.daeVariable method), 36
method), 56	SetBinaryOptimizationVariable() (pyActivity.daeSimulation
RegisterVariable() (pyDataReport-	method), 54
ing.daeDataReceiverProcess method), 60	SetBoolean() (pyCore.daeConfig method), 46
RegisterVariable() (pyDataReporting.daeDataReporter_t	SetContinuousOptimizationVariable() (pyActiv-
method), 56	ity.daeSimulation method), 54
Reinitialize() (pyActivity.daeSimulation method), 55	SetFloat() (pyCore.daeConfig method), 46
RelativeTolerance (pyActivity.daeSimulation attribute), 53	SetInitialCondition() (pyCore.daeVariable method), 36
RelativeTolerance (pyIDAS.daeDAESolver_t attribute), 60	SetInitialConditions() (pyCore.daeVariable method), 36
relax (pySuperLU_MT.superlumt_options_t attribute), 65	SetInitialGuess() (pyCore.daeVariable method), 36
Reload() (pyCore.daeConfig method), 46	SetInitialGuesses() (pyCore.daeVariable method), 36
ReportData() (pyActivity.daeSimulation method), 55	SetInputVariable() (pyActivity.daeSimulation method), 54
ReportingInterval (pyActivity.daeSimulation attribute), 55	SetInteger() (pyCore.daeConfig method), 46
ReportingOn (pyCore.daeParameter attribute), 35	SetIntegerOptimizationVariable() (pyActivity.daeSimulation
ReportingOn (pyCore.daeVariable attribute), 37	method), 54
Reporting Times (py Activity. dae Simulation attribute), 55	SetLASolver() (pyIDAS.daeIDAS method), 60
ReRun() (pyActivity.daeSimulation method), 54	SetMeasuredVariable() (pyActivity.daeSimulation method),
Reset() (pyActivity.daeSimulation method), 55	54
ReSetInitialCondition() (pyCore.daeVariable method), 36	SetModelParameter() (pyActivity.daeSimulation method), 54
ReSetInitialConditions() (pyCore.daeVariable method), 36	SetOption() (pyBONMIN.daeBONMIN method), 66
Residual (pyCore.daeEquation attribute), 39	SetOption() (pyIPOPT.daeIPOPT method), 66
residual (py core.uaci quanon autrouc), 37	I 0 115

$SetProgress()  (daetools.dae\_simulator.simulator.daeTextEditLorente (Albert 1998))  (daetools.dae\_simulator.simulator.daeTextEditLorente (Albert 1998))  (daetools.dae\_simulator.simulator.daeTextEditLorente (Albert 1998))  (daetools.dae\_simulator.simulator.daeTextEditLorente (Albert 1998))  (daetools.dae\_simulator.daeTextEditLorente (Albert 1998))  (daetools.dae\_simulator.$	osolve() (pyCore.daeIDALASolver_t method), 66
method), 52	SolveInitial() (pyActivity.daeSimulation method), 53
SetProgress() (pyCore.daeBaseLog method), 51	solvers.bonmin.pyBONMIN (module), 66
SetReportingOn() (pyCore.daeModel method), 39	solvers.ipopt.pyIPOPT (module), 66
SetReportingOn() (pyCore.daePort method), 39	solvers.nlopt.pyNLOPT (module), 66
SetString() (pyCore.daeConfig method), 46	solvers.superlu.pySuperLU (module), 63
SetupNode (pyCore.daeAction attribute), 41	solvers.superlu_mt.pySuperLU_MT (module), 64
SetupNode (pyCore.daeCondition attribute), 44	solvers.trilinos.pyTrilinos (module), 65
SetUpOptimization() (pyActivity.daeSimulation method), 54	specific_heat_capacity_t (daetools.pyDAE.variable_types at-
SetUpParameterEstimation() (pyActivity.daeSimulation	tribute), 63
method), 54	specific_heat_conductivity_t (dae-
SetUpParametersAndDomains() (pyActivity.daeSimulation	tools.pyDAE.variable_types attribute), 63
method), 53	Sqrt() (in module pyCore), 45
SetUpSensitivityAnalysis() (pyActivity.daeSimulation	Start() (pyCore.daeTCPIPLogServer method), 52
method), 54	Start() (pyDataReporting.daeDataReceiver_t method), 59
SetUpVariables() (pyActivity.daeSimulation method), 53	Start() (pyDataReporting.daeTCPIPDataReceiver method),
SetValue() (pyCore.daeParameter method), 35	59
SetValue() (pyCore.daeVariable method), 36	Start() (pyDataReporting.daeTCPIPDataReceiverServer
SetValues() (pyCore.daeParameter method), 35	method), 59
SetValues() (pyCore.daeVariable method), 36	StartIndex (pyCore.daeIndexRange attribute), 41
showMessage() (daetools.dae_simulator.simulator.daeSimulator	
method), 70	StartNewResultSet() (pyDataReporting.daeDataReporter_t
SimulationMode (pyActivity.daeSimulation attribute), 55	method), 56
Sin() (in module pyCore), 45	StartRegistration() (pyDataReporting.daeDataReporter_t
SINGLE (pySuperLU.IterRefine_t attribute), 64	method), 56
Sinh() (in module pyCore), 45	STATE() (pyCore.daeModel method), 38
slotAbout() (daetools.dae_plotter.plotter.daeMainWindow	States (pyCore.daeSTN attribute), 39
method), 68	StateTo (pyCore.daeAction attribute), 41
slotAnimatedPlot2D() (dae-	Step (pyCore.daeIndexRange attribute), 41
tools.dae_plotter.plotter.daeMainWindow	STN (pyCore.daeAction attribute), 41
method), 68	STN() (pyCore.daeModel method), 38
slotDocumentation() (daetools.dae_plotter.plotter.daeMainWin	= :
method), 68	Stop() (pyCore.daeTCPIPLogServer method), 52
slotExportCSV() (daetools.dae_plotter.plot2d.dae2DPlot	Stop() (pyDataReporting.daeDataReceiver_t method), 59
method), 69	Stop() (pyDataReporting.daeTCPIPDataReceiver method),
slotExportSparseMatrixAsMatrixMarketFormat() (dae-	59
tools.dae_simulator.simulator.daeSimulator	Stop() (pyDataReporting.daeTCPIPDataReceiverServer
method), 70	method), 59
slotOpenSparseMatrixImage() (dae-	StoreInitializationValues() (pyActivity.daeSimulation
tools.dae_simulator.simulator.daeSimulator	method), 53
method), 70	Sum() (in module pyCore), 43
slotPause() (daetools.dae_simulator.simulator.daeSimulator	superlu_options_t (class in pySuperLU), 64
method), 70	superlumt_options_t (class in pySuperLU_MT), 65
slotPlot2D() (daetools.dae_plotter.plotter.daeMainWindow	surface_edit() (in module daetools.dae_plotter.plot_options),
method), 68	69
	SWITCH_TO() (pyCore.daeModel method), 39
slotPlot3D() (daetools.dae_plotter.plotter.daeMainWindow	SWITCH_TO() (pycore.daewioder method), 39
method), 68	T
slotProperties() (daetools.dae_plotter.plot2d.dae2DPlot	
method), 69	T (pyUnits.base_unit attribute), 61
slotRemoveLine() (daetools.dae_plotter.plot2d.dae2DPlot	Tan() (in module pyCore), 45
method), 69	Tanh() (in module pyCore), 45
slotResume() (daetools.dae_simulator.simulator.daeSimulator	temperature_t (daetools.pyDAE.variable_types attribute), 63
method), 70	TeuchosParameterList (class in pyTrilinos), 66
slotRun() (daetools.dae_simulator.simulator.daeSimulator	Time (pyDataReporting.daeDataReceiverVariableValue at-
method), 70	tribute), 60
slotToggleGrid() (daetools.dae_plotter.plot2d.dae2DPlot	Time() (in module pyCore), 43
method), 69	time_t (daetools.pyDAE.variable_types attribute), 63
slotToggleLegend() (daetools.dae_plotter.plot2d.dae2DPlot	TimeHorizon (pyActivity.daeSimulation attribute), 55
method), 69	TimeValues (pyDataReporting.daeDataReceiverVariable at-
slotViewTabularData() (dae-	tribute), 60
tools.dae_plotter.plot2d.dae2DPlot method),	Tolerance (pyTrilinos.daeTrilinosSolver attribute), 66
69	

TotalNumberOfVariables (pyActivity.daeSimulation at-	W
tribute), 53	WriteDataToFile() (daetools.pyDAE.data_reporters.daeMatlabMATFileD
Type (pyCore.daeAction attribute), 41	method), 57
Type (pyCore.daeArrayRange attribute), 41	WriteDataToFile() (pyDataReporting.daeDataReporterFile
Type (pyCore.daeDomain attribute), 34	method), 56
Type (pyCore.daeDomainIndex attribute), 40	WriteDataToFile() (pyDataReport-
Type (pyCore.daeEventPort attribute), 40 Type (pyCore.daeIndexRange attribute), 41	ing.daeTEXTFileDataReporter method), 56
Type (pyCore.daeOptimizationConstraint attribute), 42	X
Type (pyCore.daeOptimizationVariable attribute), 42	
Type (pyCore.daePort attribute), 39	xtol_abs (pyNLOPT.daeNLOPT attribute), 67
Type (pyDataReporting.daeDataReceiverDomain attribute),	xtol_rel (pyNLOPT.daeNLOPT attribute), 67
59	Υ
Type (pyDataReporting.daeDataReporterDomain attribute),	
58	YES (pySuperLU.yes_no_t attribute), 64
11	YES (pySuperLU_MT.yes_no_t attribute), 65
U	yes_no_t (class in pySuperLU), 64
unit (class in pyUnits), 61	yes_no_t (class in pySuperLU_MT), 65
unitDictionary (pyUnits.unit attribute), 62	
Units (pyCore.daeDomain attribute), 34	
Units (pyCore.daeParameter attribute), 35	
Units (pyCore.daeVariableType attribute), 34	
units (pyUnits.quantity attribute), 62	
updateCurves() (daetools.dae_plotter.plot2d.dae2DPlot	
method), 69	
UpperBound (pyCore.daeDomain attribute), 34	
UpperBound (pyCore.daeOptimizationVariable attribute), 42	
UpperBound (pyCore.daeVariableType attribute), 34 UserDefinedActions (pyCore.daeOnConditionActions	
attribute), 41	
UserDefinedActions (pyCore.daeOnEventActions attribute),	
41	
V	
Value (pyCore.adouble attribute), 43	
Value (pyCore.daeMeasuredVariable attribute), 42	
Value (pyCore.daeObjectiveFunction attribute), 42	
Value (pyCore.daeOptimizationConstraint attribute), 42	
Value (pyCore.daeOptimizationVariable attribute), 42 Value (pyCore.daeVariableWrapper attribute), 46	
value (pyUnits.quantity attribute), 62	
valueInSIUnits (pyUnits.quantity attribute), 62	
Values (pyDataReporting.daeDataReceiverVariable at-	
tribute), 60	
Values (pyDataReporting.daeDataReporterVariableValue at-	
tribute), 59	
Variable (pyCore.daeVariableWrapper attribute), 46	
VariableIndexes (pyCore.daeEquationExecutionInfo at-	
tribute), 42	
Variables (pyCore.daeModel attribute), 39	
Variables (pyCore.daePort attribute), 39	
Variables  (py Data Reporting. dae Data Receiver Process  at-	
tribute), 60	
VariableType (pyCore.daeVariable attribute), 37	
VariableType (pyCore.daeVariableWrapper attribute), 46	
Variable Types (pyActivity.daeSimulation attribute), 55	
VariableWrapper (pyCore.daeAction attribute), 41	
velocity_t (daetools.pyDAE.variable_types attribute), 63	
Version (pyCore.daeObject attribute), 34	
volume_t (daetools.pyDAE.variable_types attribute), 63	