DAE Tools Project Documentation

Release 1.3.1

by Dragan Nikolic

April 22, 2013

CONTENTS

CHAPTER

ONE

GETTING STARTED WITH DAE TOOLS

CHAPTER

TWO

PYDAE USER GUIDE

CHAPTER

THREE

PYDAE API REFERENCE

3.1 Module pyCore

3.1.1 Overview

Trt mrt.

3.1.2 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`

Class adouble operates on values/derivatives of domains, parameters and variables. It supports basic mathematical operators (+, -, , /, *), comparison operators (<, <=, >, >=, ==, !=), and logical operators (and, or, not). Operands can be instances of adouble or float values.

`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.

`__len__`(*(adouble_array)self*) → int :
Returns the size of the adouble_array object.

`__getitem__`(*(adouble_array)self*, *(int)index*) → adouble :
Gets an adouble object at the specified index.

__setitem__ (*(adouble_array)self*, *(int)index*, *(adouble)value*) → 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*) → None :
Resizes the adouble_array object to the new size.

items (*(object)arg1*) → object :
Returns an iterator over adouble items in adouble_array object.

class `pyCore.`**daeCondition**
Bases: `Boost.Python.instance`

__or__ (*(daeCondition)self*, *(daeCondition)right*) → daeCondition

Logical operator `or`

__and__ (*(daeCondition)self*, *(daeCondition)right*) → daeCondition

Logical operator `and`

EventTolerance

Expressions

RuntimeNode

SetupNode

Mathematical functions

`Exp`	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

`pyCore.`**Exp** (*(adouble)arg1*) → adouble
Exp((adouble_array)arg1) -> adouble_array

`pyCore.`**`Log`**(*(adouble)arg1*) → adouble
    Log((adouble_array)arg1) -> adouble_array

`pyCore.`**`Log10`**(*(adouble)arg1*) → adouble
    Log10((adouble_array)arg1) -> adouble_array

`pyCore.`**`Sqrt`**(*(adouble)arg1*) → adouble
    Sqrt((adouble_array)arg1) -> adouble_array

`pyCore.`**`Sin`**(*(adouble)arg1*) → adouble
    Sin((adouble_array)arg1) -> adouble_array

`pyCore.`**`Cos`**(*(adouble)arg1*) → adouble
    Cos((adouble_array)arg1) -> adouble_array

`pyCore.`**`Tan`**(*(adouble)arg1*) → adouble
    Tan((adouble_array)arg1) -> adouble_array

`pyCore.`**`ASin`**(*(adouble)arg1*) → adouble
    ASin((adouble_array)arg1) -> adouble_array

`pyCore.`**`ACos`**(*(adouble)arg1*) → adouble
    ACos((adouble_array)arg1) -> adouble_array

`pyCore.`**`ATan`**(*(adouble)arg1*) → adouble
    ATan((adouble_array)arg1) -> adouble_array

`pyCore.`**`Sinh`**(*(adouble)arg1*) → adouble

`pyCore.`**`Cosh`**(*(adouble)arg1*) → adouble

`pyCore.`**`Tanh`**(*(adouble)arg1*) → adouble

`pyCore.`**`ASinh`**(*(adouble)arg1*) → adouble

`pyCore.`**`ACosh`**(*(adouble)arg1*) → adouble

`pyCore.`**`ATanh`**(*(adouble)arg1*) → adouble

`pyCore.`**`ATan2`**(*(adouble)arg1*, *(adouble)arg2*) → adouble

`pyCore.`**`Ceil`**(*(adouble)arg1*) → adouble
    Ceil((adouble_array)arg1) -> adouble_array

`pyCore.`**`Floor`**(*(adouble)arg1*) → adouble
    Floor((adouble_array)arg1) -> adouble_array

`pyCore.`**`Pow`**(*(adouble)arg1*, *(float)arg2*) → adouble
    Pow((adouble)arg1, (adouble)arg2) -> adouble

    Pow((float)arg1, (adouble)arg2) -> adouble

`pyCore.`**`Abs`**(*(adouble)arg1*) → adouble
    Abs((adouble_array)arg1) -> adouble_array

`pyCore.`**`Min`**(*(adouble)arg1*, *(adouble)arg2*) → adouble
    Min((float)arg1, (adouble)arg2) -> adouble

    Min((adouble)arg1, (float)arg2) -> adouble

    Min((adouble_array)adarray) -> adouble

`pyCore.`**`Max`**(*(adouble)arg1*, *(adouble)arg2*) → adouble
    Max((float)arg1, (adouble)arg2) -> adouble

    Max((adouble)arg1, (float)arg2) -> adouble

    Max((adouble_array)adarray) -> adouble

3.1.3 Modelling concepts

...

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

Classes

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

class `pyCore.`**`daeVariableType`**

Bases: `Boost.Python.instance`

`__init__`(*(object)arg1*) → None

__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`

`GetNameRelativeToParentModel`(*(daeObject)self*) → str

`GetStrippedName`(*(daeObject)self*) → str

`GetStrippedNameRelativeToParentModel`(*(daeObject)self*) → str

`ID`

`Library`

Model

Name

Version

class `pyCore.`**daeDomain**

Bases: `pyCore.daeObject`

`__init__`(*(object)arg1*) → None

__init__((object)self, (str)name, (daeModel)parentModel, (object)units [, (str)description='']) -> None

__init__((object)self, (str)name, (daePort)parentPort, (object)units [, (str)description='']) -> None

`__getitem__`(*(daeDomain)self, (int)index*) → adouble

`__call__`(*(daeDomain)self, (int)index*) → adouble

`CreateArray`(*(daeDomain)self, (int)noIntervals*) → None

`CreateDistributed`(*(daeDomain)self, (daeeDiscretizationMethod)discretizationMethod, (int)discretizationOrder, (int)numberOfIntervals, (float)lowerBound, (float)upperBound*) → None

`DiscretizationMethod`

`DiscretizationOrder`

`LowerBound`

`NumberOfIntervals`

`NumberOfPoints`

`Points`

`Type`

`Units`

`UpperBound`

`npyPoints`

class `pyCore.`**daeParameter**

Bases: `pyCore.daeObject`

`GetValue`(*(daeParameter)self*[, *(int)index1*[, ...[, *(int)index8*]]]) → 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*]]]) → 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.

`SetValue`(*(daeParameter)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)value*) → None

Sets the value of the parameter at the specified domain indexes (as `float` or `quantity`). How many arguments `index1, ..., index8` are used depends on the number of domains that the parameter is distributed on.

`SetValues`(*(daeParameter)self, (float)values*) → None

Sets all values of the parameter (as `float` or `quantity`).

`array`(*(daeParameter)self*[, *(object)index1*[, ...[, *(object)index8*]]]) → 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*]]]) → 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 on.

`DistributeOnDomain` (*(daeParameter)self*, *(daeDomain)domain*) → None

`Domains`

`GetDomainsIndexesMap` (*(daeParameter)self*, *(int)indexBase*) → dict

`NumberOfPoints`

`ReportingOn`

`Units`

`npyValues`

class `pyCore.`**`daeVariable`**
Bases: `pyCore.daeObject`

`GetValue` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]]) → 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.

`GetQuantity` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]]) → 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*]]], *(object)value*) → None
Sets the value of the variable at the specified domain indexes (as `float` or `quantity`). How many arguments `index1, ..., index8` are used depends on the number of domains that the variable is distributed on.

`SetValues` (*(daeVariable)self*, *(object)values*) → None
Sets all values of the variable (as `float` or `quantity`).

`AssignValue` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)value*) → None

`AssignValues` (*(daeVariable)self*, *(object)values*) → None

`ReAssignValue` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)value*) → None

`ReAssignValues` (*(daeVariable)self*, *(object)values*) → None

`SetInitialCondition` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)initialCondition*) → None

`SetInitialConditions` (*(daeVariable)self*, *(object)initialConditions*) → None

`ReSetInitialCondition` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)initialCondition*) → None

`ReSetInitialConditions` (*(daeVariable)self*, *(object)initialConditions*) → None

`SetInitialGuess` (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]], *(object)initialGuess*) → None

`SetInitialGuesses` (*(daeVariable)self*, *(object)initialGuesses*) → None

SetAbsoluteTolerances (*(daeVariable)self*, *(object)tolerances*) → None

array (*(daeVariable)self*[, *(object)index1*[, ...[, *(object)index8*]]]) → 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*]]]) → 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()`.

d2_array (*(daeVariable)self*[, *(object)index1*[, ...[, *(object)index8*]]]) → 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 number 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*]]]) → 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()`.

__call__ (*(daeVariable)self*[, *(int)index1*[, ...[, *(int)index8*]]]) → 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*]]]) → 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*]]]) → 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*]]]) → 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.

DistributeOnDomain (*(daeVariable)self*, *(daeDomain)domain*) → None

Domains

GetDomainsIndexesMap (*(daeVariable)arg1*, *(int)self*) → dict

NumberOfPoints

OverallIndex

ReportingOn

VariableType

npyIDs

npyValues

class `pyCore.`**daeModel**
Bases: `pyCore.daeObject`

Base model class.

`__init__` (*(object)arg1*) → None

__init__((object)self, (str)name [, (daeModel)parent=0 [, (str)description='']]) -> 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*) → None
:
Connects two event ports.

`ConnectPorts` (*(daeModel)self*, *(daePort)portFrom*, *(daePort)portTo*) → None :
Connects two ports.

`CreateEquation` (*(daeModel)self*, *(str)name*[, *(str)description=''*[, *(float)scaling=1.0*]]) →
daeEquation :
Creates a new equation. Used to add equations to models or states in state transition networks

`DeclareEquations` (*(daeModel)self*) → 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*) → None :
Adds the last state to a reversible state transition network.

`ELSE_IF` (*(daeModel)self*, *(daeCondition)condition*[, *(float)eventTolerance=0.0*]) → None :
Adds a new state to a reversible state transition network.

`END_IF` (*(daeModel)self*) → None :
Finalises a reversible state transition network.

`END_STN` (*(daeModel)self*) → None :
.

`Equations`
A list of equations in the model.

`EventPorts`
A list of event ports in the model.

`Export` (*(daeModel)self*, *(str)content*, *(daeeModelLanguage)language*, *(daeModelExportContext)modelExportContext*) → None :
.

`ExportObjects` (*(daeModel)self*, *(list)objects*, *(daeeModelLanguage)language*) → str :
.

`IF` (*(daeModel)self*, *(daeCondition)condition*[, *(float)eventTolerance=0.0*]) → 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 ().

ON_CONDITION(*(daeModel)self*, *(daeCondition)condition*[, *(str)switchTo=''*[, *(list)setVariableValues=[]*[, *(list)triggerEvents=[]*[, *(list)userDefinedActions=[]*[, *(float)eventTolerance=0.0*]]]]]) → None :
.

ON_EVENT(*(daeModel)self*, *(daeEventPort)eventPort*[, *(list)switchToStates=[]*[, *(list)setVariableValues=[]*[, *(list)triggerEvents=[]*[, *(list)userDefinedActions=*[ ]]]]]) → None :
.

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*) → daeState :
.

STN(*(daeModel)self*, *(str)stnName*) → daeSTN :
.

STNs
A list of state transition networks in the model.

SWITCH_TO(*(daeModel)self*, *(str)targetState*, *(daeCondition)condition*[, *(float)eventTolerance=0.0*]) → None :
.

SaveModelReport(*(daeModel)self*, *(str)xmlFilename*) → None :
.

SaveRuntimeModelReport(*(daeModel)self*, *(str)xmlFilename*) → None :
.

SetReportingOn(*(daeModel)self*, *(bool)reportingOn*) → 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`

DistributeOnDomain(*(daeEquation)arg1*, *(daeDomain)arg2*, *(daeeDomainBounds)arg3*) → daeDEDI
DistributeOnDomain((daeEquation)arg1, (daeDomain)arg2, (list)arg3) -> daeDEDI

DistributedEquationDomainInfos

EquationExecutionInfos

EquationType

Residual

Scaling

class pyCore.**daeState**
Bases: pyCore.daeObject

Equations

NestedSTNs

StateTransitions

class pyCore.**daeStateTransition**
Bases: pyCore.daeObject

Actions

Condition

class pyCore.**daePort**
Bases: pyCore.daeObject

Domains

Export (*(daePort)arg1*, *(str)arg2*, *(daeeModelLanguage)arg3*, *(daeModelExportContext)arg4*) → None

Parameters

SetReportingOn (*(daePort)arg1*, *(bool)arg2*) → None

Type

Variables

class pyCore.**daeEventPort**
Bases: pyCore.daeObject

EventData

Events

ReceiveEvent (*(daeEventPort)arg1*, *(float)arg2*) → None

RecordEvents

SendEvent (*(daeEventPort)arg1*, *(float)arg2*) → None

Type

class pyCore.**daePortConnection**
Bases: pyCore.daeObject

Equations

PortFrom

PortTo

class pyCore.**daeScalarExternalFunction**
Bases: Boost.Python.instance

__call__ (*(daeScalarExternalFunction)arg1*) → adouble

Calculate (*(daeScalarExternalFunction)arg1*, *(tuple)arg2*, *(dict)arg3*) → object
Calculate((daeScalarExternalFunction)arg1, (tuple)arg2, (dict)arg3) -> None

Name

class pyCore.**daeVectorExternalFunction**
Bases: Boost.Python.instance

`__call__` (*(daeVectorExternalFunction)arg1*) → adouble_array

`Calculate` (*(daeVectorExternalFunction)arg1*, *(tuple)arg2*, *(dict)arg3*) → list
Calculate((daeVectorExternalFunction)arg1, (tuple)arg2, (dict)arg3) -> None

`Name`

class `pyCore.`**`daeDomainIndex`**
Bases: `Boost.Python.instance`

`__init__` (*(object)arg1*) → None
__init__((object)self, (int)index) -> 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)arg1*) → None
__init__((object)self, (daeDomain)domain) -> 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)arg1*) → None
__init__((object)self, (daeDomainIndex)domainIndex) -> None

__init__((object)self, (daeIndexRange)indexRange) -> None

`DomainIndex`

`NoPoints`

`Range`

`Type`

class `pyCore.`**`daeDEDI`**
Bases: `pyCore.daeObject`

`__call__` (*(daeDEDI)self*) → adouble

`Domain`

`DomainBounds`

`DomainPoints`

class pyCore.**daeAction**
Bases: pyCore.daeObject

Execute (*(daeAction)arg1*) → None
Execute((daeAction)arg1) -> None

RuntimeNode

STN

SendEventPort

SetupNode

StateTo

Type

VariableWrapper

class pyCore.**daeOptimizationVariable**
Bases: pyCore.daeOptimizationVariable_t

LowerBound

Name

StartingPoint

Type

UpperBound

Value

class pyCore.**daeObjectiveFunction**
Bases: pyCore.daeObjectiveFunction_t

Gradients

Name

Residual

Value

class pyCore.**daeOptimizationConstraint**
Bases: pyCore.daeOptimizationConstraint_t

Gradients

Name

Residual

Type

Value

class pyCore.**daeMeasuredVariable**
Bases: pyCore.daeMeasuredVariable_t

Gradients

Name

Residual

Value

class pyCore.**daeEquationExecutionInfo**
Bases: Boost.Python.instance

EquationType

Node

VariableIndexes

Logging

daeLog_t
daeBaseLog
daeFileLog
daeStdOutLog
daeTCPIPLog
daeTCPIPLogServer

class `pyCore.`**`daeLog_t`**

Bases: `Boost.Python.instance`

`DecreaseIndent` (*(daeLog_t)arg1*, *(int)arg2*) → None
DecreaseIndent((daeLog_t)arg1, (int)arg2) -> None

ETA

Enabled

`IncreaseIndent` (*(daeLog_t)arg1*, *(int)arg2*) → None
IncreaseIndent((daeLog_t)arg1, (int)arg2) -> None

Indent

IndentString

`JoinMessages` (*(daeLog_t)arg1*, *(str)arg2*) → str
JoinMessages((daeLog_t)arg1, (str)arg2) -> None

`Message` (*(daeLog_t)arg1*, *(str)arg2*, *(int)arg3*) → None
Message((daeLog_t)arg1, (str)arg2, (int)arg3) -> None

PercentageDone

PrintProgress

Progress

class `pyCore.`**`daeBaseLog`**

Bases: `pyCore.daeLog_t`

`DecreaseIndent` (*(daeBaseLog)arg1*, *(int)arg2*) → None

`IncreaseIndent` (*(daeBaseLog)arg1*, *(int)arg2*) → None

`Message` (*(daeBaseLog)arg1*, *(str)arg2*, *(int)arg3*) → None
Message((daeBaseLog)arg1, (str)arg2, (int)arg3) -> None

`SetProgress` (*(daeBaseLog)arg1*, *(float)arg2*) → None
SetProgress((daeBaseLog)arg1, (float)arg2) -> None

class `pyCore.`**`daeFileLog`**

Bases: `pyCore.daeBaseLog`

`Message` (*(daeFileLog)arg1*, *(str)arg2*, *(int)arg3*) → None
Message((daeFileLog)arg1, (str)arg2, (int)arg3) -> None

class `pyCore.`**`daeStdOutLog`**

Bases: `pyCore.daeBaseLog`

`Message` (*(daeStdOutLog)arg1*, *(str)arg2*, *(int)arg3*) → None
Message((daeStdOutLog)arg1, (str)arg2, (int)arg3) -> None

class `pyCore.daeTCPIPLog`
Bases: `pyCore.daeBaseLog`

Message (*(daeTCPIPLog)arg1*, *(str)arg2*, *(int)arg3*) → None
Message((daeTCPIPLog)arg1, (str)arg2, (int)arg3) -> None

class `pyCore.daeTCPIPLogServer`
Bases: `Boost.Python.instance`

MessageReceived (*(daeTCPIPLogServer)arg1*, *(str)arg2*) → None
MessageReceived((daeTCPIPLogServer)arg1, (str)arg2) -> None

Functions

`d`	
`dt`	
`Time`	
`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

Auxiliary classes

`daeVariableWrapper`
`daeConfig`

class `pyCore.daeVariableWrapper`
Bases: `Boost.Python.instance`

__init__ (*(object)self*, *(daeVariable)variable*[, *(str)name='‘*]) → 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) → object

__getitem__((daeConfig)self, (object)propertyPath) → object

__setitem__((daeConfig)self, (object)propertyPath, (object)value) → None

GetBoolean((daeConfig)self, (str)propertyPath[, (bool)defaultValue]) → bool

GetFloat((daeConfig)self, (str)propertyPath[, (float)defaultValue]) → float

GetInteger((daeConfig)self, (str)propertyPath[, (int)defaultValue]) → int

GetString((daeConfig)self, (str)propertyPath[, (str)defaultValue]) → str

Reload((daeConfig)self) → None

SetBoolean((daeConfig)self, (str)propertyPath, (bool)value) → None

SetFloat((daeConfig)self, (str)propertyPath, (float)value) → None

SetInteger((daeConfig)self, (str)propertyPath, (int)value) → None

SetString((daeConfig)self, (str)propertyPath, (str)value) → None

has_key((daeConfig)self, (object)propertyPath) → object

Auxiliary functions

daeGetConfig
daeVersion
daeVersionMajor
daeVersionMinor
daeVersionBuild

pyCore.**daeGetConfig**() → object

pyCore.**daeVersion**([(bool)includeBuild=False]) → str

pyCore.**daeVersionMajor**() → int

pyCore.**daeVersionMinor**() → int

pyCore.**daeVersionBuild**() → int

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.

3.2 Module pyActivity

3.2.1 Overview

Trt mrt.

daeSimulation
daeOptimization

daeSimulation

class pyActivity.**daeSimulation**

Bases: pyActivity.daeSimulation_t

AbsoluteTolerances

ActivityAction

CleanUpSetupData (*(daeSimulation)arg1*) → None

CleanUpSetupData((daeSimulation)arg1) -> None

Constraints

CreateEqualityConstraint (*(daeSimulation)arg1*, *(str)arg2*) → object

CreateInequalityConstraint (*(daeSimulation)arg1*, *(str)arg2*) → object

CurrentTime

DAESolver

DataReporter

Finalize (*(daeSimulation)arg1*) → None

IndexMappings

InitialConditionMode

InitialDerivatives

InitialValues

Initialize (*(daeSimulation)arg1*, *(object)arg2*, *(object)arg3*, *(object)arg4*[, *(bool)CalculateSensitivities=False*]) → None

InputVariables

Integrate (*(daeSimulation)arg1*, *(daeeStopCriterion)arg2*[, *(bool)ReportDataAroundDiscontinuities=True*]) → float

IntegrateForTimeInterval (*(daeSimulation)arg1*, *(float)arg2*[, *(bool)ReportDataAroundDiscontinuities=True*]) → float

IntegrateUntilTime (*(daeSimulation)arg1*, *(float)arg2*, *(daeeStopCriterion)arg3*[, *(bool)ReportDataAroundDiscontinuities=True*]) → float

LoadInitializationValues (*(daeSimulation)arg1*, *(str)arg2*) → None

Log

MeasuredVariables

Model

ModelParameters

NextReportingTime

NumberOfEquations

NumberOfObjectiveFunctions

ObjectiveFunction

ObjectiveFunctions

OptimizationVariables

Pause (*(daeSimulation)arg1*) → None

ReRun (*(daeSimulation)arg1*) → None

RegisterData (*(daeSimulation)arg1*, *(str)arg2*) → None

Reinitialize (*(daeSimulation)arg1*) → None

RelativeTolerance

ReportData (*(daeSimulation)arg1*, *(float)arg2*) → None

ReportingInterval

ReportingTimes

Reset (*(daeSimulation)arg1*) → None

Resume (*(daeSimulation)arg1*) → None

Run (*(daeSimulation)arg1*) → None
Run((daeSimulation)arg1) -> None

SetBinaryOptimizationVariable (*(daeSimulation)arg1*, *(object)arg2*, *(bool)arg3*) → object
SetBinaryOptimizationVariable((daeSimulation)arg1, (object)arg2, (bool)arg3) -> object

SetContinuousOptimizationVariable (*(daeSimulation)arg1*, *(object)arg2*, *(float)arg3*, *(float)arg4*, *(float)arg5*) → object
SetContinuousOptimizationVariable((daeSimulation)arg1, (object)arg2, (float)arg3, (float)arg4, (float)arg5) -> object

SetInputVariable (*(daeSimulation)arg1*, *(object)arg2*) → object
SetInputVariable((daeSimulation)arg1, (object)arg2) -> object

SetIntegerOptimizationVariable (*(daeSimulation)arg1*, *(object)arg2*, *(int)arg3*, *(int)arg4*, *(int)arg5*) → object
SetIntegerOptimizationVariable((daeSimulation)arg1, (object)arg2, (int)arg3, (int)arg4, (int)arg5) -> object

SetMeasuredVariable (*(daeSimulation)arg1*, *(object)arg2*) → object
SetMeasuredVariable((daeSimulation)arg1, (object)arg2) -> object

SetModelParameter (*(daeSimulation)arg1*, *(object)arg2*, *(float)arg3*, *(float)arg4*, *(float)arg5*) → object
SetModelParameter((daeSimulation)arg1, (object)arg2, (float)arg3, (float)arg4, (float)arg5) -> object

SetUpOptimization (*(daeSimulation)arg1*) → None
SetUpOptimization((daeSimulation)arg1) -> None

SetUpParameterEstimation (*(daeSimulation)arg1*) → None
SetUpParameterEstimation((daeSimulation)arg1) -> None

SetUpParametersAndDomains (*(daeSimulation)arg1*) → None
SetUpParametersAndDomains((daeSimulation)arg1) -> None

SetUpSensitivityAnalysis (*(daeSimulation)arg1*) → None
SetUpSensitivityAnalysis((daeSimulation)arg1) -> None

SetUpVariables (*(daeSimulation)arg1*) → None
SetUpVariables((daeSimulation)arg1) -> None

SimulationMode

SolveInitial (*(daeSimulation)arg1*) → None

StoreInitializationValues (*(daeSimulation)arg1*, *(str)arg2*) → None

TimeHorizon

TotalNumberOfVariables

VariableTypes

__init__ (*(object)arg1*) → None

__instance_size__ = 440

__module__ = 'pyActivity'

__reduce__ ()

m

model

daeOptimization

class pyActivity.**daeOptimization**
Bases: pyActivity.daeOptimization_t

Finalize (*(daeOptimization)arg1*) → None

Initialize (*(daeOptimization)arg1*, *(daeSimulation_t)arg2*, *(object)arg3*, *(object)arg4*, *(object)arg5*, *(object)arg6*) → None

Run (*(daeOptimization)arg1*) → None

__init__ (*(object)arg1*) → None

__instance_size__ = 88

__module__ = 'pyActivity'

__reduce__ ()

3.3 Module pyDataReporting

3.3.1 Overview

Trt mrt.

pyDataReporting

daeDataReporter_t

class `pyDataReporting.`**`daeDataReporter_t`**

Bases: `Boost.Python.instance`

`Connect` (*(daeDataReporter_t)arg1*, *(str)arg2*, *(str)arg3*) → bool
Connect((daeDataReporter_t)arg1, (str)arg2, (str)arg3) -> None

`Disconnect` (*(daeDataReporter_t)arg1*) → bool
Disconnect((daeDataReporter_t)arg1) -> None

`EndOfData` (*(daeDataReporter_t)arg1*) → bool
EndOfData((daeDataReporter_t)arg1) -> None

`EndRegistration` (*(daeDataReporter_t)arg1*) → bool
EndRegistration((daeDataReporter_t)arg1) -> None

`IsConnected` (*(daeDataReporter_t)arg1*) → bool
IsConnected((daeDataReporter_t)arg1) -> None

`RegisterDomain` (*(daeDataReporter_t)arg1*, *(daeDataReporterDomain)arg2*) → bool
RegisterDomain((daeDataReporter_t)arg1, (daeDataReporterDomain)arg2) -> None

`RegisterVariable` (*(daeDataReporter_t)arg1*, *(daeDataReporterVariable)arg2*) → bool
RegisterVariable((daeDataReporter_t)arg1, (daeDataReporterVariable)arg2) -> None

`SendVariable` (*(daeDataReporter_t)arg1*, *(daeDataReporterVariableValue)arg2*) → bool
SendVariable((daeDataReporter_t)arg1, (daeDataReporterVariableValue)arg2) -> None

`StartNewResultSet` (*(daeDataReporter_t)arg1*, *(float)arg2*) → bool
StartNewResultSet((daeDataReporter_t)arg1, (float)arg2) -> None

`StartRegistration` (*(daeDataReporter_t)arg1*) → bool
StartRegistration((daeDataReporter_t)arg1) -> None

3.4 Module pyIDAS

3.4.1 Overview

Trt mrt.

daeIDAS

daeIDAS

class `pyIDAS.`**`daeIDAS`**

Bases: `pyIDAS.daeDAESolver_t`

`SaveMatrixAsXPM` (*(daeIDAS)arg1*, *(str)arg2*) → None

`SetLASolver` (*(daeIDAS)arg1*, *(daeeIDALASolverType)arg2*) → None

SetLASolver((daeIDAS)arg1, (object)arg2) -> None

3.5 Module pyUnits

3.5.1 Overview

Trt mrt.

3.5.2 Classes

`unit`
`quantity`

unit

class `pyUnits.`**`unit`**

Bases: `Boost.Python.instance`

`baseUnit`

`unitDictionary`

quantity

class `pyUnits.`**`quantity`**

Bases: `Boost.Python.instance`

`scaleTo` (*(quantity)arg1*, *(object)arg2*) → quantity

`units`

`value`

`valueInSIUnits`

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX