

---

**trnsystor**

***Release 1.3.2***

**Samuel Letellier-Duchesne**

**May 03, 2021**



REFERENCE GUIDE

1	Reference	1
2	Indices and tables	85
	Index	87



## REFERENCE

## 1.1 Main Classes

<i>statement.Constant</i>	CONSTANTS Statement.
<i>statement.Equation</i>	EQUATION Statement.
<i>controlcards.ControlCards</i>	ControlCards class.
<i>deck.Deck</i>	Deck class.
<i>TrnsysModel</i>	TrnsysModel class.
<i>typevariable.Parameter</i>	A subclass of <code>TypeVariable</code> specific to parameters.
<i>typevariable.Input</i>	A subclass of <code>TypeVariable</code> specific to inputs.
<i>typevariable.Output</i>	A subclass of <code>TypeVariable</code> specific to outputs.
<i>typevariable.Derivative</i>	Derivatives class.

## 1.1.1 trnsystor.statement.Constant

**class** `trnsystor.statement.Constant` (*name=None, equals\_to=None, doc=None*)  
CONSTANTS Statement.

The CONSTANTS statement is useful when simulating a number of systems with identical component configurations but with different parameter values, initial input values, or initial values of time dependent variables.

Initialize object.

**Parameters**

- **name** (*str*) – The left hand side of the equation.
- **equals\_to** (*str*, `TypeVariable`) – The right hand side of the equation.
- **doc** (*str*, *optional*) – A small description optionally printed in the deck file.

**classmethod** `from_expression` (*expression*, *doc=None*)  
Create a Constant from a string expression.

Anything before the equal sign (“=”) will become the Constant’s name and anything after will become the equality statement.

**Hint:** The simple expressions are processed much as FORTRAN arithmetic statements are, with one significant exceptions. Expressions are evaluated from left to right with no precedence accorded to any operation over another. This rule must constantly be borne in mind when writing long expressions.

**Parameters**

- **expression** (*str*) – A user-defined expression to parse.
- **doc** (*str*, *optional*) – A small description optionally printed in the deck file.

**property constant\_number**

The equation number (unique).

## 1.1.2 trnsystor.statement.Equation

**class** trnsystor.statement.**Equation** (*name=None, equals\_to=None, doc=None, model=None*)  
EQUATION Statement.

The EQUATIONS statement allows variables to be defined as algebraic functions of constants, previously defined variables, and outputs from TRNSYS components. These variables can then be used in place of numbers in the TRNSYS input file to represent inputs to components; numerical values of parameters; and initial values of inputs and time-dependent variables. The capabilities of the EQUATIONS statement overlap but greatly exceed those of the CONSTANTS statement described in the previous section.

---

**Hint:** In trnsystor, the Equation class works hand in hand with the EquationCollection class. This class behaves a little bit like the equation component in the TRNSYS Studio, meaning that you can list equation in a block, give it a name, etc. See the EquationCollection class for more details.

---

Initialize object.

**Parameters**

- **name** (*str*) – The left hand side of the equation.
- **equals\_to** (*str*, *TypeVariable*) – The right hand side of the equation.
- **doc** (*str*, *optional*) – A small description optionally printed in the deck file.
- **model** (*Component*) – The TrnsysModel this Equation belongs to.

**classmethod from\_expression** (*expression, doc=None*)  
Create an equation from a string expression.

Anything before the equal sign (“=”) will become a Constant and anything after will become the equality statement.

**Example**

Create a simple expression like so:

```
>>> equal = Equation.from_expression("TdbAmb = [011,001]")
```

**Parameters**

- **expression** (*str*) – A user-defined expression to parse.
- **doc** (*str*, *optional*) – A small description optionally printed in the deck file.

**classmethod from\_symbolic\_expression** (*name, exp, \*args, doc=None*)

Create an equation from symbolic expression.

Create an equation with a combination of a generic expression (with placeholder variables) and a list of arguments. The underlying engine will use Sympy and symbolic variables. You can use a mixture of `TypeVariable` and `Equation`, `Constant` as well as the python default `str`.

---

**Important:** If a `str` is passed in place of an expression argument (`args`), make sure to declare that string as an `Equation` or a `Constant` later in the routine.

---

## Examples

In this example, we define a variable (`var_a`) and we want it to be equal to the ‘Outlet Air Humidity Ratio’ divided by  $12 + \log(\text{Temperature to heat source})$ . In a TRNSYS deck file one would have to manually determine the unit numbers and output numbers and write something like : ‘[1, 2]/12 + log([1, 1])’. With the `from_symbolic_expression()`, we can do this very simply:

1. first, define the name of the variable:

```
>>> name = "var_a"
```

2. then, define the expression as a string. Here, the variables *a* and *b* are symbols that represent the two type outputs. Note that their name has been chosen arbitrarily.

```
>>> exp = "log(a) + b / 12"
>>> # would be also equivalent to
>>> exp = "log(x) + y / 12"
```

3. here, we define the actual variables (the type outputs) after loading our model from its proforma:

```
>>> from trnsystor import TrnsysModel
>>> fan = TrnsysModel.from_xml("fan_type.xml")
>>> vars = (fan.outputs[0], fan.outputs[1])
```

---

**Important:** The order of the symbolic variable encountered in the string expression (step 2), from left to right, must be the same for the tuple of variables. For instance, *a* is followed by *b*, therefore `fan.outputs[0]` is followed by `fan.outputs[1]`.

---

4. finally, we create the `Equation`. Note that `vars` is passed with the ‘\*’ declaration to unpack the tuple.

```
>>> from trnsystor.statement import Equation
>>> eq = Equation.from_symbolic_expression(name, exp, *vars)
>>> print(eq)
[1, 1]/12 + log([1, 2])
```

## Parameters

- **name** (*str*) – The name of the variable (left-hand side), of the equation.
- **exp** (*str*) – The expression to evaluate. Use any variable name and mathematical expression.
- **\*args** (*tuple*) – A tuple of `TypeVariable` that will replace the any variable name specified in the above expression.

- **doc** (*str*, *optional*) – A small description optionally printed in the deck file.

**Returns** The Equation Statement object.

**Return type** *Equation*

**property eq\_number**

Return the equation number (unique).

**property idx**

Return the 0-based index of the Equation.

**property unit\_number**

Return the unit number of the EquationCollection self belongs to.

**connect\_to** (*other*, *link\_style\_kwargs=None*)

Connect a single TypeVariable to TypeVariable *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

## Examples

Connect two TypeVariable objects together

```
>>> pipe1.outputs['Outlet_Air_Temperature'].connect_to(  
>>>     other=pipe2.inputs['Inlet_Air_Temperature']  
>>> )
```

**Parameters other** (*TypeVariable*) – The other object.

**Raises** **TypeError** – When trying to connect to anything other than a *TrnsysModel*.

**copy** ()

TypeVariable: Make a copy of *self*.

**classmethod from\_tag** (*tag*, *model=None*)

Class method to create a TypeVariable from an XML tag.

**Parameters**

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**property is\_connected**

Whether or not this TypeVariable is connected to another TypeVariable.

Checks if self is in any keys

**property one\_based\_idx**

Get the 1-based variable index of self such as it appears in Trnsys.

**property predecessor**

Other TypeVariable from which this Input TypeVariable is connected.

Predecessors



### 1.1.3 trnsystor.controlcards.ControlCards

```
class trnsystor.controlcards.ControlCards (version=None, simulation=None, toler-
                                         ances=None, limits=None, nancheck=None,
                                         overwritecheck=None, timereport=None,
                                         dfq=None, width=None, nocheck=None, eq-
                                         solver=None, solver=None, nolist=None,
                                         list=None, map=None)
```

ControlCards class.

The *ControlCards* is a container for all the TRNSYS Simulation Control Statements and Listing Control Statements. It implements the `_to_deck()` method which pretty-prints the statements with their docstrings.

Insure that each simulation has a SIMULATION and END statements.

The other simulation control statements are optional. Default values are assumed for TOLERANCES, LIMITS, SOLVER, EQSOLVER and DFQ if they are not present

#### Parameters

- **version** (*Version*) – The VERSION Statement. labels the deck with the TRNSYS version number. See *Version* for more details.
- **simulation** (*Simulation*) – The SIMULATION Statement.determines the starting and stopping times of the simulation as well as the time step to be used. See *Simulation* for more details.
- **tolerances** (*Tolerances, optional*) – Convergence Tolerances ( TOLERANCES). Specifies the error tolerances to be used during a TRNSYS simulation. See *Tolerances* for more details.
- **limits** (*Limits, optional*) – The LIMITS Statement. Sets limits on the number of iterations that will be performed by TRNSYS during a time step before it is determined that the differential equations and/or algebraic equations are not converging. See *Limits* for more details.
- **nancheck** (*NaNCheck, optional*) – The NAN\_CHECK Statement. An optional debugging feature in TRNSYS. If the NAN\_CHECK statement is present, then the TRNSYS kernel checks every output of each component at each iteration and generates a clean error if ever one of those outputs has been set to the FORTRAN NaN condition. See *NaNCheck* for more details.
- **overwritecheck** (*OverwriteCheck, optional*) – The OVERWRITE\_CHECK Statement. An optional debugging feature in TRNSYS. Checks to make sure that each Type did not write outside its allotted space. See *OverwriteCheck* for more details.
- **timereport** (*TimeReport, optional*) – The TIME\_REPORT Statement. Turns on or off the internal calculation of the time spent on each unit. See *TimeReport* for more details.
- **dfq** (*DFQ, optional*) – Allows the user to select one of three algorithms built into TRNSYS to numerically solve differential equations. See *DFQ* for more details.
- **width** (*Width, optional*) – Set the number of characters to be allowed on a line of TRNSYS output. See *Width* for more details.
- **nocheck** (*NoCheck, optional*) – The Convergence Check Suppression Statement. Remove up to 20 inputs for the convergence check. See *NoCheck* for more details.
- **eqsolver** (*EqSolver, optional*) – The Equation Solving Method Statement. The order in which blocks of EQUATIONS are solved is controlled by the EQSOLVER Statement. See *EqSolver* for more details.

- **solver** (*Solver*, *optional*) – The SOLVER Statement. Select the computational scheme. See *Solver* for more details.
- **nolist** (*NoList*, *optional*) – The NOLIST Statement. See *NoList* for more details.
- **list** (*List*, *optional*) – The LIST Statement. See *List* for more details.
- **map** (*Map*, *optional*) – The MAP Statement. See *Map* for more details.

---

**Note:** Some Statements have not been implemented because only TRNSYS gods use them. Here is a list of Statements that have been ignored:

- The Convergence Promotion Statement (ACCELERATE)
  - The Calling Order Specification Statement (LOOP)
- 

**classmethod** `all()`

Return a SimulationCard with all available Statements.

If not initialized, default values are used. This class method is not recommended since many of the Statements are a time consuming process and should be used as a debugging tool.

**See also:**

- `basic_template()`
- `debug_template()`

**classmethod** `debug_template()`

Return a SimulationCard with useful debugging Statements.

**classmethod** `basic_template()`

Return a SimulationCard with only the required Statements.

**set\_statement** (*statement*)

Set *statement*.

## 1.1.4 trnsystor.deck.Deck

**class** `trnsystor.deck.Deck` (*name*, *author=None*, *date\_created=None*, *control\_cards=None*, *mod-els=None*, *canvas\_width=1200*, *canvas\_height=1000*)

Deck class.

The Deck class holds `TrnsysModel` objects, the `ControlCards` and specifies the name of the project. This class handles reading from a file (see `read_file()`) and printing to a file (see `save()`).

Initialize a Deck object.

### Parameters

- **name** (*str*) – The name of the project.
- **author** (*str*) – The author of the project.
- **date\_created** (*str*) – The creation date. If None, defaults to `datetime.datetime.now()`.
- **control\_cards** (`ControlCards`, *optional*) – The `ControlCards`. See `ControlCards` for more details.

- **models** (*list or trnsystor.collections.components.ComponentCollection*) – A list of Components (TrnsysModel, EquationCollection, etc.). If a list is passed, it is converted to a ComponentCollection. **name** (*str*): A name for this deck. Could be the name of the project.

**Returns** The Deck object.

**Return type** *Deck*

**classmethod read\_file** (*file, author=None, date\_created=None, proforma\_root=None*)

Returns a Deck from a file.

#### Parameters

- **file** (*str or Path*) – Either the absolute or relative path to the file to be opened.
- **author** (*str*) – The author of the project.
- **date\_created** (*str*) – The creation date. If None, defaults to `datetime.datetime.now()`.
- **proforma\_root** (*str*) – Either the absolute or relative path to the folder where proformas (in xml format) are stored.

**property graph**

Return the MultiDiGraph of self.

**check\_deck\_integrity** ()

Checks if Deck definition passes a few obvious rules.

**update\_models** (*amodel*)

Update the `models` attribute with a TrnsysModel (or list).

**Parameters** **amodel** (*Component or list of Component*) –

**Returns** None.

**remove\_models** (*amodel*)

Remove *amodel* from `self.models`.

**to\_file** (*path\_or\_buf, encoding=None, mode='w'*)

Save the Deck object to file.

## Examples

```
>>> from trnsystor.deck import Deck
>>> deck = Deck("Unnamed")
>>> deck.to_file("my_project.dck", None, "w")
```

#### Parameters

- **path\_or\_buf** (*Union[str, Path, IO[AnyStr]]*) – str or file handle, default None File path or object, if None is provided the result is returned as a string. If a file object is passed it should be opened with `newline=""`, disabling universal newlines.
- **encoding** (*str or None*) – Encoding to use.
- **mode** (*str*) – Mode to open `path_or_buf` with.

**save** (*path\_or\_buf*, *encoding=None*, *mode='w'*)

Save Deck to file.

See `to_file()`

**return\_equation\_or\_constant** (*name*)

Return Equation or Constant for name.

If *name* parses to int literal, then the *int* is returned.

**static set\_typevariable** (*dck*, *i*, *model*, *tvar*, *key*)

Set the value to the TypeVariable.

#### Parameters

- **dck** (*Deck*) – the Deck object.
- **i** (*int*) – the idx of the TypeVariable.
- **model** (*Component*) – the component to modify.
- **tvar** (*str* or *float*) – the new value to set.
- **key** (*str*) – the specific type of TypeVariable, eg.: ‘inputs’, ‘parameters’, ‘outputs’, ‘initial\_input\_values’.

### 1.1.5 trnsystor.TrnsysModel

**class** `trnsystor.TrnsysModel` (*meta*, *name*)

TrnsysModel class.

Initialize object.

Alone, this `__init__` method does not do much. See the `from_xml()` class method for the official constructor of this class.

#### Parameters

- **meta** (*MetaData*) – A class containing the model’s metadata.
- **name** (*str*) – A user-defined name for this model.

**classmethod** `from_xml` (*xml*, *\*\*kwargs*)

Class method to create a *TrnsysModel* from an xml string.

#### Examples

Simply pass the xml path to the constructor.

```
>>> from trnsystor import TrnsysModel
>>> fan1 = TrnsysModel.from_xml("Tests/input_files/Type146.xml")
```

#### Parameters

- **xml** (*str* or *Path*) – The path of the xml file.
- **\*\*kwargs** –

**Returns** The TRNSYS model.

**Return type** *TrnsysType*

**copy** (*invalidate\_connections=True*)

Copy object.

The new object has a new `unit_number`. The new object is translated by 50 pts to the right on the canvas.

**Parameters** `invalidate_connections` (*bool*) – If True, connections to other models will be reset.

**property derivatives**

Return derivatives of self.

**property special\_cards**

Return special cards of self.

**property initial\_input\_values**

Return initial input values of self.

**property parameters**

Return parameters of self.

**property external\_files**

Return external files of self.

**property anchor\_points**

Return the 8-AnchorPoints as a dict.

The anchor point location ('top-left', etc.) is the key.

**property reverse\_anchor\_points**

Reverse anchor points.

**update\_meta** (*new\_meta*)

Update self with new MetaData.

**plot** ()

Plot the model.

**property centroid**

Returns the model's center Point().

**Type** Point

**connect\_to** (*other, mapping=None, link\_style\_kwargs=None*)

Connect the outputs of self to the inputs of other.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

## Examples

Connect two *TrnsysModel* objects together by creating a mapping of the outputs of pipe\_1 to the inputs of pipe\_2. In this example we connect output\_0 of pipe\_1 to input\_0 of pipe\_2 and output\_1 of pipe\_1 to input\_1 of pipe\_2:

```
>>> pipe_1.connect_to(pipe_2, mapping={0:0, 1:1})
```

The same can be achieved using input/output names.

```
>>> pipe_1.connect_to(pipe_2, mapping={'Outlet_Air_Temperature':  
>>> 'Inlet_Air_Temperature', 'Outlet_Air_Humidity_Ratio':  
>>> 'Inlet_Air_Humidity_Ratio'})
```

### Parameters

- **other** (*Component*) – The other object
- **mapping** (*dict*) – Mapping of output to input numbers (or names)
- **link\_style\_kwargs** (*dict*, *optional*) – dict of *LinkStyle* parameters

**Raises** ***TypeError*** – A *TypeError* is raised when trying to connect to anything other than a *:class:TrnsysModel*.

### property inputs

returns the model's inputs.

Type *InputCollection*

### invalidate\_connections()

Iterate over successors and predecessors and remove the edges.

Todo: restore paths in self.studio\_canvas.grid

### property is\_connected

Whether or not this *Component* is connected to another *TypeVariable*.

Connected to or connected by.

### property link\_styles

Return *LinkStyles* of self.

### property model

Return the path of this model's proforma.

### property outputs

returns the model's outputs.

Type *OutputCollection*

### property predecessors

Other objects from which this *TypeVariable* is connected. Predecessors.

### set\_canvas\_position(pt, trnsys\_coords=False)

Set position of self in the canvas.

Use cartesian coordinates: origin 0,0 is at bottom-left.

---

**Hint:** The Studio Canvas origin corresponds to the top-left of the canvas. The x coordinates increase from left to right, while the y coordinates increase from top to bottom.

- top-left = “\* \$POSITION 0 0”
- bottom-left = “\* \$POSITION 0 2000”
- top-right = “\* \$POSITION 2000” 0
- bottom-right = “\* \$POSITION 2000 2000”

For convenience, users should deal with cartesian coordinates. trnsystor will deal with the transformation.

---

**Parameters**

- **pt** (*Point or 2-tuple*) – The Point geometry or a tuple of (x, y) coordinates.
- **trnsys\_coords** (*bool*) – Set to True if pt is given in Trnsys Studio coordinates: origin 0,0 is at top-left.

**set\_component\_layer** (*layers*)

Change the layer of self. Pass a list to change multiple layers.

**set\_link\_style** (*other, loc='best', color='#1f78b4', linestyle='-', linewidth=1, path=None*)

Set outgoing link styles between self and other.

**Parameters**

- **other** (*Component*) – The destination model.
- **loc** (*str or tuple*) – loc (str): The location of the anchor. The strings 'top-left', 'top-right', 'bottom-left', 'bottom-right' place the anchor point at the corresponding corner of the *TrnsysModel*. The strings 'top-center', 'center-right', 'bottom-center', 'center-left' place the anchor point at the edge of the corresponding *TrnsysModel*. The string 'best' places the anchor point at the location, among the eight locations defined so far, with the shortest distance with the destination *TrnsysModel* (other). The location can also be a 2-tuple giving the coordinates of the origin *TrnsysModel* and the destination *TrnsysModel*.
- **color** (*str or tuple*) – The color of the line. Accepts any matplotlib color. You can specify colors in many ways, including full names ('green'), hex strings ('#008000'), RGB or RGBA tuples ((0,1,0,1)) or grayscale intensities as a string ('0.8').
- **linestyle** (*str*) – Possible values: '-' or 'solid', '-' or 'dashed', '-' or 'dashdot', '-' or 'dotted', '-' or 'dashdotdot'.
- **linewidth** (*float*) – The width of the line in points.
- **path** (*LineString or MultiLineString, optional*) – The path of the link.

**property successors**

Other objects to which this TypeVariable is connected. Successors.

**property type\_number**

104 for Type104.

**Type** Return the model's type number, eg.

**property unit\_name**

'Type104'.

**Type** Return the model's unit name, eg.

**property unit\_number**

Return the model's unit number (unique).

### 1.1.6 trnsystor.typevariable.Parameter

**class** trnsystor.typevariable.Parameter(*val*, *\*\*kwargs*)

A subclass of *TypeVariable* specific to parameters.

A subclass of *TypeVariable* specific to parameters.

**connect\_to**(*other*, *link\_style\_kwargs=None*)

Connect a single TypeVariable to TypeVariable *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

#### Examples

Connect two *TypeVariable* objects together

```
>>> pipe_1.outputs['Outlet_Air_Temperature'].connect_to(  
>>>     other=pipe2.inputs['Inlet_Air_Temperature']  
>>> )
```

**Parameters** *other* (*TypeVariable*) – The other object.

**Raises** *TypeError* – When trying to connect to anything other than a *TrnsysModel*.

**copy**()

TypeVariable: Make a copy of *self*.

**classmethod** *from\_tag*(*tag*, *model=None*)

Class method to create a TypeVariable from an XML tag.

#### Parameters

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**property** *idx*

Get the 0-based variable index of *self*.

**property** *is\_connected*

Whether or not this TypeVariable is connected to another TypeVariable.

Checks if *self* is in any keys

**property** *one\_based\_idx*

Get the 1-based variable index of *self* such as it appears in Trnsys.

**property** *predecessor*

Other TypeVariable from which this Input TypeVariable is connected.

Predecessors



### 1.1.7 trnsystor.typevariable.Input

**class** trnsystor.typevariable.Input (val, \*\*kwargs)

A subclass of *TypeVariable* specific to inputs.

A subclass of *TypeVariable* specific to inputs.

**connect\_to** (other, link\_style\_kwargs=None)

Connect a single TypeVariable to TypeVariable *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

#### Examples

Connect two *TypeVariable* objects together

```
>>> pipe_1.outputs['Outlet_Air_Temperature'].connect_to(
>>>     other=pipe2.inputs['Inlet_Air_Temperature']
>>> )
```

**Parameters** *other* (*TypeVariable*) – The other object.

**Raises** *TypeError* – When trying to connect to anything other than a *TrnsysModel*.

**copy** ()

TypeVariable: Make a copy of *self*.

**classmethod** **from\_tag** (tag, model=None)

Class method to create a TypeVariable from an XML tag.

#### Parameters

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**property** **idx**

Get the 0-based variable index of *self*.

**property** **is\_connected**

Whether or not this TypeVariable is connected to another TypeVariable.

Checks if *self* is in any keys

**property** **one\_based\_idx**

Get the 1-based variable index of *self* such as it appears in Trnsys.

**property** **predecessor**

Other TypeVariable from which this Input TypeVariable is connected.

Predecessors

### 1.1.8 trnsystor.typevariable.Output

**class** trnsystor.typevariable.Output (val, \*\*kwargs)

A subclass of *TypeVariable* specific to outputs.

A subclass of *TypeVariable* specific to outputs.

**property** is\_connected

Return True if self has any successor.

**property** successors

Other TypeVariables to which this TypeVariable is connected. Successors.

**connect\_to** (other, link\_style\_kwargs=None)

Connect a single TypeVariable to TypeVariable *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

#### Examples

Connect two *TypeVariable* objects together

```
>>> pipe_1.outputs['Outlet_Air_Temperature'].connect_to(  
>>>     other=pipe_2.inputs['Inlet_Air_Temperature']  
>>> )
```

**Parameters** *other* (*TypeVariable*) – The other object.

**Raises** *TypeError* – When trying to connect to anything other than a *TrnsysModel*.

**copy** ()

TypeVariable: Make a copy of self.

**classmethod** from\_tag (tag, model=None)

Class method to create a TypeVariable from an XML tag.

**Parameters**

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**property** idx

Get the 0-based variable index of self.

**property** one\_based\_idx

Get the 1-based variable index of self such as it appears in Trnsys.

**property** predecessor

Other TypeVariable from which this Input TypeVariable is connected.

Predecessors

### 1.1.9 trnsystor.typevariable.Derivative

**class** trnsystor.typevariable.Derivative (val, \*\*kwargs)  
Derivatives class.

the DERIVATIVES for a given TrnsysModel specify initial values, such as the initial temperatures of various nodes in a thermal storage tank or the initial zone temperatures in a multi zone building.

A subclass of *TypeVariable* specific to derivatives.

**connect\_to** (other, link\_style\_kwargs=None)  
Connect a single TypeVariable to TypeVariable *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

#### Examples

Connect two *TypeVariable* objects together

```
>>> pipe_1.outputs['Outlet_Air_Temperature'].connect_to(
>>>     other=pipe2.inputs['Inlet_Air_Temperature']
>>> )
```

**Parameters** *other* (*TypeVariable*) – The other object.

**Raises** *TypeError* – When trying to connect to anything other than a TrnsysModel.

**copy** ()  
TypeVariable: Make a copy of self.

**classmethod from\_tag** (tag, model=None)  
Class method to create a TypeVariable from an XML tag.

**Parameters**

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**property idx**  
Get the 0-based variable index of self.

**property is\_connected**  
Whether or not this TypeVariable is connected to another TypeVariable.  
Checks if self is in any keys

**property one\_based\_idx**  
Get the 1-based variable index of self such as it appears in Trnsys.

**property predecessor**  
Other TypeVariable from which this Input TypeVariable is connected.  
Predecessors

## 1.2 Trnsys Statements

<i>Statement</i>	Statement class.
<i>Version</i>	VERSION Statement.
<i>NaNCheck</i>	NaNCheck Statement.
<i>OverwriteCheck</i>	OverwriteCheck Statement.
<i>TimeReport</i>	TIME_REPORT Statement.
<i>List</i>	LIST Statement.
<i>Simulation</i>	SIMULATION Statement.
<i>Tolerances</i>	TOLERANCES Statement.
<i>Limits</i>	LIMITS Statement.
<i>DFQ</i>	DFQ Statement.
<i>NoCheck</i>	NoCheck Statement.
<i>NoList</i>	NOLIST Statement.
<i>Map</i>	MAP Statement.
<i>EqSolver</i>	EqSolver Statement.
<i>End</i>	END Statement.
<i>Solver</i>	SOLVER Statement.

### 1.2.1 trnsystor.statement.Statement

**class** trnsystor.statement.**Statement**

Statement class.

This is the base class for many of the TRNSYS Simulation Control and Listing Control Statements. It implements common methods such as the repr() method.

Initialize object.

### 1.2.2 trnsystor.statement.Version

**class** trnsystor.statement.**Version** (v=(18, 0))

VERSION Statement.

Added with TRNSYS version 15. The idea of the command is that by labeling decks with the TRNSYS version number that they were created under, it is easy to keep TRNSYS backwards compatible. The version number is saved by the TRNSYS kernel and can be acted upon.

Initialize the Version Statement.

**Parameters** **v** (*tuple*) – A tuple of (major, minor) eg. 18.0 :> (18, 0)

**classmethod** **from\_string** (*string*)

Create Version statement from str version number. eg. 18.0.

### 1.2.3 trnsystor.statement.NaNCheck

**class** trnsystor.statement.NaNCheck (n=0)  
NaNCheck Statement.

One problem that has plagued TRNSYS simulation debuggers is that in Fortran, the “Not a Number” (NaN) condition can be passed along through numerous subroutines without being flagged as an error. For example, a division by zero results in a variable being set to NaN. This NaN can then be used in subsequent equation, causing them to be set to NaN as well. The problem persists for a time until a Range Check or an Integer Overflow error occurs and actually stops simulation progress. To alleviate the problem, the NAN\_CHECK Statement was added as an optional debugging feature in TRNSYS input files.

Initialize a NaNCheck object.

---

**Hint:** If the NAN\_CHECK statement is present (n=1), then the TRNSYS kernel checks every output of each component at each iteration and generates a clean error if ever one of those outputs has been set to the FORTRAN NaN condition. Because this checking is very time consuming, users are not advised to leave NAN\_CHECK set in their input files as it causes simulations to run much more slowly.

---

**Parameters** **n** (*int*) – Is 0 if the NAN\_CHECK feature is not desired or 1 if NAN\_CHECK feature is desired. Default is 0.

### 1.2.4 trnsystor.statement.OverwriteCheck

**class** trnsystor.statement.OverwriteCheck (n=0)  
OverwriteCheck Statement.

A common error in non standard and user written TRNSYS Type routines is to reserve too little space in the global output array. By default, each Type is accorded 20 spots in the global TRNSYS output array. However, there is no way to prevent the Type from then writing in (for example) the 21st spot; the entire global output array is always accessible. By activating the OVERWRITE\_CHECK statement, the TRNSYS kernel checks to make sure that each Type did not write outside its allotted space. As with the NAN\_CHECK statement, OVERWRITE\_CHECK is a time consuming process and should only be used as a debugging tool when a simulation is ending in error.

Initialize an OVERWRITE\_CHECK object.

---

**Hint:** OVERWRITE\_CHECK is a time consuming process and should only be used as a debugging tool when a simulation is ending in error.

---

**Parameters** **n** (*int*) – Is 0 if the OVERWRITE\_CHECK feature is not desired or 1 if OVERWRITE\_CHECK feature is desired.

### 1.2.5 trnsystor.statement.TimeReport

**class** trnsystor.statement.**TimeReport** (*n=0*)  
TIME\_REPORT Statement.

The statement TIME\_REPORT turns on or off the internal calculation of the time spent on each unit. If this feature is desired, the listing file will contain this information at the end of the file.

Initialize a TIME\_REPORT object.

**Parameters** *n* (*int*) – Is 0 if the TIME\_REPORT feature is not desired or 1 if TIME\_REPORT feature is desired.

### 1.2.6 trnsystor.statement.List

**class** trnsystor.statement.**List** (*activate=False*)  
LIST Statement.

The LIST statement is used to turn on the TRNSYS processor listing after it has been turned off by a NOLIST Statement.

Initialize object.

---

**Hint:** The listing is assumed to be on at the beginning of a TRNSYS input file. As many LIST cards as desired may appear in a TRNSYS input file and may be located anywhere in the input file.

---

**Parameters** *activate* (*bool*) – Print to deck if True.

### 1.2.7 trnsystor.statement.Simulation

**class** trnsystor.statement.**Simulation** (*start=0, stop=8760, step=1*)  
SIMULATION Statement.

The SIMULATION statement is required for all simulations, and must be placed in the TRNSYS input file prior to the first UNIT-TYPE Statement. The simulation statement determines the starting and stopping times of the simulation as well as the time step to be used.

Initialize the Simulation Statement.

**Attention:** With TRNSYS 16 and beyond, the starting time is now specified as the time at the beginning of the first time step.

**Parameters**

- **start** (*int*) – The hour of the year at which the simulation is to begin.
- **stop** (*int*) – The hour of the year at which the simulation is to end.
- **step** (*float*) – The time step to be used (hours).

### 1.2.8 trnsystor.statement.Tolerances

**class** trnsystor.statement.**Tolerances** (*epsilon\_d=0.01, epsilon\_a=0.01*)  
TOLERANCES Statement.

The TOLERANCES statement is an optional control statement used to specify the error tolerances to be used during a TRNSYS simulation.

Initialize object.

#### Parameters

- **epsilon\_d** – is a relative (and -epsilon\_d is an absolute) error tolerance controlling the integration error.
- **epsilon\_a** – is a relative (and -epsilon\_a is an absolute) error tolerance controlling the convergence of input and output variables.

### 1.2.9 trnsystor.statement.Limits

**class** trnsystor.statement.**Limits** (*m=25, n=10, p=None*)  
LIMITS Statement.

The LIMITS statement is an optional control statement used to set limits on the number of iterations that will be performed by TRNSYS during a time step before it is determined that the differential equations and/or algebraic equations are not converging.

Initialize object.

#### Parameters

- **m** (*int*) – is the maximum number of iterations which can be performed during a time-step before a WARNING message is printed out.
- **n** (*int*) – is the maximum number of WARNING messages which may be printed before the simulation terminates in ERROR.
- **p** (*int, optional*) – is an optional limit. If any component is called p times in one time step, then the component will be traced (See Section 2.3.5) for all subsequent calls in the timestep. When p is not specified by the user, TRNSYS sets p equal to m.

### 1.2.10 trnsystor.statement.DFQ

**class** trnsystor.statement.**DFQ** (*k=1*)  
DFQ Statement.

The optional DFQ card allows the user to select one of three algorithms built into TRNSYS to numerically solve differential equations (see Manual 08-Programmer's Guide for additional information about solution of differential equations).

Initialize the Differential Equation Solving Method Statement.

**Parameters** **k** (*int, optional*) – an integer between 1 and 3. If a DFQ card is not present in the TRNSYS input file, DFQ 1 is assumed.

---

**Note:** The three numerical integration algorithms are:

1. Modified-Euler method (a 2nd order Runge-Kutta method)
2. Non-self-starting Heun's method (a 2nd order Predictor-Corrector method)

### 3. Fourth-order Adams method (a 4th order Predictor-Corrector method)

---

#### 1.2.11 trnsystor.statement.NoCheck

**class** trnsystor.statement.NoCheck (*inputs=None*)

NoCheck Statement.

TRNSYS allows up to 20 different INPUTS to be removed from the list of INPUTS to be checked for convergence (see Section 1.9).

Initialize object.

**Parameters** *inputs* (*list of Input*) – The list of Inputs.

#### 1.2.12 trnsystor.statement.NoList

**class** trnsystor.statement.NoList (*active=True*)

NOLIST Statement.

The NOLIST statement is used to turn off the listing of the TRNSYS input file.

Initialize object.

**Parameters** *active* (*bool*) – Setting activate to True will add the NOLIST statement

#### 1.2.13 trnsystor.statement.Map

**class** trnsystor.statement.Map (*activate=True*)

MAP Statement.

The MAP statement is an optional control statement that is used to obtain a component output map listing which is particularly useful in debugging component interconnections.

Setting active to True will add the MAP Statement.

**Parameters** *activate* (*bool*) – Setting active to True will add the MAP statement

#### 1.2.14 trnsystor.statement.EqSolver

**class** trnsystor.statement.EqSolver (*n=0*)

EqSolver Statement.

With the release of TRNSYS 16, new methods for solving blocks of EQUATIONS statements were added. For additional information on EQUATIONS statements, please refer to section 6.3.9. The order in which blocks of EQUATIONS are solved is controlled by the EQSOLVER Statement.

Initialize object.

---

**Hint:** *n* can have any of the following values:

1. *n=0* (default if no value is provided) if a component output or TIME changes, update the block of equations that depend upon those values. Then update components that depend upon the first block of equations. Continue looping until all equations have been updated appropriately. This equation blocking method is most like the method used in TRNSYS version 15 and before.



- 
2.  $n=1$  if a component output or TIME changes by more than the value set in the TOLERANCES Statement (see Section 6.3.3), update the block of equations that depend upon those values. Then update components that depend upon the first block of equations. Continue looping until all equations have been updated appropriately.
  3.  $n=2$  treat equations as a component and update them only after updating all components.
- 

**Parameters**  $n$  (*int*) – The order in which the equations are solved.

### 1.2.15 trnsystor.statement.End

**class** trnsystor.statement.End  
END Statement.

The END statement must be the last line of a TRNSYS input file. It signals the TRNSYS processor that no more control statements follow and that the simulation may begin.

Initialize object.

### 1.2.16 trnsystor.statement.Solver

**class** trnsystor.statement.Solver ( $k=0, rf\_min=1, rf\_max=1$ )  
SOLVER Statement.

A SOLVER command has been added to TRNSYS to select the computational scheme. The optional SOLVER card allows the user to select one of two algorithms built into TRNSYS to numerically solve the system of algebraic and differential equations.

Initialize object.

**Parameters**

- $k$  (*int*) – the solution algorithm.
- $rf\_min$  (*float*) – the minimum relaxation factor.
- $rf\_max$  (*float*) – the maximum relaxation factor.

---

**Note:**  $k$  is either the integer 0 or 1. If a SOLVER card is not present in the TRNSYS input file, SOLVER 0 is assumed. If  $k = 0$ , the SOLVER statement takes two additional parameters, RFmin and RFmax:

**The two solution algorithms ( $k$ ) are:**

- 0: Successive Substitution
  - 1: Powell's Method
-

## 1.3 Helper Classes

<code>trnsysmodel.MetaData</code>	General information that is associated with a <i>TrnsysModel</i> .
<code>component.Component</code>	Component class.
<code>externalfile.ExternalFile</code>	ExternalFile class.
<code>typevariable.TypeVariable</code>	TypeVariable class.
<code>typecycle.TypeCycle</code>	TypeCycle class.
<code>studio.StudioHeader</code>	StudioHeader class.
<code>linkstyle.LinkStyle</code>	LinkStyle class.
<code>anchorpoint.AnchorPoint</code>	Handles the anchor point.

### 1.3.1 trnsystor.trnsysmodel.MetaData

```
class trnsystor.trnsysmodel.MetaData (object=None, author=None, organization=None,
                                     editor=None, creationDate=None, modification-
                                     Date=None, mode=None, validation=None,
                                     icon=None, type=None, maxInstance=None, key-
                                     words=None, details=None, comment=None, vari-
                                     ables=None, plugin=None, variablesComment=None,
                                     cycles=None, source=None, externalFiles=None,
                                     compileCommand=None, model=None, special-
                                     Cards=None, **kwargs)
```

General information that is associated with a *TrnsysModel*.

Initialize object with arguments.

This information is contained in the General Tab of the Proforma.

#### Parameters

- **object** (*str*) – A generic name describing the component model.
- **author** (*str*) – The name of the person who wrote the model.
- **organization** (*str*) – The name of organization with which the Author is affiliated.
- **editor** (*str*) – Often, the person creating the Simulation Studio Proforma is not the original author and so the name of the Editor may also be important.
- **creationDate** (*str*) – This is the date of when the model was first written.
- **modificationDate** (*str*) – This is the date when the Proforma was mostly recently revised.
- **mode** (*int*) – 1-Detailed, 2-Simplified, 3-Empirical, 4- Conventional
- **validation** (*int*) – Determine the type of validation that was performed on this model. This can be 1-qualitative, 2-numerical, 3-analytical, 4-experimental and 5-‘in assembly’ meaning that it was verified as part of a larger system which was verified.
- **icon** (*Path*) – Path to the icon.
- **type** (*int*) – The type number.
- **maxInstance** (*int*) – The maximum number of instances this type can be used.
- **keywords** (*str*) – keywords associated with this model.

- **details** (*str*) – The detailed description contains an explanation of the model including a mathematical description of the model
- **comment** (*str*) – The text entered here will appear as a comment in the TRNSYS input file. This allows to attach important information about the component to all its users, including users who prefer to edit the input file with a text editor. This text should be short, to avoid overloading the input file.
- **variables** (*dict*, *optional*) – a list of `TypeVariable`.
- **plugin** (*Path*) – The plug-in path contains the path to the an external application which will be executed to modify component properties instead of the classical properties window.
- **variablesComment** (*str*) – #todo What is this?
- **cycles** (*list*, *optional*) – List of `TypeCycle`.
- **source** (*Path*) – Path of the source code.
- **externalFiles** (*trnsystor.external\_file.ExternalFileCollection*) – A class handling ExternalFiles for this object.
- **compileCommand** (*str*) – Command used to recompile this type.
- **model** (*Path*) – Path of the xml or tmf file.
- **specialCards** (*list of SpecialCards*) – List of `SpecialCards`.
- **\*\*kwargs** – Other keyword arguments passed to the constructor.

**classmethod from\_tag** (*tag*, *\*\*kwargs*)

Create a `TrnsysModel` from an xml tag.

#### Parameters

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **\*\*kwargs** –

**check\_extra\_tags** (*kwargs*)

Detect extra tags in the proforma and warn.

Parameters **kwargs** (*dict*) – dictionary of extra keyword-arguments that would be passed to the constructor.

**classmethod from\_xml** (*xml*, *\*\*kwargs*)

Initialize `MetaData` from xml file.

## 1.3.2 trnsystor.component.Component

**class** `trnsystor.component.Component` (*\*args*, *\*\*kwargs*)

Component class.

Base class for `Trnsys` elements that interact with the Studio. `TrnsysModel`, `ConstantCollection` and `EquationCollection` implement this class.

Initialize class.

#### Parameters

- **name** (*str*) – Name of the component.
- **meta** (*MetaData*) – `MetaData` associated with this component.

**copy()**

Return copy of self.

**property link\_styles**

Return LinkStyles of self.

**set\_canvas\_position** (*pt, trnsys\_coords=False*)

Set position of self in the canvas.

Use cartesian coordinates: origin 0,0 is at bottom-left.

---

**Hint:** The Studio Canvas origin corresponds to the top-left of the canvas. The x coordinates increase from left to right, while the y coordinates increase from top to bottom.

- top-left = “\* \$POSITION 0 0”
- bottom-left = “\* \$POSITION 0 2000”
- top-right = “\* \$POSITION 2000” 0
- bottom-right = “\* \$POSITION 2000 2000”

For convenience, users should deal with cartesian coordinates. trnsystor will deal with the transformation.

---

#### Parameters

- **pt** (*Point or 2-tuple*) – The Point geometry or a tuple of (x, y) coordinates.
- **trnsys\_coords** (*bool*) – Set to True if pt is given in Trnsys Studio coordinates: origin 0,0 is at top-left.

**set\_component\_layer** (*layers*)

Change the layer of self. Pass a list to change multiple layers.

**property unit\_number**

Return the model’s unit number (unique).

**property type\_number**

104 for Type104.

**Type** Return the model’s type number, eg.

**property unit\_name**

‘Type104’.

**Type** Return the model’s unit name, eg.

**property model**

Return the path of this model’s proforma.

**property inputs**

returns the model’s inputs.

**Type** *InputCollection*

**property outputs**

returns the model’s outputs.

**Type** *OutputCollection*

**property centroid**

Returns the model’s center Point().

**Type** Point

**set\_link\_style** (*other*, *loc*='best', *color*='#1f78b4', *linestyle*='-', *linewidth*=1, *path*=None)

Set outgoing link styles between self and other.

**Parameters**

- **other** (*Component*) – The destination model.
- **loc** (*str* or *tuple*) – *loc* (*str*): The location of the anchor. The strings 'top-left', 'top-right', 'bottom-left', 'bottom-right' place the anchor point at the corresponding corner of the *TrnsysModel*. The strings 'top-center', 'center-right', 'bottom-center', 'center-left' place the anchor point at the edge of the corresponding *TrnsysModel*. The string 'best' places the anchor point at the location, among the eight locations defined so far, with the shortest distance with the destination *TrnsysModel* (*other*). The location can also be a 2-tuple giving the coordinates of the origin *TrnsysModel* and the destination *TrnsysModel*.
- **color** (*str* or *tuple*) – The color of the line. Accepts any matplotlib color. You can specify colors in many ways, including full names ('green'), hex strings ('#008000'), RGB or RGBA tuples ((0,1,0,1)) or grayscale intensities as a string ('0.8').
- **linestyle** (*str*) – Possible values: '-' or 'solid', '-' or 'dashed', '-.' or 'dashdot', ':' or 'dotted', '-.' or 'dashdotdot'.
- **linewidth** (*float*) – The width of the line in points.
- **path** (*LineString* or *MultiLineString*, *optional*) – The path of the link.

**connect\_to** (*other*, *mapping*=None, *link\_style\_kwargs*=None)

Connect the outputs of self to the inputs of other.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

**Examples**

Connect two *TrnsysModel* objects together by creating a mapping of the outputs of pipe\_1 to the inputs of pipe\_2. In this example we connect output\_0 of pipe\_1 to input\_0 of pipe\_2 and output\_1 of pipe\_1 to input\_1 of pipe\_2:

```
>>> pipe_1.connect_to(pipe_2, mapping={0:0, 1:1})
```

The same can be achieved using input/output names.

```
>>> pipe_1.connect_to(pipe_2, mapping={'Outlet_Air_Temperature':
>>> 'Inlet_Air_Temperature', 'Outlet_Air_Humidity_Ratio':
>>> 'Inlet_Air_Humidity_Ratio'})
```

**Parameters**

- **other** (*Component*) – The other object
- **mapping** (*dict*) – Mapping of output to input numbers (or names)
- **link\_style\_kwargs** (*dict*, *optional*) – dict of *LinkStyle* parameters

**Raises `TypeError`** – A *TypeError* is raised when trying to connect to anything other than a *:class:TrnsysModel*.

**property successors**

Other objects to which this TypeVariable is connected. Successors.

**property is\_connected**

Whether or not this Component is connected to another TypeVariable.

Connected to or connected by.

**property predecessors**

Other objects from which this TypeVariable is connected. Predecessors.

**invalidate\_connections()**

Iterate over successors and predecessors and remove the edges.

Todo: restore paths in self.studio\_canvas.grid

### 1.3.3 trnsystor.externalfile.ExternalFile

**class** trnsystor.externalfile.**ExternalFile**(*question, default, answers, parameter, designate*)

ExternalFile class.

The External File Specification allows the user to associate a TRNSYS parameter ( typically a logical unit) with an external file through the use of the TRNSYS ASSIGN, FILES, or DESIGNATE statements. This feature allows the author to describe a question that will be asked in the assembly window. If the ASSIGN statement is used, a parameter has to be associated with this external file to define its FORTRAN logical unit number. For example, “Which file contains the meteorological information?” When the input file is generated, it will contain a TRNSYS ASSIGN statement with the answer to the question and the value of the associated parameter. If the user would instead rather use the DESIGNATE input file keyword for their component (please see more information in the TRNEdit manual), the “Designate” checkbox will have to be clicked.

Initialize object from arguments.

**Parameters**

- **question** (*str*) – Question to ask.
- **default** (*str*) – Default answer.
- **answers** (*list of str*) – List of possible answers.
- **parameter** (*str*) – The parameter associated with the external file.
- **designate** (*bool*) – If True, the external files are assigned to logical unit numbers from within the TRNSYS input file. Files that are assigned to a logical unit number using a DESIGNATE statement will not be opened by the TRNSYS kernel.

**classmethod from\_tag** (*tag*)

Create ExternalFile from Tag.

**Parameters** *tag* (*Tag*) – The XML tag with its attributes and contents.



**classmethod** `from_tag(tag, model=None)`

Class method to create a `TypeVariable` from an XML tag.

**Parameters**

- **tag** (*Tag*) – The XML tag with its attributes and contents.
- **model** (*TrnsysModel*) – The model.

**copy()**

`TypeVariable`: Make a copy of `self`.

**property** `is_connected`

Whether or not this `TypeVariable` is connected to another `TypeVariable`.

Checks if `self` is in any keys

**property** `predecessor`

Other `TypeVariable` from which this `Input TypeVariable` is connected.

Predecessors

**property** `idx`

Get the 0-based variable index of `self`.

**property** `one_based_idx`

Get the 1-based variable index of `self` such as it appears in `Trnsys`.

**connect\_to** (*other, link\_style\_kwargs=None*)

Connect a single `TypeVariable` to `TypeVariable` *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output `.dck` file.

---

## Examples

Connect two `TypeVariable` objects together

```
>>> pipe1.outputs['Outlet_Air_Temperature'].connect_to(  
>>>     other=pipe2.inputs['Inlet_Air_Temperature']  
>>> )
```

**Parameters** **other** (*TypeVariable*) – The other object.

**Raises** `TypeError` – When trying to connect to anything other than a `TrnsysModel`.

## 1.3.5 trnsystor.typecycle.TypeCycle

**class** `trnsystor.typecycle.TypeCycle` (*role=None, firstRow=None, lastRow=None, cycles=None, minSize=None, maxSize=None, paramName=None, question=None*)

`TypeCycle` class.

Initialize object.

**classmethod** `from_tag(tag)`

Create `TypeCycle` from `Tag`.



**Parameters** **tag** (*Tag*) – The XML tag with its attributes and contents.

**property default**

Return the default value of self.

**property idxs**

0-based index of the TypeVariable(s) concerned with this cycle.

**property is\_question**

Return True if self is a question.

### 1.3.6 trnsystor.studio.StudioHeader

**class** trnsystor.studio.StudioHeader (*unit\_name, model, position, layer=None*)

StudioHeader class.

Each TrnsysModel has a StudioHeader which handles the studio comments such as position, UNIT\_NAME, model, POSITION, LAYER, LINK\_STYLE

Initialize object.

**Parameters**

- **unit\_name** (*str*) – The unit\_name, eg.: “Type104”.
- **model** (*Path*) – The path of the tmf/xml file.
- **position** (*Point, optional*) – The Point containing coordinates on the canvas.
- **layer** (*list, optional*) – list of layer names on which the model is placed. Defaults to “Main”.

**classmethod from\_component** (*model*)

Create object from TrnsysModel.

### 1.3.7 trnsystor.linkstyle.LinkStyle

**class** trnsystor.linkstyle.LinkStyle (*u, v, loc, color='black', linestyle='-', linewidth=None, path=None, autopath=True*)

LinkStyle class.

Initialize class.

**Parameters**

- **u** (*Component*) – from Model.
- **v** (*Component*) – to Model.
- **loc** (*str or tuple*) – loc (*str*): The location of the anchor. The strings ‘top-left’, ‘top-right’, ‘bottom-left’, ‘bottom-right’ place the anchor point at the corresponding corner of the TrnsysModel. The strings ‘top-center’, ‘center-right’, ‘bottom-center’, ‘center-left’ place the anchor point at the edge of the corresponding TrnsysModel. The string ‘best’ places the anchor point at the location, among the eight locations defined so far, with the shortest distance with the destination TrnsysModel (other). The location can also be a 2-tuple giving the coordinates of the origin TrnsysModel and the destination TrnsysModel.
- **color** (*str or tuple*) – The color of the line. Accepts any matplotlib color. You can specify colors in many ways, including full names (‘green’), hex strings (‘#008000’), RGB or RGBA tuples ((0,1,0,1)) or grayscale intensities as a string (‘0.8’).

- **linestyle** (*str*) – Possible values: ‘-’ or ‘solid’, ‘-’ or ‘dashed’, ‘-.’ or ‘dashdot’, ‘.’ or ‘dotted’, ‘-.’ or ‘dashdotdot’.
- **linewidth** (*float*) – The link line width in points.
- **path** (*LineString* or *MultiLineString*) – The path the link should follow.
- **autopath** (*bool*) – If True, find best path.

**property path**

Return the path of self.

**property anchor\_ids**

Return studio anchor ids.

**set\_color** (*color*)

Set the color of the line.

**get\_color** ()

Return the line color.

**set\_linestyle** (*ls*)

Set the linestyle of the line.

**Parameters** **ls** (*str*) – Possible values: ‘-’ or ‘solid’, ‘-’ or ‘dashed’, ‘-.’ or ‘dashdot’, ‘.’ or ‘dotted’, ‘-.’ or ‘dashdotdot’.

**get\_linestyle** ()

Return the linestyle.

See also `set_linestyle()`.

**set\_linewidth** (*lw*)

Set the line width in points.

**Parameters** **lw** (*float*) – The line width in points.

**get\_linewidth** ()

Return the linewidth.

See also `set_linewidth()`.

## 1.3.8 trnsystor.anchorpoint.AnchorPoint

**class** trnsystor.anchorpoint.**AnchorPoint** (*model*, *offset=20*, *height=40*, *width=40*)

Handles the anchor point. There are 6 anchor points around a component.

Initialize object.

**Parameters**

- **model** (*Component*) – The Component.
- **offset** (*float*) – The offset to give the anchor points from the center of the model position.
- **height** (*float*) – The height of the component in points.
- **width** (*float*) – The width of the component in points.

**studio\_anchor** (*other*, *loc*)

Return the studio anchor based on a location.

**Parameters**

- **other** (*TrnsysModel*) – The other TrnsysModel used to find the anchor of self.
- **loc** (*2-tuple*) – A 2-tuple of location, eg.: (“best”, “best”) or of `anchor_ids`.

**find\_best\_anchors** (*other*)

Find best anchor points to connect self and other.

**property anchor\_points**

Return dict of anchor points str->tuple.

**property reverse\_anchor\_points**

Return dict of anchor points tuple->str.

**property studio\_anchor\_mapping**

Return dict of anchor mapping str->tuple.

**property studio\_anchor\_reverse\_mapping**

Return dict of anchor mapping tuple->str.

**get\_octo\_pts\_dict** (*offset=10*)

Define 8-anchor Point around the TrnsysModel.

In cartesian space and returns a named-dict with human readable meaning. These points are equally dispersed at the four corners and 4 edges of the center, at distance = `offset`.

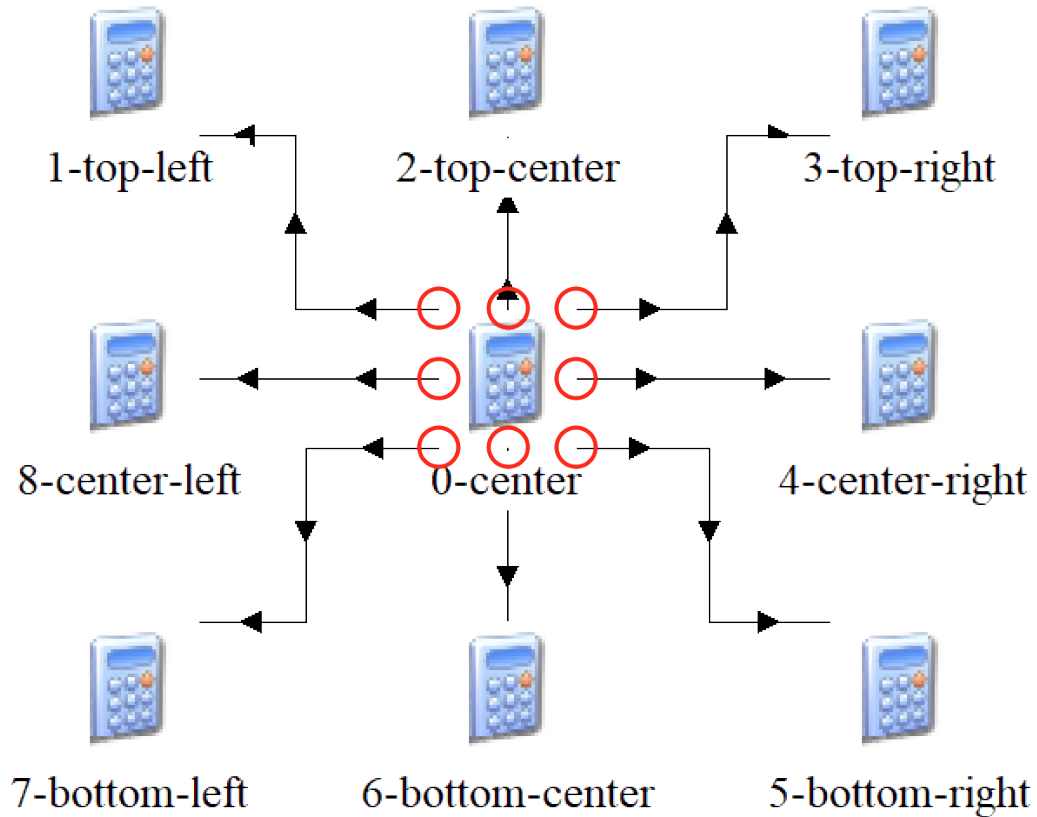
**See also:**

- `trnsystor.component.Component.set_link_style()`
- `trnsystor.linkstyle.LinkStyle`

**Parameters** `offset` (*float*) – The offset around the center point of self.

---

**Note:** In the Studio, a component has 8 anchor points at the four corners and four edges. units. Links can be created on these connections.



**property centroid**  
Return centroid of self.

## 1.4 Collections

<i>ConstantCollection</i>	A class that behaves like a dict and collects one or more Constants.
<i>EquationCollection</i>	Behaves like a dict and collects one or more Equations.
<i>ComponentCollection</i>	A class that handles collections of components.
<i>ExternalFileCollection</i>	A collection of <i>ExternalFile</i> objects.
<i>CycleCollection</i>	Collection of <i>trnsystor.typecycle.TypeCycle</i> .
<i>VariableCollection</i>	A collection of <i>VariableType</i> as a dict.
<i>InputCollection</i>	Subclass of <i>VariableCollection</i> specific to Inputs.
<i>OutputCollection</i>	Subclass of <i>VariableCollection</i> specific to Outputs.

continues on next page

Table 4 – continued from previous page

<i>ParameterCollection</i>	Subclass of <i>VariableCollection</i> specific to Parameters.
----------------------------	---------------------------------------------------------------

### 1.4.1 trnsystor.collections.ConstantCollection

**class** trnsystor.collections.**ConstantCollection** (*mutable=None*, *name=None*, *\*\*kwargs*)

A class that behaves like a dict and collects one or more Constants.

You can pass a dict of Equation or you can pass a list of Equation. In the latter, the Equation.name attribute will be used as a key.

Initialize a new ConstantCollection.

#### Example

```
>>> c_1 = Constant.from_expression("A = 1")
>>> c_2 = Constant.from_expression("B = 2")
>>> ConstantCollection([c_1, c_2])
```

#### Parameters

- **mutable** (*Iterable*, *optional*) – An iterable.
- **name** (*str*) – A user defined name for this collection of constants. This name will be used to identify this block of constants in the .dck file;

**update** (*E=None*, *\*\*F*)

Update D from a dict/list/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for cts.name, cts in E: D[cts.name] = cts In either case, this is followed by: for k in F: D[k] = F[k]

#### Parameters

- **E** (*list*, *dict* or *Constant*) – The constant to add or update in D ( self).
- **F** (*list*, *dict* or *Constant*) – Other constants to update are passed.

**property size**

Return len(self).

**property unit\_number**

Return the unit\_number of self. Negative by design.

---

**Hint:** Only TrnsysModel objects have a positive unit\_number.

---

**property centroid**

Returns the model's center Point().

**Type** Point

**clear** () → None. Remove all items from D.

**connect\_to** (*other*, *mapping=None*, *link\_style\_kwargs=None*)  
Connect the outputs of *self* to the inputs of *other*.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output *.dck* file.

---

## Examples

Connect two `TrnsysModel` objects together by creating a mapping of the outputs of `pipe_1` to the inputs of `pipe_2`. In this example we connect `output_0` of `pipe_1` to `input_0` of `pipe_2` and `output_1` of `pipe_1` to `input_1` of `pipe_2`:

```
>>> pipe_1.connect_to(pipe_2, mapping={0:0, 1:1})
```

The same can be achieved using input/output names.

```
>>> pipe_1.connect_to(pipe_2, mapping={'Outlet_Air_Temperature':  
>>> 'Inlet_Air_Temperature', 'Outlet_Air_Humidity_Ratio':  
>>> 'Inlet_Air_Humidity_Ratio'})
```

## Parameters

- **other** (`Component`) – The other object
- **mapping** (`dict`) – Mapping of output to input numbers (or names)
- **link\_style\_kwargs** (`dict`, *optional*) – dict of `LinkStyle` parameters

**Raises** `TypeError` – A `TypeError` is raised when trying to connect to anything other than a `:class:'TrnsysModel'`.

**copy** ()  
Return copy of self.

**get** (*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to None.

**property inputs**  
returns the model's inputs.

Type `InputCollection`

**invalidate\_connections** ()  
Iterate over successors and predecessors and remove the edges.  
  
Todo: restore paths in `self.studio_canvas.grid`

**property is\_connected**  
Whether or not this `Component` is connected to another `TypeVariable`.  
  
Connected to or connected by.

**items** () → a set-like object providing a view on *D*'s items

**keys** () → a set-like object providing a view on *D*'s keys

**property link\_styles**  
Return `LinkStyles` of self.

**property model**

Return the path of this model's proforma.

**property outputs**

returns the model's outputs.

Type *OutputCollection*

**pop** ( $k[d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  $\rightarrow (k, v)$ , remove and return some (key, value) pair

as a 2-tuple; but raise `KeyError` if D is empty.

**property predecessors**

Other objects from which this `TypeVariable` is connected. Predecessors.

**set\_canvas\_position** ( $pt$ ,  $trnsys\_coords=False$ )

Set position of self in the canvas.

Use cartesian coordinates: origin 0,0 is at bottom-left.

---

**Hint:** The Studio Canvas origin corresponds to the top-left of the canvas. The x coordinates increase from left to right, while the y coordinates increase from top to bottom.

- top-left = “\* \$POSITION 0 0”
- bottom-left = “\* \$POSITION 0 2000”
- top-right = “\* \$POSITION 2000” 0
- bottom-right = “\* \$POSITION 2000 2000”

For convenience, users should deal with cartesian coordinates. trnsystor will deal with the transformation.

---

**Parameters**

- **pt** (*Point* or *2-tuple*) – The Point geometry or a tuple of (x, y) coordinates.
- **trnsys\_coords** (*bool*) – Set to True if pt is given in Trnsys Studio coordinates: origin 0,0 is at top-left.

**set\_component\_layer** ( $layers$ )

Change the layer of self. Pass a list to change multiple layers.

**set\_link\_style** ( $other$ ,  $loc='best'$ ,  $color='#1f78b4'$ ,  $linestyle='-', linewidth=1$ ,  $path=None$ )

Set outgoing link styles between self and other.

**Parameters**

- **other** (*Component*) – The destination model.
- **loc** (*str* or *tuple*) – loc (str): The location of the anchor. The strings ‘top-left’, ‘top-right’, ‘bottom-left’, ‘bottom-right’ place the anchor point at the corresponding corner of the `TrnsysModel`. The strings ‘top-center’, ‘center-right’, ‘bottom-center’, ‘center-left’ place the anchor point at the edge of the corresponding `TrnsysModel`. The string ‘best’ places the anchor point at the location, among the eight locations defined so far, with the shortest distance with the destination `TrnsysModel` (other). The location can also be a 2-tuple giving the coordinates of the origin `TrnsysModel` and the destination `TrnsysModel`.

- **color** (*str* or *tuple*) – The color of the line. Accepts any matplotlib color. You can specify colors in many ways, including full names ('green'), hex strings ('#008000'), RGB or RGBA tuples ((0,1,0,1)) or grayscale intensities as a string ('0.8').
- **linestyle** (*str*) – Possible values: '-' or 'solid', '-' or 'dashed', '-' or 'dashdot', '-' or 'dotted', '-' or 'dashdotdot'.
- **linewidth** (*float*) – The width of the line in points.
- **path** (*LineString* or *MultiLineString*, *optional*) – The path of the link.

**setdefault** ( $k[d]$ )  $\rightarrow$  D.get(k,d), also set D[k]=d if k not in D

property successors

Other objects to which this TypeVariable is connected. Successors.

```
property type_number
```

104 for Type104.

**Type** Return the model's type number, eg.

```
property unit_name
```

‘Type104’.

**Type** Return the model's unit name, eg.

**values** () → an object providing a view on D's values

### 1.4.2 trnsystor.collections.EquationCollection

```
class trnsystor.collections.EquationCollection (mutable=None, name=None,
                                              **kwargs)
```

Behaves like a dict and collects one or more Equations.

This class behaves a little bit like the equation component in the TRNSYS Studio, meaning that you can list equation in a block, give it a name, etc.

You can pass a dict of Equation or you can pass a list of Equation. In this case, the `Equation.name` attribute will be used as a key.

**Hint:** Creating equations in `trnsystor` is done through the `Equation` class. Equations are then collected in this `EquationCollection`. See the `Equation` class for more details.

Since equation blocks don't have a unit number, an incremental negative unit number is given to instantiated `EquationCollections` to ensure that it is compatible with its parent class (see `Component`).

Initialize a new EquationCollection.

### Example

```
>>> equal = Equation.from_expression("TdbAmb = [011,001]")
>>> equa2 = Equation.from_expression("rhAmb = [011,007]")
>>> EquationCollection([equal, equa2])
```

## Parameters

- **mutable** (*Iterable*, *optional*) – An iterable (dict or list).



- **name** (*str*) – A user defined name for this collection of equations. This name will be used to identify this block of equations in the .dck file;

**update** (*E=None, \*\*F*)

Update D from a dict/list/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[ k] = E[k] If E is present and lacks a .keys() method, then does: for eq.name, eq in E: D[eq.name] = eq In either case, this is followed by: for k in F: D[k] = F[k]

#### Parameters

- **E** (*list, dict or Equation*) – The equation to add or update in D ( self).
- **F** (*list, dict or Equation*) – Other Equations to update are passed.

**Returns** None

**property size**

Return len(self).

**property unit\_name**

Return name of self.

This type does not have a unit\_name.

**property model**

This model does not have a proforma. Return class name.

**property centroid**

Returns the model's center Point().

**Type** Point

**clear** () → None. Remove all items from D.

**connect\_to** (*other, mapping=None, link\_style\_kwargs=None*)

Connect the outputs of self to the inputs of other.

---

**Important:** Keep in mind that since python traditionally uses 0-based indexing, the same logic is used in this package even though TRNSYS uses traditionally 1-based indexing. The package will internally handle the 1-based index in the output .dck file.

---

## Examples

Connect two TrnsysModel objects together by creating a mapping of the outputs of pipe\_1 to the inputs of pipe\_2. In this example we connect output\_0 of pipe\_1 to input\_0 of pipe\_2 and output\_1 of pipe\_1 to input\_1 of pipe\_2:

```
>>> pipe_1.connect_to(pipe_2, mapping={0:0, 1:1})
```

The same can be achieved using input/output names.

```
>>> pipe_1.connect_to(pipe_2, mapping={'Outlet_Air_Temperature':
>>> 'Inlet_Air_Temperature', 'Outlet_Air_Humidity_Ratio':
>>> 'Inlet_Air_Humidity_Ratio'})
```

#### Parameters

- **other** (*Component*) – The other object
- **mapping** (*dict*) – Mapping of output to input numbers (or names)
- **link\_style\_kwargs** (*dict*, *optional*) – dict of *LinkStyle* parameters

Raises **TypeError** – A *TypeError* is raised when trying to connect to anything other than a *:class:TrnsysModel*.

**copy** ()

Return copy of self.

**get** (*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to None.

**property inputs**

returns the model's inputs.

Type *InputCollection*

**invalidate\_connections** ()

Iterate over successors and predecessors and remove the edges.

Todo: restore paths in self.studio\_canvas.grid

**property is\_connected**

Whether or not this *Component* is connected to another *TypeVariable*.

Connected to or connected by.

**items** () → a set-like object providing a view on *D*'s items

**keys** () → a set-like object providing a view on *D*'s keys

**property link\_styles**

Return *LinkStyles* of self.

**property outputs**

returns the model's outputs.

Type *OutputCollection*

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised.

**popitem** () → (*k*, *v*), remove and return some (key, value) pair

as a 2-tuple; but raise *KeyError* if *D* is empty.

**property predecessors**

Other objects from which this *TypeVariable* is connected. Predecessors.

**set\_canvas\_position** (*pt*, *trnsys\_coords=False*)

Set position of self in the canvas.

Use cartesian coordinates: origin 0,0 is at bottom-left.

---

**Hint:** The Studio Canvas origin corresponds to the top-left of the canvas. The x coordinates increase from left to right, while the y coordinates increase from top to bottom.

- top-left = “\* \$POSITION 0 0”
- bottom-left = “\* \$POSITION 0 2000”
- top-right = “\* \$POSITION 2000” 0
- bottom-right = “\* \$POSITION 2000 2000”

For convenience, users should deal with cartesian coordinates. trnsystor will deal with the transformation.

### Parameters

- **pt** (*Point* or *2-tuple*) – The Point geometry or a tuple of (x, y) coordinates.
- **trnsys\_coords** (*bool*) – Set to True if pt is given in Trnsys Studio coordinates: origin 0,0 is at top-left.

**set\_component\_layer** (*layers*)

Change the layer of self. Pass a list to change multiple layers.

**set\_link\_style** (*other, loc='best', color='#1f78b4', linestyle='-', linewidth=1, path=None*)

Set outgoing link styles between self and other.

### Parameters

- **other** (*Component*) – The destination model.
- **loc** (*str* or *tuple*) – loc (str): The location of the anchor. The strings ‘top-left’, ‘top-right’, ‘bottom-left’, ‘bottom-right’ place the anchor point at the corresponding corner of the TrnsysModel. The strings ‘top-center’, ‘center-right’, ‘bottom-center’, ‘center-left’ place the anchor point at the edge of the corresponding TrnsysModel. The string ‘best’ places the anchor point at the location, among the eight locations defined so far, with the shortest distance with the destination TrnsysModel (other). The location can also be a 2-tuple giving the coordinates of the origin TrnsysModel and the destination TrnsysModel.
- **color** (*str* or *tuple*) – The color of the line. Accepts any matplotlib color. You can specify colors in many ways, including full names (‘green’), hex strings (‘#008000’), RGB or RGBA tuples ((0,1,0,1)) or grayscale intensities as a string (‘0.8’).
- **linestyle** (*str*) – Possible values: ‘-’ or ‘solid’, ‘-’ or ‘dashed’, ‘-.’ or ‘dashdot’, ‘.’ or ‘dotted’, ‘-.’ or ‘dashdotdot’.
- **linewidth** (*float*) – The width of the line in points.
- **path** (*LineString* or *MultiLineString*, *optional*) – The path of the link.

**setdefault** (*k*, *d*) → D.get(k,d), also set D[k]=d if k not in D

### property successors

Other objects to which this TypeVariable is connected. Successors.

### property type\_number

104 for Type104.

**Type** Return the model’s type number, eg.

### property unit\_number

Return the model’s unit number (unique).

**values** () → an object providing a view on D’s values

### 1.4.3 trnsystor.collections.ComponentCollection

**class** trnsystor.collections.**ComponentCollection** (*initlist=None*)

A class that handles collections of components.

**Supported members:**

- TrnsysModels
- EquationCollections
- ConstantCollections

Get a component from a ComponentCollection using either the component's unit number or its full name.

#### Examples

```
>>> from trnsystor.collections import ComponentCollection
>>> cc = ComponentCollection()
>>> cc.update({tank_type: tank_type})
>>> cc['Storage Tank; Fixed Inlets, Uniform Losses']._unit = 1
>>> cc[1]
Type146: Single Speed Fan/Blower
>>> cc['Single Speed Fan/Blower']
Type146: Single Speed Fan/Blower
```

**property** **iloc**

Access a component by its unit\_number.

**property** **loc**

Access a components by its identify (self).

#### Examples

```
>>> cc = ComponentCollection([tank_type])
>>> assert cc.loc[tank_type] == cc.iloc[tank_type.unit_number]
```

**append** (*item*)

S.append(value) – append value to the end of the sequence

**clear** () → None – remove all items from S

**count** (*value*) → integer – return number of occurrences of value

**extend** (*other*)

S.extend(iterable) – extend sequence by appending elements from the iterable

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (*i*, *item*)

S.insert(index, value) – insert value before index

**pop** ([*index*]) → item – remove and return item at index (default last).

Raise IndexError if list is empty or index is out of range.

**remove** (*item*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**reverse()**  
 S.reverse() – reverse *IN PLACE*

### 1.4.4 trnsystor.collections.ExternalFileCollection

**class** trnsystor.collections.**ExternalFileCollection** (\*\*kwargs)  
 A collection of ExternalFile objects.

**classmethod from\_dict** (dictionary)  
 Construct from a dict of ExternalFile objects.  
 The object's id is used as the key.

**Parameters dictionary** (dict) – The dict of {key: ExternalFile}

**clear()** → None. Remove all items from D.

**get** (k[, d]) → D[k] if k in D, else d. d defaults to None.

**items()** → a set-like object providing a view on D's items

**keys()** → a set-like object providing a view on D's keys

**pop** (k[, d]) → v, remove specified key and return the corresponding value.  
 If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()** → (k, v), remove and return some (key, value) pair  
 as a 2-tuple; but raise KeyError if D is empty.

**setdefault** (k[, d]) → D.get(k,d), also set D[k]=d if k not in D

**update** ([E], \*\*F) → None. Update D from mapping/iterable E and F.  
 If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method,  
 does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

**values()** → an object providing a view on D's values

### 1.4.5 trnsystor.collections.CycleCollection

**class** trnsystor.collections.**CycleCollection** (initlist=None)  
 Collection of *trnsystor.typecycle.TypeCycle*.

**append** (item)  
 S.append(value) – append value to the end of the sequence

**clear()** → None – remove all items from S

**count** (value) → integer – return number of occurrences of value

**extend** (other)  
 S.extend(iterable) – extend sequence by appending elements from the iterable

**index** (value[, start[, stop]]) → integer – return first index of value.  
 Raises ValueError if the value is not present.  
 Supporting start and stop arguments is optional, but recommended.

**insert** (i, item)  
 S.insert(index, value) – insert value before index

**pop** ([index]) → item – remove and return item at index (default last).  
 Raise IndexError if list is empty or index is out of range.

**remove** (*item*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**reverse** ()

S.reverse() – reverse *IN PLACE*

## 1.4.6 trnsystor.collections.VariableCollection

**class** trnsystor.collections.VariableCollection (\*\*kwargs)

A collection of VariableType as a dict.

Handles getting and setting variable values.

**classmethod** from\_dict (*dictionary*)

Return VariableCollection from dict.

Sets also the class attribute using named\_key.

**property** size

The number of variable in the collection.

**clear** () → None. Remove all items from D.

**get** (*k*, *d*) → D[k] if k in D, else d. d defaults to None.

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (*k*, *d*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** () → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

**setdefault** (*k*, *d*) → D.get(k,d), also set D[k]=d if k not in D

**update** ([*E*], \*\**F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

**values** () → an object providing a view on D's values

## 1.4.7 trnsystor.collections.InputCollection

**class** trnsystor.collections.InputCollection (\*\*kwargs)

Subclass of *VariableCollection* specific to Inputs.

---

**Hint:** Iterating over *InputCollection* will not pass Inputs that considered *questions*. For example, Type15 (printer) has a question for the number of variables to be printed by the component. This question can be accessed with `.inputs["How_many_variables_are_to_be_printed_by_this_component_"]` to modify the number of values. But when iterating over the inputs, the question will not be returned in the iterator; only regular inputs will.

---

**property** size

Return the number of inputs excluding questions.

**clear** () → None. Remove all items from D.

**classmethod from\_dict** (*dictionary*)

Return VariableCollection from dict.

Sets also the class attribute using `named_key`.

**get** (*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to None.

**items** () → a set-like object providing a view on *D*'s items

**keys** () → a set-like object providing a view on *D*'s keys

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.  
If key is not found, *d* is returned if given, otherwise `KeyError` is raised.

**popitem** () → (*k*, *v*), remove and return some (key, value) pair  
as a 2-tuple; but raise `KeyError` if *D* is empty.

**setdefault** (*k*, *d*) → *D*.get(*k*,*d*), also set *D*[*k*]=*d* if *k* not in *D*

**update** ([*E*], *\*\*F*) → None. Update *D* from mapping/iterable *E* and *F*.

If *E* present and has a `.keys()` method, does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* present and lacks `.keys()` method,  
does: for (*k*, *v*) in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k*, *v* in *F*.items(): *D*[*k*] = *v*

**values** () → an object providing a view on *D*'s values

## 1.4.8 trnsystor.collections.OutputCollection

**class** trnsystor.collections.**OutputCollection** (*\*\*kwargs*)

Subclass of *VariableCollection* specific to Outputs.

**clear** () → None. Remove all items from *D*.

**classmethod from\_dict** (*dictionary*)

Return VariableCollection from dict.

Sets also the class attribute using `named_key`.

**get** (*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to None.

**items** () → a set-like object providing a view on *D*'s items

**keys** () → a set-like object providing a view on *D*'s keys

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.  
If key is not found, *d* is returned if given, otherwise `KeyError` is raised.

**popitem** () → (*k*, *v*), remove and return some (key, value) pair  
as a 2-tuple; but raise `KeyError` if *D* is empty.

**setdefault** (*k*, *d*) → *D*.get(*k*,*d*), also set *D*[*k*]=*d* if *k* not in *D*

**property size**

The number of variable in the collection.

**update** ([*E*], *\*\*F*) → None. Update *D* from mapping/iterable *E* and *F*.

If *E* present and has a `.keys()` method, does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* present and lacks `.keys()` method,  
does: for (*k*, *v*) in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k*, *v* in *F*.items(): *D*[*k*] = *v*

**values** () → an object providing a view on *D*'s values

### 1.4.9 trnsystor.collections.ParameterCollection

**class** trnsystor.collections.**ParameterCollection** (\*\*kwargs)

Subclass of *VariableCollection* specific to Parameters.

**property size**

Return the number of inputs.

**clear** () → None. Remove all items from D.

**classmethod from\_dict** (dictionary)

Return VariableCollection from dict.

Sets also the class attribute using *named\_key*.

**get** (k[, d]) → D[k] if k in D, else d. d defaults to None.

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (k[, d]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise *KeyError* is raised.

**popitem** () → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise *KeyError* if D is empty.

**setdefault** (k[, d]) → D.get(k,d), also set D[k]=d if k not in D

**update** ([E], \*\*F) → None. Update D from mapping/iterable E and F.

If E present and has a *.keys()* method, does: for k in E: D[k] = E[k] If E present and lacks *.keys()* method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

**values** () → an object providing a view on D's values

## 1.5 Utils

<i>affine_transform</i>	Apply affine transformation to geometry.
<i>get_int_from_rgb</i>	Convert an RBG color to its TRNSYS Studio compatible int color.
<i>get_rgb_from_int</i>	Convert an rgb int color to its red, green and blue colors.
<i>DeckFilePrinter</i>	Print derivative of a function of symbols in deck file form.
<i>print_my_latex</i>	Most of the printers define their own wrappers for <i>print()</i> .
<i>TypeVariableSymbol</i>	This is a subclass of the <i>sympy Symbol</i> class.



### 1.5.1 trnsystor.utils.affine\_transform

`trnsystor.utils.affine_transform(geom, matrix=None)`

Apply affine transformation to geometry.

By, default, flip geometry along the x axis.

---

**Hint:** visit [affine\\_matrix](#) for other affine transformation matrices.

---

#### Parameters

- **geom** (*BaseGeometry*) – The geometry.
- **matrix** (*np.array*) – The coefficient matrix is provided as a list or tuple.

### 1.5.2 trnsystor.utils.get\_int\_from\_rgb

`trnsystor.utils.get_int_from_rgb(rgb)`

Convert an RGB color to its TRNSYS Studio compatible int color.

Values are used ranging from 0 to 255 for each of the components.

---

**Important:** Unlike Java, the TRNSYS Studio will want an integer where bits 0-7 are the blue value, 8-15 the green, and 16-23 the red.

---

#### Examples

Get the rgb int from an rgb 3-tuple

```
>>> get_int_from_rgb((211, 122, 145))
9534163
```

**Parameters** **rgb** (*tuple*) – The red, green and blue values. All values assumed to be in range [0, 255].

**Returns** the rgb int.

**Return type** (*int*)

### 1.5.3 trnsystor.utils.get\_rgb\_from\_int

`trnsystor.utils.get_rgb_from_int(rgb_int)`

Convert an rgb int color to its red, green and blue colors.

Values are used ranging from 0 to 255 for each of the components.

---

**Important:** Unlike Java, the TRNSYS Studio will want an integer where bits 0-7 are the blue value, 8-15 the green, and 16-23 the red.

---

## Examples

Get the rgb tuple from a an rgb int.

```
>>> get_rgb_from_int(9534163)
(211, 122, 145)
```

**Parameters** `rgb_int` (*int*) – An rgb int representation.

**Returns** (r, g, b) tuple.

**Return type** (tuple)

### 1.5.4 trnsystor.utils.DeckFilePrinter

**class** `trnsystor.utils.DeckFilePrinter` (*settings=None*)

Print derivative of a function of symbols in deck file form.

This will override the `sympy.printing.str.StrPrinter#_print_Symbol()` method to print the `TypeVariable`'s `unit_number` and `output_number`.

**doprint** (*expr*)

Returns printer's representation for `expr` (as a string)

**emptyPrinter** (*expr*)

`str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str`

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

**classmethod** `set_global_settings` (*\*\*settings*)

Set system-wide printing settings.

### 1.5.5 trnsystor.utils.print\_my\_latex

`trnsystor.utils.print_my_latex` (*expr*)

Most of the printers define their own wrappers for `print()`.

These wrappers usually take printer settings. Our printer does not have any settings.

### 1.5.6 trnsystor.utils.TypeVariableSymbol

**class** `trnsystor.utils.TypeVariableSymbol` (*type\_variable, \*\*assumptions*)

This is a subclass of the `sympy.Symbol` class.

It is a bit of a hack, so hopefully nothing bad will happen.

`TypeVariableSymbol` are identified by `TypeVariable` and assumptions.

```
>>> from trnsystor.utils import TypeVariableSymbol
>>> TypeVariableSymbol("x") == TypeVariableSymbol("x")
True
>>> TypeVariableSymbol("x", real=True) == TypeVariableSymbol("x",
real=False)
False
```

**Parameters**

- **type\_variable** (`TypeVariable`) – The `TypeVariable` to defined as a Symbol.
- **\*\*assumptions** – See `sympy.core.assumptions` for more details.

**apart** ( $x=None$ , **\*\*args**)

See the `apart` function in `sympy.polys`

**property args**

Returns a tuple of arguments of 'self'.

**Examples**

```
>>> from sympy import cot
>>> from sympy.abc import x, y
```

```
>>> cot(x).args
(x,)
```

```
>>> cot(x).args[0]
x
```

```
>>> (x*y).args
(x, y)
```

```
>>> (x*y).args[1]
y
```

**Notes**

Never use `self._args`, always use `self.args`. Only use `_args` in `__new__` when creating a new function. Don't override `.args()` from `Basic` (so that it's easy to change the interface in the future if needed).

**args\_cnc** ( $cset=False$ ,  $warn=True$ ,  $split_1=True$ )

Return [commutative factors, non-commutative factors] of self.

`self` is treated as a `Mul` and the ordering of the factors is maintained. If `cset` is `True` the commutative factors will be returned in a set. If there were repeated factors (as may happen with an unevaluated `Mul`) then an error will be raised unless it is explicitly suppressed by setting `warn` to `False`.

Note: `-1` is always separated from a `Number` unless `split_1` is `False`.

```
>>> from sympy import symbols, oo
>>> A, B = symbols('A B', commutative=0)
>>> x, y = symbols('x y')
>>> (-2*x*y).args_cnc()
[[-1, 2, x, y], []]
>>> (-2.5*x).args_cnc()
[[-1, 2.5, x], []]
>>> (-2*x*A*B*y).args_cnc()
[[-1, 2, x, y], [A, B]]
>>> (-2*x*A*B*y).args_cnc(split_1=False)
[[-2, x, y], [A, B]]
```

(continues on next page)

(continued from previous page)

```
>>> (-2*x*y).args_cnc(cset=True)
[[-1, 2, x, y], []]
```

The arg is always treated as a Mul:

```
>>> (-2 + x + A).args_cnc()
[[], [x - 2 + A]]
>>> (-oo).args_cnc() # -oo is a singleton
[[-1, oo], []]
```

**as\_coeff\_Add** (*rational=False*)

Efficiently extract the coefficient of a summation.

**as\_coeff\_Mul** (*rational=False*)

Efficiently extract the coefficient of a product.

**as\_coeff\_add** (\*deps)

Return the tuple (c, args) where self is written as an Add, a.

c should be a Rational added to any terms of the Add that are independent of deps.

args should be a tuple of all other terms of a; args is empty if self is a Number or if self is independent of deps (when given).

This should be used when you don't know if self is an Add or not but you want to treat self as an Add or if you want to process the individual arguments of the tail of self as an Add.

- if you know self is an Add and want only the head, use self.args[0];
- if you don't want to process the arguments of the tail but need the tail then use self.as\_two\_terms() which gives the head and tail.
- if you want to split self into an independent and dependent parts use self.as\_independent(\*deps)

```
>>> from sympy import S
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_add()
(3, ())
>>> (3 + x).as_coeff_add()
(3, (x,))
>>> (3 + x + y).as_coeff_add(x)
(y + 3, (x,))
>>> (3 + y).as_coeff_add(x)
(y + 3, ())
```

**as\_coeff\_exponent** (x)

$c*x**e \rightarrow c, e$  where x can be any symbolic expression.

**as\_coeff\_mul** (\*deps, \*\*kwargs)

Return the tuple (c, args) where self is written as a Mul, m.

c should be a Rational multiplied by any factors of the Mul that are independent of deps.

args should be a tuple of all other factors of m; args is empty if self is a Number or if self is independent of deps (when given).

This should be used when you don't know if self is a Mul or not but you want to treat self as a Mul or if you want to process the individual arguments of the tail of self as a Mul.

- if you know self is a Mul and want only the head, use self.args[0];

- if you don't want to process the arguments of the tail but need the tail then use `self.as_two_terms()` which gives the head and tail;
- if you want to split `self` into an independent and dependent parts use `self.as_independent(*deps)`

```
>>> from sympy import S
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_mul()
(3, ())
>>> (3*x*y).as_coeff_mul()
(3, (x, y))
>>> (3*x*y).as_coeff_mul(x)
(3*y, (x,))
>>> (3*y).as_coeff_mul(x)
(3*y, ())
```

### **as\_coefficient** (*expr*)

Extracts symbolic coefficient at the given expression. In other words, this functions separates 'self' into the product of 'expr' and 'expr'-free coefficient. If such separation is not possible it will return None.

### Examples

```
>>> from sympy import E, pi, sin, I, Poly
>>> from sympy.abc import x
```

```
>>> E.as_coefficient(E)
1
>>> (2*E).as_coefficient(E)
2
>>> (2*sin(E)*E).as_coefficient(E)
```

Two terms have E in them so a sum is returned. (If one were desiring the coefficient of the term exactly matching E then the constant from the returned expression could be selected. Or, for greater precision, a method of Poly can be used to indicate the desired term from which the coefficient is desired.)

```
>>> (2*E + x*E).as_coefficient(E)
x + 2
>>> _.args[0] # just want the exact match
2
>>> p = Poly(2*E + x*E); p
Poly(x*E + 2*E, x, E, domain='ZZ')
>>> p.coeff_monomial(E)
2
>>> p.nth(0, 1)
2
```

Since the following cannot be written as a product containing E as a factor, None is returned. (If the coefficient  $2*x$  is desired then the `coeff` method should be used.)

```
>>> (2*E*x + x).as_coefficient(E)
>>> (2*E*x + x).coeff(E)
2*x
```

```
>>> (E*(x + 1) + x).as_coefficient(E)
```

```
>>> (2*pi*I).as_coefficient(pi*I)
2
>>> (2*I).as_coefficient(pi*I)
```

See also:

**coeff** return sum of terms have a given factor

**as\_coeff\_Add** separate the additive constant from an expression

**as\_coeff\_Mul** separate the multiplicative constant from an expression

**as\_independent** separate x-dependent terms/factors from others

**sympy.polys.polytools.Poly.coeff\_monomial** efficiently find the single coefficient of a monomial in Poly

**sympy.polys.polytools.Poly.nth** like coeff\_monomial but powers of monomial terms are used

**as\_coefficients\_dict()**

Return a dictionary mapping terms to their Rational coefficient. Since the dictionary is a defaultdict, inquiries about terms which were not present will return a coefficient of 0. If an expression is not an Add it is considered to have a single term.

## Examples

```
>>> from sympy.abc import a, x
>>> (3*x + a*x + 4).as_coefficients_dict()
{1: 4, x: 3, a*x: 1}
>>> _[a]
0
>>> (3*a*x).as_coefficients_dict()
{a*x: 3}
```

**as\_content\_primitive(radical=False, clear=True)**

This method should recursively remove a Rational from all arguments and return that (content) and the new self (primitive). The content should always be positive and `Mul(*foo.as_content_primitive()) == foo`. The primitive need not be in canonical form and should try to preserve the underlying structure if possible (i.e. `expand_mul` should not be applied to self).

## Examples

```
>>> from sympy import sqrt
>>> from sympy.abc import x, y, z
```

```
>>> eq = 2 + 2*x + 2*y*(3 + 3*y)
```

The `as_content_primitive` function is recursive and retains structure:

```
>>> eq.as_content_primitive()
(2, x + 3*y*(y + 1) + 1)
```

Integer powers will have Rationals extracted from the base:

```
>>> ((2 + 6*x)**2).as_content_primitive()
(4, (3*x + 1)**2)
>>> ((2 + 6*x)**(2*y)).as_content_primitive()
(1, (2*(3*x + 1))**(2*y))
```

Terms may end up joining once their `as_content_primitives` are added:

```
>>> ((5*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(11, x*(y + 1))
>>> ((3*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(9, x*(y + 1))
>>> ((3*(z*(1 + y)) + 2.0*x*(3 + 3*y))).as_content_primitive()
(1, 6.0*x*(y + 1) + 3*z*(y + 1))
>>> ((5*(x*(1 + y)) + 2*x*(3 + 3*y))**2).as_content_primitive()
(121, x**2*(y + 1)**2)
>>> ((x*(1 + y) + 0.4*x*(3 + 3*y))**2).as_content_primitive()
(1, 4.84*x**2*(y + 1)**2)
```

Radical content can also be factored out of the primitive:

```
>>> (2*sqrt(2) + 4*sqrt(10)).as_content_primitive(radical=True)
(2, sqrt(2)*(1 + 2*sqrt(5)))
```

If `clear=False` (default is `True`) then content will not be removed from an `Add` if it can be distributed to leave one or more terms with integer coefficients.

```
>>> (x/2 + y).as_content_primitive()
(1/2, x + 2*y)
>>> (x/2 + y).as_content_primitive(clear=False)
(1, x/2 + y)
```

### `as_dummy()`

Return the expression with any objects having structurally bound symbols replaced with unique, canonical symbols within the object in which they appear and having only the default assumption for commutativity being `True`.

### Examples

```
>>> from sympy import Integral, Symbol
>>> from sympy.abc import x, y
>>> r = Symbol('r', real=True)
>>> Integral(r, (r, x)).as_dummy()
Integral(_0, (_0, x))
>>> _.variables[0].is_real is None
True
```

## Notes

Any object that has structural dummy variables should have a property, *bound\_symbols* that returns a list of structural dummy symbols of the object itself.

Lambda and Subs have bound symbols, but because of how they are cached, they already compare the same regardless of their bound symbols:

```
>>> from sympy import Lambda
>>> Lambda(x, x + 1) == Lambda(y, y + 1)
True
```

### **as\_expr** (\*gens)

Convert a polynomial to a SymPy expression.

## Examples

```
>>> from sympy import sin
>>> from sympy.abc import x, y
```

```
>>> f = (x**2 + x*y).as_poly(x, y)
>>> f.as_expr()
x**2 + x*y
```

```
>>> sin(x).as_expr()
sin(x)
```

### **as\_independent** (\*deps, \*\*hint)

A mostly naive separation of a Mul or Add into arguments that are not dependent on deps. To obtain as complete a separation of variables as possible, use a separation method first, e.g.:

- `separatevars()` to change Mul, Add and Pow (including exp) into Mul
- `.expand(mul=True)` to change Add or Mul into Add
- `.expand(log=True)` to change log expr into an Add

The only non-naive thing that is done here is to respect noncommutative ordering of variables and to always return (0, 0) for *self* of zero regardless of hints.

For nonzero *self*, the returned tuple (i, d) has the following interpretation:

- i will have no variable that appears in deps
- d will either have terms that contain variables that are in deps, or be equal to 0 (when self is an Add) or 1 (when self is a Mul)
- if self is an Add then  $\text{self} = i + d$
- if self is a Mul then  $\text{self} = i * d$
- otherwise (self, S.One) or (S.One, self) is returned.

To force the expression to be treated as an Add, use the hint `as_Add=True`



## Examples

– self is an Add

```
>>> from sympy import sin, cos, exp
>>> from sympy.abc import x, y, z
```

```
>>> (x + x*y).as_independent(x)
(0, x*y + x)
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> (2*x*sin(x) + y + x + z).as_independent(x)
(y + z, 2*x*sin(x) + x)
>>> (2*x*sin(x) + y + x + z).as_independent(x, y)
(z, 2*x*sin(x) + x + y)
```

– self is a Mul

```
>>> (x*sin(x)*cos(y)).as_independent(x)
(cos(y), x*sin(x))
```

non-commutative terms cannot always be separated out when self is a Mul

```
>>> from sympy import symbols
>>> n1, n2, n3 = symbols('n1 n2 n3', commutative=False)
>>> (n1 + n1*n2).as_independent(n2)
(n1, n1*n2)
>>> (n2*n1 + n1*n2).as_independent(n2)
(0, n1*n2 + n2*n1)
>>> (n1*n2*n3).as_independent(n1)
(1, n1*n2*n3)
>>> (n1*n2*n3).as_independent(n2)
(n1, n2*n3)
>>> ((x-n1)*(x-y)).as_independent(x)
(1, (x - y)*(x - n1))
```

– self is anything else:

```
>>> (sin(x)).as_independent(x)
(1, sin(x))
>>> (sin(x)).as_independent(y)
(sin(x), 1)
>>> exp(x+y).as_independent(x)
(1, exp(x + y))
```

– force self to be treated as an Add:

```
>>> (3*x).as_independent(x, as_Add=True)
(0, 3*x)
```

– force self to be treated as a Mul:

```
>>> (3+x).as_independent(x, as_Add=False)
(1, x + 3)
>>> (-3+x).as_independent(x, as_Add=False)
(1, x - 3)
```

Note how the below differs from the above in making the constant on the dep term positive.

```
>>> (y*(-3+x)).as_independent(x)
(y, x - 3)
```

– use `.as_independent()` for true independence testing instead of `.has()`. The former considers only symbols in the free symbols while the latter considers all symbols

```
>>> from sympy import Integral
>>> I = Integral(x, (x, 1, 2))
>>> I.has(x)
True
>>> x in I.free_symbols
False
>>> I.as_independent(x) == (I, 1)
True
>>> (I + x).as_independent(x) == (I, x)
True
```

Note: when trying to get independent terms, a separation method might need to be used first. In this case, it is important to keep track of what you send to this routine so you know how to interpret the returned values

```
>>> from sympy import separatevars, log
>>> separatevars(exp(x+y)).as_independent(x)
(exp(y), exp(x))
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> separatevars(x + x*y).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).expand(mul=True).as_independent(y)
(x, x*y)
>>> a, b=symbols('a b', positive=True)
>>> (log(a*b).expand(log=True)).as_independent(b)
(log(a), log(b))
```

**See also:**

`separatevars`, `expand`, `sympy.core.add.Add.as_two_terms`, `sympy.core.mul.Mul.as_two_terms`, `as_coeff_add`, `as_coeff_mul`

**`as_leading_term(*symbols)`**

Returns the leading (nonzero) term of the series expansion of self.

The `_eval_as_leading_term` routines are used to do this, and they must always return a non-zero value.

## Examples

```
>>> from sympy.abc import x
>>> (1 + x + x**2).as_leading_term(x)
1
>>> (1/x**2 + x + x**2).as_leading_term(x)
x**(-2)
```

**as\_numer\_denom()**  
expression -> a/b -> a, b

This is just a stub that should be defined by an object's class methods to get anything else.

**See also:**

**normal** return a/b instead of a, b

**as\_ordered\_factors** (order=None)  
Return list of ordered factors (if Mul) else [self].

**as\_ordered\_terms** (order=None, data=False)  
Transform an expression to an ordered list of terms.

## Examples

```
>>> from sympy import sin, cos
>>> from sympy.abc import x
```

```
>>> (sin(x)**2*cos(x) + sin(x)**2 + 1).as_ordered_terms()
[sin(x)**2*cos(x), sin(x)**2, 1]
```

**as\_poly** (\*gens, \*\*args)  
Converts self to a polynomial or returns None.

```
>>> from sympy import sin
>>> from sympy.abc import x, y
```

```
>>> print((x**2 + x*y).as_poly())
Poly(x**2 + x*y, x, y, domain='ZZ')
```

```
>>> print((x**2 + x*y).as_poly(x, y))
Poly(x**2 + x*y, x, y, domain='ZZ')
```

```
>>> print((x**2 + sin(y)).as_poly(x, y))
None
```

**as\_powers\_dict** ()  
Return self as a dictionary of factors with each factor being treated as a power. The keys are the bases of the factors and the values, the corresponding exponents. The resulting dictionary should be used with caution if the expression is a Mul and contains non-commutative factors since the order that they appeared will be lost in the dictionary.

**See also:**

**as\_ordered\_factors** An alternative for noncommutative applications, returning an ordered list of factors.

**args\_cnc** Similar to `as_ordered_factors`, but guarantees separation of commutative and noncommutative factors.

**as\_real\_imag** (*deep=True, \*\*hints*)

Performs complex expansion on 'self' and returns a tuple containing collected both real and imaginary parts. This method can't be confused with `re()` and `im()` functions, which does not perform complex expansion at evaluation.

However it is possible to expand both `re()` and `im()` functions and get exactly the same results as with a single call to this function.

```
>>> from sympy import symbols, I
```

```
>>> x, y = symbols('x,y', real=True)
```

```
>>> (x + y*I).as_real_imag()
(x, y)
```

```
>>> from sympy.abc import z, w
```

```
>>> (z + w*I).as_real_imag()
(re(z) - im(w), re(w) + im(z))
```

**as\_set** ()

Rewrites Boolean expression in terms of real sets.

## Examples

```
>>> from sympy import Symbol, Eq, Or, And
>>> x = Symbol('x', real=True)
>>> Eq(x, 0).as_set()
FiniteSet(0)
>>> (x > 0).as_set()
Interval.open(0, oo)
>>> And(-2 < x, x < 2).as_set()
Interval.open(-2, 2)
>>> Or(x < -2, 2 < x).as_set()
Union(Interval.open(-oo, -2), Interval.open(2, oo))
```

**as\_terms** ()

Transform an expression to a list of terms.

**aseries** (*x=None, n=6, bound=0, hir=False*)

Asymptotic Series expansion of self. This is equivalent to `self.series(x, oo, n)`.

### Parameters

- **self** (*Expression*) – The expression whose series is to be expanded.
- **x** (*Symbol*) – It is the variable of the expression to be calculated.
- **n** (*Value*) – The number of terms upto which the series is to be expanded.
- **hir** (*Boolean*) – Set this parameter to be True to produce hierarchical series. It stops the recursion at an early level and may provide nicer and more useful results.

- **bound** (*Value*, *Integer*) – Use the bound parameter to give limit on rewriting coefficients in its normalised form.

## Examples

```
>>> from sympy import sin, exp
>>> from sympy.abc import x, y
```

```
>>> e = sin(1/x + exp(-x)) - sin(1/x)
```

```
>>> e.aseries(x)
(1/(24*x**4) - 1/(2*x**2) + 1 + O(x**(-6), (x, oo)))*exp(-x)
```

```
>>> e.aseries(x, n=3, hir=True)
-exp(-2*x)*sin(1/x)/2 + exp(-x)*cos(1/x) + O(exp(-3*x), (x, oo))
```

```
>>> e = exp(exp(x)/(1 - 1/x))
```

```
>>> e.aseries(x)
exp(exp(x)/(1 - 1/x))
```

```
>>> e.aseries(x, bound=3)
exp(exp(x)/x**2)*exp(exp(x)/x)*exp(-exp(x) + exp(x)/(1 - 1/x) - exp(x)/x -
↳exp(x)/x**2)*exp(exp(x))
```

**Returns** Asymptotic series expansion of the expression.

**Return type** Expr

## Notes

This algorithm is directly induced from the limit computational algorithm provided by Gruntz. It majorly uses the mrv and rewrite sub-routines. The overall idea of this algorithm is first to look for the most rapidly varying subexpression  $w$  of a given expression  $f$  and then expands  $f$  in a series in  $w$ . Then same thing is recursively done on the leading coefficient till we get constant coefficients.

If the most rapidly varying subexpression of a given expression  $f$  is  $f$  itself, the algorithm tries to find a normalised representation of the mrv set and rewrites  $f$  using this normalised representation.

If the expansion contains an order term, it will be either  $O(x ** (-n))$  or  $O(w ** (-n))$  where  $w$  belongs to the most rapidly varying expression of `self`.

## References

**See also:**

**Expr.aseries** See the docstring of this function for complete details of this wrapper.

**property assumptions0**

Return object *type* assumptions.

For example:

Symbol('x', real=True) Symbol('x', integer=True)

are different objects. In other words, besides Python type (Symbol in this case), the initial assumptions are also forming their typeinfo.

## Examples

```
>>> from sympy import Symbol
>>> from sympy.abc import x
>>> x.assumptions0
{'commutative': True}
>>> x = Symbol("x", positive=True)
>>> x.assumptions0
{'commutative': True, 'complex': True, 'extended_negative': False,
'extended_nonnegative': True, 'extended_nonpositive': False,
'extended_nonzero': True, 'extended_positive': True, 'extended_real':
True, 'finite': True, 'hermitian': True, 'imaginary': False,
'infinite': False, 'negative': False, 'nonnegative': True,
'nonpositive': False, 'nonzero': True, 'positive': True, 'real':
True, 'zero': False}
```

### atoms (\*types)

Returns the atoms that form the current object.

By default, only objects that are truly atomic and can't be divided into smaller pieces are returned: symbols, numbers, and number symbols like I and pi. It is possible to request atoms of any type, however, as demonstrated below.

## Examples

```
>>> from sympy import I, pi, sin
>>> from sympy.abc import x, y
>>> (1 + x + 2*sin(y + I*pi)).atoms()
{1, 2, I, pi, x, y}
```

If one or more types are given, the results will contain only those types of atoms.

```
>>> from sympy import Number, NumberSymbol, Symbol
>>> (1 + x + 2*sin(y + I*pi)).atoms(Symbol)
{x, y}
```

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Number)
{1, 2}
```

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol)
{1, 2, pi}
```

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol, I)
{1, 2, I, pi}
```

Note that I (imaginary unit) and zoo (complex infinity) are special types of number symbols and are not part of the NumberSymbol class.

The type can be given implicitly, too:

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(x) # x is a Symbol
{x, y}
```

Be careful to check your assumptions when using the implicit option since `S(1).is_Integer = True` but `type(S(1))` is `One`, a special type of sympy atom, while `type(S(2))` is type `Integer` and will find all integers in an expression:

```
>>> from sympy import S
>>> (1 + x + 2*sin(y + I*pi)).atoms(S(1))
{1}
```

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(S(2))
{1, 2}
```

Finally, arguments to `atoms()` can select more than atomic atoms: any sympy type (loaded in `core/_init_.py`) can be listed as an argument and those types of “atoms” as found in scanning the arguments of the expression recursively:

```
>>> from sympy import Function, Mul
>>> from sympy.core.function import AppliedUndef
>>> f = Function('f')
>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(Function)
{f(x), sin(y + I*pi)}
>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(AppliedUndef)
{f(x)}
```

```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Mul)
{I*pi, 2*sin(y + I*pi)}
```

### property `binary_symbols`

Return from the atoms of self those which are free symbols.

For most expressions, all symbols are free symbols. For some classes this is not true. e.g. Integrals use Symbols for the dummy variables which are bound variables, so `Integral` has a method to return all symbols except those. `Derivative` keeps track of symbols with respect to which it will perform a derivative; those are bound variables, too, so it has its own `free_symbols` method.

Any other method that uses bound variables should implement a `free_symbols` method.

### method `cancel(*gens, **args)`

See the `cancel` function in `sympy.polys`

### property `canonical_variables`

Return a dictionary mapping any variable defined in `self.bound_symbols` to Symbols that do not clash with any existing symbol in the expression.

## Examples

```
>>> from sympy import Lambda
>>> from sympy.abc import x
>>> Lambda(x, 2*x).canonical_variables
{x: _0}
```

### classmethod `class_key()`

Nice order of classes.

**coeff** (*x*, *n=1*, *right=False*)

Returns the coefficient from the term(s) containing  $x**n$ . If *n* is zero then all terms independent of *x* will be returned.

When *x* is noncommutative, the coefficient to the left (default) or right of *x* can be returned. The keyword 'right' is ignored when *x* is commutative.

**See also:**

**as\_coefficient** separate the expression into a coefficient and factor

**as\_coeff\_Add** separate the additive constant from an expression

**as\_coeff\_Mul** separate the multiplicative constant from an expression

**as\_independent** separate *x*-dependent terms/factors from others

**sympy.polys.polytools.Poly.coeff\_monomial** efficiently find the single coefficient of a monomial in Poly

**sympy.polys.polytools.Poly.nth** like **coeff\_monomial** but powers of monomial terms are used

## Examples

```
>>> from sympy import symbols
>>> from sympy.abc import x, y, z
```

You can select terms that have an explicit negative in front of them:

```
>>> (-x + 2*y).coeff(-1)
x
>>> (x - 2*y).coeff(-1)
2*y
```

You can select terms with no Rational coefficient:

```
>>> (x + 2*y).coeff(1)
x
>>> (3 + 2*x + 4*x**2).coeff(1)
0
```

You can select terms independent of *x* by making *n=0*; in this case `expr.as_independent(x)[0]` is returned (and 0 will be returned instead of None):

```
>>> (3 + 2*x + 4*x**2).coeff(x, 0)
3
>>> eq = ((x + 1)**3).expand() + 1
>>> eq
x**3 + 3*x**2 + 3*x + 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 2]
>>> eq -= 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 0]
```

You can select terms that have a numerical term in front of them:



```
>>> (-x - 2*y).coeff(2)
-y
>>> from sympy import sqrt
>>> (x + sqrt(2)*x).coeff(sqrt(2))
x
```

The matching is exact:

```
>>> (3 + 2*x + 4*x**2).coeff(x)
2
>>> (3 + 2*x + 4*x**2).coeff(x**2)
4
>>> (3 + 2*x + 4*x**2).coeff(x**3)
0
>>> (z*(x + y)**2).coeff((x + y)**2)
z
>>> (z*(x + y)**2).coeff(x + y)
0
```

In addition, no factoring is done, so  $1 + z*(1 + y)$  is not obtained from the following:

```
>>> (x + z*(x + x*y)).coeff(x)
1
```

If such factoring is desired, `factor_terms` can be used first:

```
>>> from sympy import factor_terms
>>> factor_terms(x + z*(x + x*y)).coeff(x)
z*(y + 1) + 1
```

```
>>> n, m, o = symbols('n m o', commutative=False)
>>> n.coeff(n)
1
>>> (3*n).coeff(n)
3
>>> (n*m + m*n*m).coeff(n) # = (1 + m)*n*m
1 + m
>>> (n*m + m*n*m).coeff(n, right=True) # = (1 + m)*n*m
m
```

If there is more than one possible coefficient 0 is returned:

```
>>> (n*m + m*n).coeff(n)
0
```

If there is only one possible coefficient, it is returned:

```
>>> (n*m + x*m*n).coeff(m*n)
x
>>> (n*m + x*m*n).coeff(m*n, right=1)
1
```

**collect** (*syms, func=None, evaluate=True, exact=False, distribute\_order\_term=True*)

See the collect function in `sympy.simplify`

**combsimp** ()

See the combsimp function in `sympy.simplify`

**compare** (*other*)

Return -1, 0, 1 if the object is smaller, equal, or greater than other.

Not in the mathematical sense. If the object is of a different type from the “other” then their classes are ordered according to the sorted\_classes list.

**Examples**

```
>>> from sympy.abc import x, y
>>> x.compare(y)
-1
>>> x.compare(x)
0
>>> y.compare(x)
1
```

**compute\_leading\_term** (*x*, *logx=None*)

as\_leading\_term is only allowed for results of .series() This is a wrapper to compute a series first.

**could\_extract\_minus\_sign** ()

Return True if self is not in a canonical form with respect to its sign.

For most expressions, e, there will be a difference in e and -e. When there is, True will be returned for one and False for the other; False will be returned if there is no difference.

**Examples**

```
>>> from sympy.abc import x, y
>>> e = x - y
>>> {i.could_extract_minus_sign() for i in (e, -e)}
{False, True}
```

**count** (*query*)

Count the number of matching subexpressions.

**count\_ops** (*visual=None*)

wrapper for count\_ops that returns the operation count.

**doit** (*\*\*hints*)

Evaluate objects that are not evaluated by default like limits, integrals, sums and products. All objects of this kind will be evaluated recursively, unless some species were excluded via ‘hints’ or unless the ‘deep’ hint was set to ‘False’.

```
>>> from sympy import Integral
>>> from sympy.abc import x
```

```
>>> 2*Integral(x, x)
2*Integral(x, x)
```

```
>>> (2*Integral(x, x)).doit()
x**2
```

```
>>> (2*Integral(x, x)).doit(deep=False)
2*Integral(x, x)
```

**dummy\_eq** (*other*, *symbol=None*)

Compare two expressions and handle dummy symbols.

### Examples

```
>>> from sympy import Dummy
>>> from sympy.abc import x, y
```

```
>>> u = Dummy('u')
```

```
>>> (u**2 + 1).dummy_eq(x**2 + 1)
True
>>> (u**2 + 1) == (x**2 + 1)
False
```

```
>>> (u**2 + y).dummy_eq(x**2 + y, x)
True
>>> (u**2 + y).dummy_eq(x**2 + y, y)
False
```

**equals** (*other*, *failing\_expression=False*)

Return True if self == other, False if it doesn't, or None. If failing\_expression is True then the expression which did not simplify to a 0 will be returned instead of None.

If self is a Number (or complex number) that is not zero, then the result is False.

If self is a number and has not evaluated to zero, evalf will be used to test whether the expression evaluates to zero. If it does so and the result has significance (i.e. the precision is either -1, for a Rational result, or is greater than 1) then the evalf value will be used to return True or False.

**evalf** (*n=15*, *subs=None*, *maxn=100*, *chop=False*, *strict=False*, *quad=None*, *verbose=False*)

Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

**subs=<dict>** Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}. The substitutions must be given as a dictionary.

**maxn=<integer>** Allow a maximum temporary working precision of maxn digits (default=100)

**chop=<bool>** Replace tiny real or imaginary parts in subresults by exact zeros (default=False)

**strict=<bool>** Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)

**quad=<str>** Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.

**verbose=<bool>** Print debug information (default=False)

## Notes

When Floats are naively substituted into an expression, precision errors may adversely affect the result. For example, adding  $1e16$  (a Float) to 1 will truncate to  $1e16$ ; if  $1e16$  is then subtracted, the result will be 0. That is exactly what happens in the following:

```
>>> from sympy.abc import x, y, z
>>> values = {x: 1e16, y: 1, z: 1e16}
>>> (x + y - z).subs(values)
0
```

Using the subs argument for evalf is the accurate way to evaluate such an expression:

```
>>> (x + y - z).evalf(subs=values)
1.0000000000000000
```

**expand** (*deep=True, modulus=None, power\_base=True, power\_exp=True, mul=True, log=True, multinomial=True, basic=True, \*\*hints*)  
Expand an expression using hints.

See the docstring of the `expand()` function in `sympy.core.function` for more information.

**property expr\_free\_symbols**

Like `free_symbols`, but returns the free symbols only if they are contained in an expression node.

## Examples

```
>>> from sympy.abc import x, y
>>> (x + y).expr_free_symbols
{x, y}
```

If the expression is contained in a non-expression object, don't return the free symbols. Compare:

```
>>> from sympy import Tuple
>>> t = Tuple(x + y)
>>> t.expr_free_symbols
set()
>>> t.free_symbols
{x, y}
```

**extract\_additively** (*c*)

Return `self - c` if it's possible to subtract `c` from `self` and make all matching coefficients move towards zero, else return `None`.

## Examples

```
>>> from sympy.abc import x, y
>>> e = 2*x + 3
>>> e.extract_additively(x + 1)
x + 2
>>> e.extract_additively(3*x)
>>> e.extract_additively(4)
>>> (y*(x + 1)).extract_additively(x + 1)
>>> ((x + 1)*(x + 2*y + 1) + 3).extract_additively(x + 1)
(x + 1)*(x + 2*y) + 3
```

Sometimes auto-expansion will return a less simplified result than desired; `gcd_terms` might be used in such cases:

```
>>> from sympy import gcd_terms
>>> (4*x*(y + 1) + y).extract_additively(x)
4*x*(y + 1) + x*(4*y + 3) - x*(4*y + 4) + y
>>> gcd_terms(_)
x*(4*y + 3) + y
```

See also:

*`extract_multiplicatively`, `coeff`, `as_coefficient`*

**`extract_branch_factor`** (*allow\_half=False*)

Try to write self as  $\exp_{\text{polar}}(2\pi i n) * z$  in a nice way. Return (z, n).

```
>>> from sympy import exp_polar, I, pi
>>> from sympy.abc import x, y
>>> exp_polar(I*pi).extract_branch_factor()
(exp_polar(I*pi), 0)
>>> exp_polar(2*I*pi).extract_branch_factor()
(1, 1)
>>> exp_polar(-pi*I).extract_branch_factor()
(exp_polar(I*pi), -1)
>>> exp_polar(3*pi*I + x).extract_branch_factor()
(exp_polar(x + I*pi), 1)
>>> (y*exp_polar(-5*pi*I)*exp_polar(3*pi*I + 2*pi*x)).extract_branch_factor()
(y*exp_polar(2*pi*x), -1)
>>> exp_polar(-I*pi/2).extract_branch_factor()
(exp_polar(-I*pi/2), 0)
```

If `allow_half` is `True`, also extract  $\exp_{\text{polar}}(I\pi)$ :

```
>>> exp_polar(I*pi).extract_branch_factor(allow_half=True)
(1, 1/2)
>>> exp_polar(2*I*pi).extract_branch_factor(allow_half=True)
(1, 1)
>>> exp_polar(3*I*pi).extract_branch_factor(allow_half=True)
(1, 3/2)
>>> exp_polar(-I*pi).extract_branch_factor(allow_half=True)
(1, -1/2)
```

**`extract_multiplicatively`** (*c*)

Return `None` if it's not possible to make self in the form  $c * \text{something}$  in a nice way, i.e. preserving the properties of arguments of self.

## Examples

```
>>> from sympy import symbols, Rational
```

```
>>> x, y = symbols('x,y', real=True)
```

```
>>> ((x*y)**3).extract_multiplicatively(x**2 * y)
x*y**2
```

```
>>> ((x*y)**3).extract_multiplicatively(x**4 * y)
```

```
>>> (2*x).extract_multiplicatively(2)
x
```

```
>>> (2*x).extract_multiplicatively(3)
```

```
>>> (Rational(1, 2)*x).extract_multiplicatively(3)
x/6
```

**factor** (\*gens, \*\*args)

See the factor() function in sympy.polys.polytools

**find** (query, group=False)

Find all subexpressions matching a query.

**fourier\_series** (limits=None)

Compute fourier sine/cosine series of self.

See the docstring of the `fourier_series()` in sympy.series.fourier for more information.

**fps** (x=None, x0=0, dir=1, hyper=True, order=4, rational=True, full=False)

Compute formal power power series of self.

See the docstring of the `fps()` function in sympy.series.formal for more information.

**property free\_symbols**

Return from the atoms of self those which are free symbols.

For most expressions, all symbols are free symbols. For some classes this is not true. e.g. Integrals use Symbols for the dummy variables which are bound variables, so Integral has a method to return all symbols except those. Derivative keeps track of symbols with respect to which it will perform a derivative; those are bound variables, too, so it has its own free\_symbols method.

Any other method that uses bound variables should implement a free\_symbols method.

**classmethod fromiter** (args, \*\*assumptions)

Create a new object from an iterable.

This is a convenience function that allows one to create objects from any iterable, without having to convert to a list or tuple first.

## Examples

```
>>> from sympy import Tuple
>>> Tuple.fromiter(i for i in range(5))
(0, 1, 2, 3, 4)
```

**property func**

The top-level function in an expression.

The following should hold for all objects:

```
>> x == x.func(*x.args)
```

## Examples

```
>>> from sympy.abc import x
>>> a = 2*x
>>> a.func
<class 'sympy.core.mul.Mul'>
>>> a.args
(2, x)
>>> a.func(*a.args)
2*x
>>> a == a.func(*a.args)
True
```

### **gammasimp()**

See the gammasimp function in sympy.simplify

### **getO()**

Returns the additive  $O(\cdot)$  symbol if there is one, else None.

### **getn()**

Returns the order of the expression.

The order is determined either from the  $O(\dots)$  term. If there is no  $O(\dots)$  term, it returns None.

## Examples

```
>>> from sympy import O
>>> from sympy.abc import x
>>> (1 + x + O(x**2)).getn()
2
>>> (1 + x).getn()
```

### **has(\*patterns)**

Test whether any subexpression matches any of the patterns.

## Examples

```
>>> from sympy import sin
>>> from sympy.abc import x, y, z
>>> (x**2 + sin(x*y)).has(z)
False
>>> (x**2 + sin(x*y)).has(x, y, z)
True
>>> x.has(x)
True
```

Note has is a structural algorithm with no knowledge of mathematics. Consider the following half-open interval:

```
>>> from sympy.sets import Interval
>>> i = Interval.Lopen(0, 5); i
Interval.Lopen(0, 5)
>>> i.args
(0, 5, True, False)
>>> i.has(4) # there is no "4" in the arguments
```

(continues on next page)

(continued from previous page)

```
False
>>> i.has(0)  # there is a "0" in the arguments
True
```

Instead, use `contains` to determine whether a number is in the interval or not:

```
>>> i.contains(4)
True
>>> i.contains(0)
False
```

Note that `expr.has(*patterns)` is exactly equivalent to `any(expr.has(p) for p in patterns)`. In particular, `False` is returned when the list of patterns is empty.

```
>>> x.has()
False
```

**integrate** (\*args, \*\*kwargs)

See the `integrate` function in `sympy.integrals`

**invert** (g, \*gens, \*\*args)

Return the multiplicative inverse of `self mod g` where `self` (and `g`) may be symbolic expressions).

See also:

`sympy.core.numbers.mod_inverse`, `sympy.polys.polytools.invert`

**is\_algebraic\_expr** (\*syms)

This tests whether a given expression is algebraic or not, in the given symbols, `syms`. When `syms` is not given, all free symbols will be used. The rational function does not have to be in expanded or in any kind of canonical form.

This function returns `False` for expressions that are “algebraic expressions” with symbolic exponents. This is a simple extension to the `is_rational_function`, including rational exponentiation.

## Examples

```
>>> from sympy import Symbol, sqrt
>>> x = Symbol('x', real=True)
>>> sqrt(1 + x).is_rational_function()
False
>>> sqrt(1 + x).is_algebraic_expr()
True
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be an algebraic expression to become one.

```
>>> from sympy import exp, factor
>>> a = sqrt(exp(x)**2 + 2*exp(x) + 1) / (exp(x) + 1)
>>> a.is_algebraic_expr(x)
False
>>> factor(a).is_algebraic_expr()
True
```

See also:

`is_rational_function`



## References

- [https://en.wikipedia.org/wiki/Algebraic\\_expression](https://en.wikipedia.org/wiki/Algebraic_expression)

### `is_constant (*wrt, **flags)`

Return True if self is constant, False if not, or None if the constancy could not be determined conclusively.

If an expression has no free symbols then it is a constant. If there are free symbols it is possible that the expression is a constant, perhaps (but not necessarily) zero. To test such expressions, a few strategies are tried:

1) numerical evaluation at two random points. If two such evaluations give two different values and the values have a precision greater than 1 then self is not constant. If the evaluations agree or could not be obtained with any precision, no decision is made. The numerical testing is done only if `wrt` is different than the free symbols.

2) differentiation with respect to variables in 'wrt' (or all free symbols if omitted) to see if the expression is constant or not. This will not always lead to an expression that is zero even though an expression is constant (see added test in `test_expr.py`). If all derivatives are zero then self is constant with respect to the given symbols.

3) finding out zeros of denominator expression with `free_symbols`. It won't be constant if there are zeros. It gives more negative answers for expression that are not constant.

If neither evaluation nor differentiation can prove the expression is constant, None is returned unless two numerical values happened to be the same and the flag `failing_number` is True – in that case the numerical value will be returned.

If flag `simplify=False` is passed, self will not be simplified; the default is True since self should be simplified before testing.

## Examples

```
>>> from sympy import cos, sin, Sum, S, pi
>>> from sympy.abc import a, n, x, y
>>> x.is_constant()
False
>>> S(2).is_constant()
True
>>> Sum(x, (x, 1, 10)).is_constant()
True
>>> Sum(x, (x, 1, n)).is_constant()
False
>>> Sum(x, (x, 1, n)).is_constant(y)
True
>>> Sum(x, (x, 1, n)).is_constant(n)
False
>>> Sum(x, (x, 1, n)).is_constant(x)
True
>>> eq = a*cos(x)**2 + a*sin(x)**2 - a
>>> eq.is_constant()
True
>>> eq.subs({x: pi, a: 2}) == eq.subs({x: pi, a: 3}) == 0
True
```

```
>>> (0**x).is_constant()
False
```

(continues on next page)

(continued from previous page)

```

>>> x.is_constant()
False
>>> (x**x).is_constant()
False
>>> one = cos(x)**2 + sin(x)**2
>>> one.is_constant()
True
>>> ((one - 1)**(x + 1)).is_constant() in (True, False) # could be 0 or 1
True

```

**is\_polynomial(\*syms)**

Return True if self is a polynomial in syms and False otherwise.

This checks if self is an exact polynomial in syms. This function returns False for expressions that are “polynomials” with symbolic exponents. Thus, you should be able to apply polynomial algorithms to expressions for which this returns True, and Poly(expr, \*syms) should work if and only if expr.is\_polynomial(\*syms) returns True. The polynomial does not have to be in expanded form. If no symbols are given, all free symbols in the expression will be used.

This is not part of the assumptions system. You cannot do Symbol('z', polynomial=True).

**Examples**

```

>>> from sympy import Symbol
>>> x = Symbol('x')
>>> ((x**2 + 1)**4).is_polynomial(x)
True
>>> ((x**2 + 1)**4).is_polynomial()
True
>>> (2**x + 1).is_polynomial(x)
False

```

```

>>> n = Symbol('n', nonnegative=True, integer=True)
>>> (x**n + 1).is_polynomial(x)
False

```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a polynomial to become one.

```

>>> from sympy import sqrt, factor, cancel
>>> y = Symbol('y', positive=True)
>>> a = sqrt(y**2 + 2*y + 1)
>>> a.is_polynomial(y)
False
>>> factor(a)
y + 1
>>> factor(a).is_polynomial(y)
True

```

```

>>> b = (y**2 + 2*y + 1)/(y + 1)
>>> b.is_polynomial(y)
False
>>> cancel(b)
y + 1

```

(continues on next page)

(continued from previous page)

```
>>> cancel(b).is_polynomial(y)
True
```

See also `.is_rational_function()`

**is\_rational\_function** (\*syms)

Test whether function is a ratio of two polynomials in the given symbols, syms. When syms is not given, all free symbols will be used. The rational function does not have to be in expanded or in any kind of canonical form.

This function returns False for expressions that are “rational functions” with symbolic exponents. Thus, you should be able to call `.as_numer_denom()` and apply polynomial algorithms to the result for expressions for which this returns True.

This is not part of the assumptions system. You cannot do `Symbol('z', rational_function=True)`.

## Examples

```
>>> from sympy import Symbol, sin
>>> from sympy.abc import x, y
```

```
>>> (x/y).is_rational_function()
True
```

```
>>> (x**2).is_rational_function()
True
```

```
>>> (x/sin(y)).is_rational_function(y)
False
```

```
>>> n = Symbol('n', integer=True)
>>> (x**n + 1).is_rational_function(x)
False
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a rational function to become one.

```
>>> from sympy import sqrt, factor
>>> y = Symbol('y', positive=True)
>>> a = sqrt(y**2 + 2*y + 1)/y
>>> a.is_rational_function(y)
False
>>> factor(a)
(y + 1)/y
>>> factor(a).is_rational_function(y)
True
```

See also `is_algebraic_expr()`.

**leadterm** (x)

Returns the leading term  $a*x**b$  as a tuple (a, b).

## Examples

```
>>> from sympy.abc import x
>>> (1+x+x**2).leadterm(x)
(1, 0)
>>> (1/x**2+x+x**2).leadterm(x)
(1, -2)
```

**limit** (*x*, *xlim*, *dir*='+')

Compute limit  $x \rightarrow xlim$ .

**lseries** (*x=None*, *x0=0*, *dir*='+', *logx=None*)

Wrapper for series yielding an iterator of the terms of the series.

Note: an infinite series will yield an infinite iterator. The following, for example, will never terminate. It will just keep printing terms of the  $\sin(x)$  series:

```
for term in sin(x).lseries(x):
    print term
```

The advantage of `lseries()` over `nseries()` is that many times you are just interested in the next term in the series (i.e. the first term for example), but you don't know how many you should ask for in `nseries()` using the "n" parameter.

See also `nseries()`.

**match** (*pattern*, *old=False*)

Pattern matching.

Wild symbols match all.

Return `None` when expression (*self*) does not match with pattern. Otherwise return a dictionary such that:

```
pattern.xreplace(self.match(pattern)) == self
```

## Examples

```
>>> from sympy import Wild
>>> from sympy.abc import x, y
>>> p = Wild("p")
>>> q = Wild("q")
>>> r = Wild("r")
>>> e = (x+y)**(x+y)
>>> e.match(p**p)
{p_: x + y}
>>> e.match(p**q)
{p_: x + y, q_: x + y}
>>> e = (2*x)**2
>>> e.match(p*q**r)
{p_: 4, q_: x, r_: 2}
>>> (p*q**r).xreplace(e.match(p*q**r))
4*x**2
```

The `old` flag will give the old-style pattern matching where expressions and patterns are essentially solved to give the match. Both of the following give `None` unless `old=True`:

```
>>> (x - 2).match(p - x, old=True)
{p_: 2*x - 2}
>>> (2/x).match(p*x, old=True)
{p_: 2/x**2}
```

**matches** (*expr*, *repl\_dict*={}, *old*=False)

Helper method for match() that looks for a match between Wild symbols in self and expressions in expr.

## Examples

```
>>> from sympy import symbols, Wild, Basic
>>> a, b, c = symbols('a b c')
>>> x = Wild('x')
>>> Basic(a + x, x).matches(Basic(a + b, c)) is None
True
>>> Basic(a + x, x).matches(Basic(a + b + c, b + c))
{x_: b + c}
```

**n** (*n*=15, *subs*=None, *maxn*=100, *chop*=False, *strict*=False, *quad*=None, *verbose*=False)

Evaluate the given formula to an accuracy of *n* digits. Optional keyword arguments:

**subs**=<dict> Substitute numerical values for symbols, e.g. `subs={x:3, y:1+pi}`. The substitutions must be given as a dictionary.

**maxn**=<integer> Allow a maximum temporary working precision of *maxn* digits (default=100)

**chop**=<bool> Replace tiny real or imaginary parts in subresults by exact zeros (default=False)

**strict**=<bool> Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available *maxprec* (default=False)

**quad**=<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try `quad='osc'`.

**verbose**=<bool> Print debug information (default=False)

## Notes

When Floats are naively substituted into an expression, precision errors may adversely affect the result. For example, adding `1e16` (a Float) to 1 will truncate to `1e16`; if `1e16` is then subtracted, the result will be 0. That is exactly what happens in the following:

```
>>> from sympy.abc import x, y, z
>>> values = {x: 1e16, y: 1, z: 1e16}
>>> (x + y - z).subs(values)
0
```

Using the `subs` argument for `evalf` is the accurate way to evaluate such an expression:

```
>>> (x + y - z).evalf(subs=values)
1.0000000000000000
```

**nseries** (*x*=None, *x0*=0, *n*=6, *dir*='+', *logx*=None)

Wrapper to `_eval_nseries` if assumptions allow, else to `series`.

If *x* is given, *x0* is 0, *dir*='+', and self has *x*, then `_eval_nseries` is called. This calculates “*n*” terms in the innermost expressions and then builds up the final series just by “cross-multiplying” everything out.

The optional `logx` parameter can be used to replace any `log(x)` in the returned series with a symbolic value to avoid evaluating `log(x)` at 0. A symbol to use in place of `log(x)` should be provided.

Advantage – it's fast, because we don't have to determine how many terms we need to calculate in advance.

Disadvantage – you may end up with less terms than you may have expected, but the  $O(x^{**n})$  term appended will always be correct and so the result, though perhaps shorter, will also be correct.

If any of those assumptions is not met, this is treated like a wrapper to `series` which will try harder to return the correct number of terms.

See also `lseries()`.

## Examples

```
>>> from sympy import sin, log, Symbol
>>> from sympy.abc import x, y
>>> sin(x).nseries(x, 0, 6)
x - x**3/6 + x**5/120 + O(x**6)
>>> log(x+1).nseries(x, 0, 5)
x - x**2/2 + x**3/3 - x**4/4 + O(x**5)
```

Handling of the `logx` parameter — in the following example the expansion fails since `sin` does not have an asymptotic expansion at -oo (the limit of `log(x)` as `x` approaches 0):

```
>>> e = sin(log(x))
>>> e.nseries(x, 0, 6)
Traceback (most recent call last):
...
PoleError: ...
...
>>> logx = Symbol('logx')
>>> e.nseries(x, 0, 6, logx=logx)
sin(logx)
```

In the following example, the expansion works but gives only an Order term unless the `logx` parameter is used:

```
>>> e = x**y
>>> e.nseries(x, 0, 2)
O(log(x)**2)
>>> e.nseries(x, 0, 2, logx=logx)
exp(logx*y)
```

**`nsimplify`** (*constants*=[], *tolerance*=None, *full*=False)

See the `nsimplify` function in `sympy.simplify`

**`powsimp`** (*\*args*, *\*\*kwargs*)

See the `powsimp` function in `sympy.simplify`

**`primitive`** ()

Return the positive Rational that can be extracted non-recursively from every term of self (i.e., self is treated like an `Add`). This is like the `as_coeff_Mul()` method but `primitive` always extracts a positive Rational (never a negative or a Float).

## Examples

```
>>> from sympy.abc import x
>>> (3*(x + 1)**2).primitive()
(3, (x + 1)**2)
>>> a = (6*x + 2); a.primitive()
(2, 3*x + 1)
>>> b = (x/2 + 3); b.primitive()
(1/2, x + 6)
>>> (a*b).primitive() == (1, a*b)
True
```

**radsimp** (*\*\*kwargs*)

See the radsimp function in sympy.simplify

**ratsimp** ()

See the ratsimp function in sympy.simplify

**rcall** (*\*args*)

Apply on the argument recursively through the expression tree.

This method is used to simulate a common abuse of notation for operators. For instance in SymPy the following will not work:

```
(x+Lambda(y, 2*y))(z) == x+2*z,
```

however you can use

```
>>> from sympy import Lambda
>>> from sympy.abc import x, y, z
>>> (x + Lambda(y, 2*y)).rcall(z)
x + 2*z
```

**refine** (*assumption=True*)

See the refine function in sympy.assumptions

**removeO** ()

Removes the additive O(..) symbol if there is one

**replace** (*query, value, map=False, simultaneous=True, exact=None*)

Replace matching subexpressions of `self` with `value`.

If `map = True` then also return the mapping `{old: new}` where `old` was a sub-expression found with `query` and `new` is the replacement value for it. If the expression itself doesn't match the `query`, then the returned value will be `self.xreplace(map)` otherwise it should be `self.subs(ordered(map.items()))`.

Traverses an expression tree and performs replacement of matching subexpressions from the bottom to the top of the tree. The default approach is to do the replacement in a simultaneous fashion so changes made are targeted only once. If this is not desired or causes problems, `simultaneous` can be set to `False`.

In addition, if an expression containing more than one Wild symbol is being used to match subexpressions and the `exact` flag is `None` it will be set to `True` so the match will only succeed if all non-zero values are received for each Wild that appears in the match pattern. Setting this to `False` accepts a match of 0; while setting it `True` accepts all matches that have a 0 in them. See example below for cautions.

The list of possible combinations of queries and replacement values is listed below:

## Examples

Initial setup

```
>>> from sympy import log, sin, cos, tan, Wild, Mul, Add
>>> from sympy.abc import x, y
>>> f = log(sin(x)) + tan(sin(x**2))
```

### 1.1. type -> type obj.replace(type, newtype)

When object of type `type` is found, replace it with the result of passing its argument(s) to `newtype`.

```
>>> f.replace(sin, cos)
log(cos(x)) + tan(cos(x**2))
>>> sin(x).replace(sin, cos, map=True)
(cos(x), {sin(x): cos(x)})
>>> (x*y).replace(Mul, Add)
x + y
```

### 1.2. type -> func obj.replace(type, func)

When object of type `type` is found, apply `func` to its argument(s). `func` must be written to handle the number of arguments of `type`.

```
>>> f.replace(sin, lambda arg: sin(2*arg))
log(sin(2*x)) + tan(sin(2*x**2))
>>> (x*y).replace(Mul, lambda *args: sin(2*Mul(*args)))
sin(2*x*y)
```

### 2.1. pattern -> expr obj.replace(pattern(wild), expr(wild))

Replace subexpressions matching `pattern` with the expression written in terms of the Wild symbols in `pattern`.

```
>>> a, b = map(Wild, 'ab')
>>> f.replace(sin(a), tan(a))
log(tan(x)) + tan(tan(x**2))
>>> f.replace(sin(a), tan(a/2))
log(tan(x/2)) + tan(tan(x**2/2))
>>> f.replace(sin(a), a)
log(x) + tan(x**2)
>>> (x*y).replace(a*x, a)
y
```

Matching is exact by default when more than one Wild symbol is used: matching fails unless the match gives non-zero values for all Wild symbols:

```
>>> (2*x + y).replace(a*x + b, b - a)
y - 2
>>> (2*x).replace(a*x + b, b - a)
2*x
```

When set to `False`, the results may be non-intuitive:

```
>>> (2*x).replace(a*x + b, b - a, exact=False)
2/x
```

### 2.2. pattern -> func obj.replace(pattern(wild), lambda wild: expr(wild))



All behavior is the same as in 2.1 but now a function in terms of pattern variables is used rather than an expression:

```
>>> f.replace(sin(a), lambda a: sin(2*a))
log(sin(2*x)) + tan(sin(2*x**2))
```

### 3.1. func -> func obj.replace(filter, func)

Replace subexpression *e* with *func*(*e*) if *filter*(*e*) is True.

```
>>> g = 2*sin(x**3)
>>> g.replace(lambda expr: expr.is_Number, lambda expr: expr**2)
4*sin(x**9)
```

The expression itself is also targeted by the query but is done in such a fashion that changes are not made twice.

```
>>> e = x*(x*y + 1)
>>> e.replace(lambda x: x.is_Mul, lambda x: 2*x)
2*x*(2*x*y + 1)
```

When matching a single symbol, *exact* will default to True, but this may or may not be the behavior that is desired:

Here, we want *exact=False*:

```
>>> from sympy import Function
>>> f = Function('f')
>>> e = f(1) + f(0)
>>> q = f(a), lambda a: f(a + 1)
>>> e.replace(*q, exact=False)
f(1) + f(2)
>>> e.replace(*q, exact=True)
f(0) + f(2)
```

But here, the nature of matching makes selecting the right setting tricky:

```
>>> e = x**(1 + y)
>>> (x**(1 + y)).replace(x**(1 + a), lambda a: x**-a, exact=False)
1
>>> (x**(1 + y)).replace(x**(1 + a), lambda a: x**-a, exact=True)
x**(-x - y + 1)
>>> (x**y).replace(x**(1 + a), lambda a: x**-a, exact=False)
1
>>> (x**y).replace(x**(1 + a), lambda a: x**-a, exact=True)
x**(1 - y)
```

It is probably better to use a different form of the query that describes the target expression more precisely:

```
>>> (1 + x**(1 + y)).replace(
... lambda x: x.is_Pow and x.exp.is_Add and x.exp.args[0] == 1,
... lambda x: x.base**(1 - (x.exp - 1)))
...
x**(1 - y) + 1
```

See also:

**subs** substitution of subexpressions as defined by the objects themselves.

**xreplace** exact node replacement in expr tree; also capable of using matching rules

**rewrite** (\*args, \*\*hints)

Rewrite functions in terms of other functions.

Rewrites expression containing applications of functions of one kind in terms of functions of different kind. For example you can rewrite trigonometric functions as complex exponentials or combinatorial functions as gamma function.

As a pattern this function accepts a list of functions to to rewrite (instances of DefinedFunction class). As rule you can use string or a destination function instance (in this case rewrite() will use the str() function).

There is also the possibility to pass hints on how to rewrite the given expressions. For now there is only one such hint defined called 'deep'. When 'deep' is set to False it will forbid functions to rewrite their contents.

## Examples

```
>>> from sympy import sin, exp
>>> from sympy.abc import x
```

Unspecified pattern:

```
>>> sin(x).rewrite(exp)
-I*(exp(I*x) - exp(-I*x))/2
```

Pattern as a single function:

```
>>> sin(x).rewrite(sin, exp)
-I*(exp(I*x) - exp(-I*x))/2
```

Pattern as a list of functions:

```
>>> sin(x).rewrite([sin, ], exp)
-I*(exp(I*x) - exp(-I*x))/2
```

**round** (n=None)

Return x rounded to the given decimal place.

If a complex number would results, apply round to the real and imaginary components of the number.

## Examples

```
>>> from sympy import pi, E, I, S, Add, Mul, Number
>>> pi.round()
3
>>> pi.round(2)
3.14
>>> (2*pi + E*I).round()
6 + 3*I
```

The round method has a chopping effect:

```
>>> (2*pi + I/10).round()
6
>>> (pi/10 + 2*I).round()
2*I
>>> (pi/10 + E*I).round(2)
0.31 + 2.72*I
```

## Notes

The Python builtin function, `round`, always returns a float in Python 2 while the SymPy `round` method (and `round` with a `Number` argument in Python 3) returns a `Number`.

```
>>> from sympy.core.compatibility import PY3
>>> isinstance(round(S(123), -2), Number if PY3 else float)
True
```

For a consistent behavior, and Python 3 rounding rules, import `round` from `sympy.core.compatibility`.

```
>>> from sympy.core.compatibility import round
>>> isinstance(round(S(123), -2), Number)
True
```

**separate** (*deep=False, force=False*)

See the `separate` function in `sympy.simplify`

**series** (*x=None, x0=0, n=6, dir='+', logx=None*)

Series expansion of “self” around  $x = x_0$  yielding either terms of the series one by one (the lazy series given when  $n=None$ ), else all the terms at once when  $n \neq None$ .

Returns the series expansion of “self” around the point  $x = x_0$  with respect to  $x$  up to  $O((x - x_0)^{n+1})$ ,  $x, x_0$  (default  $n$  is 6).

If  $x=None$  and `self` is univariate, the univariate symbol will be supplied, otherwise an error will be raised.

### Parameters

- **expr** (*Expression*) – The expression whose series is to be expanded.
- **x** (*Symbol*) – It is the variable of the expression to be calculated.
- **x0** (*Value*) – The value around which  $x$  is calculated. Can be any value from  $-\infty$  to  $\infty$ .
- **n** (*Value*) – The number of terms upto which the series is to be expanded.
- **dir** (*String, optional*) – The series-expansion can be bi-directional. If `dir="+"`, then  $(x \rightarrow x_0+)$ . If `dir="-"`, then  $(x \rightarrow x_0-)$ . For infinite `x0` ( $\infty$  or  $-\infty$ ), the `dir` argument is determined from the direction of the infinity (i.e., `dir="-"` for  $\infty$ ).
- **logx** (*optional*) – It is used to replace any  $\log(x)$  in the returned series with a symbolic value rather than evaluating the actual value.

## Examples

```
>>> from sympy import cos, exp, tan, oo, series
>>> from sympy.abc import x, y
>>> cos(x).series()
1 - x**2/2 + x**4/24 + O(x**6)
>>> cos(x).series(n=4)
1 - x**2/2 + O(x**4)
>>> cos(x).series(x, x0=1, n=2)
cos(1) - (x - 1)*sin(1) + O((x - 1)**2, (x, 1))
>>> e = cos(x + exp(y))
>>> e.series(y, n=2)
cos(x + 1) - y*sin(x + 1) + O(y**2)
>>> e.series(x, n=2)
cos(exp(y)) - x*sin(exp(y)) + O(x**2)
```

If `n=None` then a generator of the series terms will be returned.

```
>>> term=cos(x).series(n=None)
>>> [next(term) for i in range(2)]
[1, -x**2/2]
```

For `dir=+` (default) the series is calculated from the right and for `dir=-` the series from the left. For smooth functions this flag will not alter the results.

```
>>> abs(x).series(dir="+")
x
>>> abs(x).series(dir="-")
-x
>>> f = tan(x)
>>> f.series(x, 2, 6, "+")
tan(2) + (1 + tan(2)**2)*(x - 2) + (x - 2)**2*(tan(2)**3 + tan(2)) +
(x - 2)**3*(1/3 + 4*tan(2)**2/3 + tan(2)**4) + (x - 2)**4*(tan(2)**5 +
5*tan(2)**3/3 + 2*tan(2)/3) + (x - 2)**5*(2/15 + 17*tan(2)**2/15 +
2*tan(2)**4 + tan(2)**6) + O((x - 2)**6, (x, 2))
```

```
>>> f.series(x, 2, 3, "-")
tan(2) + (2 - x)*(-tan(2)**2 - 1) + (2 - x)**2*(tan(2)**3 + tan(2))
+ O((x - 2)**3, (x, 2))
```

**Returns** `Expr` – Series expansion of the expression about `x0`

**Return type** Expression

**Raises**

- **TypeError** – If “`n`” and “`x0`” are infinity objects
- **PoleError** – If “`x0`” is an infinity object

**simplify** (*\*\*kwargs*)

See the `simplify` function in `sympy.simplify`

**sort\_key** (*order=None*)

Return a sort key.

## Examples

```
>>> from sympy.core import S, I
```

```
>>> sorted([S(1)/2, I, -I], key=lambda x: x.sort_key())
[1/2, -I, I]
```

```
>>> S("[x, 1/x, 1/x**2, x**2, x**(1/2), x**(1/4), x**(3/2)]")
[x, 1/x, x**(-2), x**2, sqrt(x), x**(1/4), x**(3/2)]
>>> sorted(_, key=lambda x: x.sort_key())
[x**(-2), 1/x, x**(1/4), sqrt(x), x, x**(3/2), x**2]
```

**subs** (\*args, \*\*kwargs)

Substitutes old for new in an expression after sympifying args.

**args** is either:

- two arguments, e.g. `foo.subs(old, new)`
- one iterable argument, e.g. `foo.subs(iterable)`. The iterable may be
  - o an iterable container with (old, new) pairs. In this case the replacements are processed in the order given with successive patterns possibly affecting replacements already made.
  - o a dict or set whose key/value items correspond to old/new pairs. In this case the old/new pairs will be sorted by op count and in case of a tie, by number of args and the default\_sort\_key. The resulting sorted list is then processed as an iterable container (see previous).

If the keyword `simultaneous` is `True`, the subexpressions will not be evaluated until all the substitutions have been made.

## Examples

```
>>> from sympy import pi, exp, limit, oo
>>> from sympy.abc import x, y
>>> (1 + x*y).subs(x, pi)
pi*y + 1
>>> (1 + x*y).subs({x:pi, y:2})
1 + 2*pi
>>> (1 + x*y).subs([(x, pi), (y, 2)])
1 + 2*pi
>>> reps = [(y, x**2), (x, 2)]
>>> (x + y).subs(reps)
6
>>> (x + y).subs(reversed(reps))
x**2 + 2
```

```
>>> (x**2 + x**4).subs(x**2, y)
y**2 + y
```

To replace only the `x**2` but not the `x**4`, use `xreplace`:

```
>>> (x**2 + x**4).xreplace({x**2: y})
x**4 + y
```

To delay evaluation until all substitutions have been made, set the keyword `simultaneous` to `True`:

```
>>> (x/y).subs([(x, 0), (y, 0)])
0
>>> (x/y).subs([(x, 0), (y, 0)], simultaneous=True)
nan
```

This has the added feature of not allowing subsequent substitutions to affect those already made:

```
>>> ((x + y)/y).subs({x + y: y, y: x + y})
1
>>> ((x + y)/y).subs({x + y: y, y: x + y}, simultaneous=True)
y/(x + y)
```

In order to obtain a canonical result, unordered iterables are sorted by `count_op` length, number of arguments and by the `default_sort_key` to break any ties. All other iterables are left unsorted.

```
>>> from sympy import sqrt, sin, cos
>>> from sympy.abc import a, b, c, d, e
```

```
>>> A = (sqrt(sin(2*x)), a)
>>> B = (sin(2*x), b)
>>> C = (cos(2*x), c)
>>> D = (x, d)
>>> E = (exp(x), e)
```

```
>>> expr = sqrt(sin(2*x))*sin(exp(x)*x)*cos(2*x) + sin(2*x)
```

```
>>> expr.subs(dict([A, B, C, D, E]))
a*c*sin(d*e) + b
```

The resulting expression represents a literal replacement of the old arguments with the new arguments. This may not reflect the limiting behavior of the expression:

```
>>> (x**3 - 3*x).subs({x: oo})
nan
```

```
>>> limit(x**3 - 3*x, x, oo)
oo
```

If the substitution will be followed by numerical evaluation, it is better to pass the substitution to evalf as

```
>>> (1/x).evalf(subs={x: 3.0}, n=21)
0.33333333333333333333
```

rather than

```
>>> (1/x).subs({x: 3.0}).evalf(21)
0.333333333333333314830
```

as the former will ensure that the desired level of precision is obtained.

**See also:**

**replace** replacement capable of doing wildcard-like matching, parsing of match, and conditional replacements

**xreplace** exact node replacement in expr tree; also capable of using matching rules

`sympy.core.evalf.EvalfMixin.evalf` calculates the given formula to a desired level of precision

**taylor\_term** (*n*, *x*, *\*previous\_terms*)  
General method for the taylor term.

This method is slow, because it differentiates *n*-times. Subclasses can redefine it to make it faster by using the “previous\_terms”.

**together** (*\*args*, *\*\*kwargs*)  
See the together function in sympy.polys

**trigsimp** (*\*\*args*)  
See the trigsimp function in sympy.simplify

**xreplace** (*rule*, *hack2=False*)  
Replace occurrences of objects within the expression.

**Parameters** *rule* (*dict-like*) – Expresses a replacement rule

**Returns** *xreplace*

**Return type** the result of the replacement

## Examples

```
>>> from sympy import symbols, pi, exp
>>> x, y, z = symbols('x y z')
>>> (1 + x*y).xreplace({x: pi})
pi*y + 1
>>> (1 + x*y).xreplace({x: pi, y: 2})
1 + 2*pi
```

Replacements occur only if an entire node in the expression tree is matched:

```
>>> (x*y + z).xreplace({x*y: pi})
z + pi
>>> (x*y*z).xreplace({x*y: pi})
x*y*z
>>> (2*x).xreplace({2*x: y, x: z})
y
>>> (2*2*x).xreplace({2*x: y, x: z})
4*z
>>> (x + y + 2).xreplace({x + y: 2})
x + y + 2
>>> (x + 2 + exp(x + 2)).xreplace({x + 2: y})
x + exp(y) + 2
```

*xreplace* doesn't differentiate between free and bound symbols. In the following, *subs*(*x*, *y*) would not change *x* since it is a bound symbol, but *xreplace* does:

```
>>> from sympy import Integral
>>> Integral(x, (x, 1, 2*x)).xreplace({x: y})
Integral(y, (y, 1, 2*y))
```

Trying to replace *x* with an expression raises an error:

```
>>> Integral(x, (x, 1, 2*x)).xreplace({x: 2*y})
ValueError: Invalid limits given: ((2*y, 1, 4*y),)
```

**See also:**

*replace* replacement capable of doing wildcard-like matching, parsing of match, and conditional replacements

*subs* substitution of subexpressions as defined by the objects themselves.



## INDICES AND TABLES

- `genindex`
- `search`



## A

affine\_transform() (in module *trnsystor.utils*), 45  
 all() (*trnsystor.controlcards.ControlCards* class method), 6  
 anchor\_ids() (*trnsystor.linkstyle.LinkStyle* property), 30  
 anchor\_points() (*trnsystor.anchorpoint.AnchorPoint* property), 31  
 anchor\_points() (*trnsystor.TrnsysModel* property), 9  
 AnchorPoint (class in *trnsystor.anchorpoint*), 30  
 apart() (*trnsystor.utils.TypeVariableSymbol* method), 47  
 append() (*trnsystor.collections.ComponentCollection* method), 40  
 append() (*trnsystor.collections.CycleCollection* method), 41  
 args() (*trnsystor.utils.TypeVariableSymbol* property), 47  
 args\_cnc() (*trnsystor.utils.TypeVariableSymbol* method), 47  
 as\_coeff\_Add() (*trnsystor.utils.TypeVariableSymbol* method), 48  
 as\_coeff\_add() (*trnsystor.utils.TypeVariableSymbol* method), 48  
 as\_coeff\_exponent() (*trnsystor.utils.TypeVariableSymbol* method), 48  
 as\_coeff\_Mul() (*trnsystor.utils.TypeVariableSymbol* method), 48  
 as\_coeff\_mul() (*trnsystor.utils.TypeVariableSymbol* method), 48  
 as\_coefficient() (*trnsystor.utils.TypeVariableSymbol* method), 49  
 as\_coefficients\_dict() (*trnsystor.utils.TypeVariableSymbol* method), 50  
 as\_content\_primitive() (*trnsystor.utils.TypeVariableSymbol* method), 50  
 as\_dummy() (*trnsystor.utils.TypeVariableSymbol* method), 51  
 as\_expr() (*trnsystor.utils.TypeVariableSymbol* method), 52  
 as\_independent() (*trnsys-*

*tor.utils.TypeVariableSymbol* method), 52  
 as\_leading\_term() (*trnsystor.utils.TypeVariableSymbol* method), 54  
 as\_numer\_denom() (*trnsystor.utils.TypeVariableSymbol* method), 55  
 as\_ordered\_factors() (*trnsystor.utils.TypeVariableSymbol* method), 55  
 as\_ordered\_terms() (*trnsystor.utils.TypeVariableSymbol* method), 55  
 as\_poly() (*trnsystor.utils.TypeVariableSymbol* method), 55  
 as\_powers\_dict() (*trnsystor.utils.TypeVariableSymbol* method), 55  
 as\_real\_imag() (*trnsystor.utils.TypeVariableSymbol* method), 56  
 as\_set() (*trnsystor.utils.TypeVariableSymbol* method), 56  
 as\_terms() (*trnsystor.utils.TypeVariableSymbol* method), 56  
 aseries() (*trnsystor.utils.TypeVariableSymbol* method), 56  
 assumptions0() (*trnsystor.utils.TypeVariableSymbol* property), 57  
 atoms() (*trnsystor.utils.TypeVariableSymbol* method), 58

## B

basic\_template() (*trnsystor.controlcards.ControlCards* class method), 6  
 binary\_symbols() (*trnsystor.utils.TypeVariableSymbol* property), 59

## C

cancel() (*trnsystor.utils.TypeVariableSymbol* method), 59  
 canonical\_variables() (*trnsystor.utils.TypeVariableSymbol* property), 59  
 centroid() (*trnsystor.anchorpoint.AnchorPoint* property), 32  
 centroid() (*trnsystor.collections.ConstantCollection* property), 33

`centroid()` (*trnsystor.collections.EquationCollection* property), 37  
`centroid()` (*trnsystor.component.Component* property), 24  
`centroid()` (*trnsystor.TrnsysModel* property), 9  
`check_deck_integrity()` (*trnsystor.deck.Deck* method), 7  
`check_extra_tags()` (*trnsystor.trnsysmodel.MetaData* method), 23  
`class_key()` (*trnsystor.utils.TypeVariableSymbol* class method), 59  
`clear()` (*trnsystor.collections.ComponentCollection* method), 40  
`clear()` (*trnsystor.collections.ConstantCollection* method), 33  
`clear()` (*trnsystor.collections.CycleCollection* method), 41  
`clear()` (*trnsystor.collections.EquationCollection* method), 37  
`clear()` (*trnsystor.collections.ExternalFileCollection* method), 41  
`clear()` (*trnsystor.collections.InputCollection* method), 42  
`clear()` (*trnsystor.collections.OutputCollection* method), 43  
`clear()` (*trnsystor.collections.ParameterCollection* method), 44  
`clear()` (*trnsystor.collections.VariableCollection* method), 42  
`coeff()` (*trnsystor.utils.TypeVariableSymbol* method), 59  
`collect()` (*trnsystor.utils.TypeVariableSymbol* method), 61  
`combsimp()` (*trnsystor.utils.TypeVariableSymbol* method), 61  
`compare()` (*trnsystor.utils.TypeVariableSymbol* method), 61  
`Component` (class in *trnsystor.component*), 23  
`ComponentCollection` (class in *trnsystor.collections*), 40  
`compute_leading_term()` (*trnsystor.utils.TypeVariableSymbol* method), 62  
`connect_to()` (*trnsystor.collections.ConstantCollection* method), 33  
`connect_to()` (*trnsystor.collections.EquationCollection* method), 37  
`connect_to()` (*trnsystor.component.Component* method), 25  
`connect_to()` (*trnsystor.statement.Equation* method), 4  
`connect_to()` (*trnsystor.TrnsysModel* method), 9  
`connect_to()` (*trnsystor.typevariable.Derivative* method), 15  
`connect_to()` (*trnsystor.typevariable.Input* method), 13  
`connect_to()` (*trnsystor.typevariable.Output* method), 14  
`connect_to()` (*trnsystor.typevariable.Parameter* method), 12  
`connect_to()` (*trnsystor.typevariable.TypeVariable* method), 28  
`Constant` (class in *trnsystor.statement*), 1  
`constant_number()` (*trnsystor.statement.Constant* property), 2  
`ConstantCollection` (class in *trnsystor.collections*), 33  
`ControlCards` (class in *trnsystor.controlcards*), 5  
`copy()` (*trnsystor.collections.ConstantCollection* method), 34  
`copy()` (*trnsystor.collections.EquationCollection* method), 38  
`copy()` (*trnsystor.component.Component* method), 23  
`copy()` (*trnsystor.statement.Equation* method), 4  
`copy()` (*trnsystor.TrnsysModel* method), 8  
`copy()` (*trnsystor.typevariable.Derivative* method), 15  
`copy()` (*trnsystor.typevariable.Input* method), 13  
`copy()` (*trnsystor.typevariable.Output* method), 14  
`copy()` (*trnsystor.typevariable.Parameter* method), 12  
`copy()` (*trnsystor.typevariable.TypeVariable* method), 28  
`could_extract_minus_sign()` (*trnsystor.utils.TypeVariableSymbol* method), 62  
`count()` (*trnsystor.collections.ComponentCollection* method), 40  
`count()` (*trnsystor.collections.CycleCollection* method), 41  
`count()` (*trnsystor.utils.TypeVariableSymbol* method), 62  
`count_ops()` (*trnsystor.utils.TypeVariableSymbol* method), 62  
`CycleCollection` (class in *trnsystor.collections*), 41

## D

`debug_template()` (*trnsystor.controlcards.ControlCards* class method), 6  
`Deck` (class in *trnsystor.deck*), 6  
`DeckFilePrinter` (class in *trnsystor.utils*), 46  
`default()` (*trnsystor.typecycle.TypeCycle* property), 29  
`Derivative` (class in *trnsystor.typevariable*), 15  
`derivatives()` (*trnsystor.TrnsysModel* property), 9  
`DFQ` (class in *trnsystor.statement*), 19  
`doit()` (*trnsystor.utils.TypeVariableSymbol* method), 62  
`doprint()` (*trnsystor.utils.DeckFilePrinter* method), 46

`dummy_eq()` (*trnsystor.utils.TypeVariableSymbol*  
method), 62

## E

`emptyPrinter()` (*trnsystor.utils.DeckFilePrinter*  
method), 46

`End` (class in *trnsystor.statement*), 21

`eq_number()` (*trnsystor.statement.Equation* property),  
4

`EqSolver` (class in *trnsystor.statement*), 20

`equals()` (*trnsystor.utils.TypeVariableSymbol* method),  
63

`Equation` (class in *trnsystor.statement*), 2

`EquationCollection` (class in *trnsys-*  
*tor.collections*), 36

`evalf()` (*trnsystor.utils.TypeVariableSymbol* method),  
63

`expand()` (*trnsystor.utils.TypeVariableSymbol* method),  
64

`expr_free_symbols()` (*trnsys-*  
*tor.utils.TypeVariableSymbol* property), 64

`extend()` (*trnsystor.collections.ComponentCollection*  
method), 40

`extend()` (*trnsystor.collections.CycleCollection*  
method), 41

`external_files()` (*trnsystor.TrnsysModel* prop-  
erty), 9

`ExternalFile` (class in *trnsystor.externalfile*), 26

`ExternalFileCollection` (class in *trnsys-*  
*tor.collections*), 41

`extract_additively()` (*trnsys-*  
*tor.utils.TypeVariableSymbol* method), 64

`extract_branch_factor()` (*trnsys-*  
*tor.utils.TypeVariableSymbol* method), 65

`extract_multiplicatively()` (*trnsys-*  
*tor.utils.TypeVariableSymbol* method), 65

## F

`factor()` (*trnsystor.utils.TypeVariableSymbol* method),  
66

`find()` (*trnsystor.utils.TypeVariableSymbol* method), 66

`find_best_anchors()` (*trnsys-*  
*tor.anchorpoint.AnchorPoint* method), 31

`fourier_series()` (*trnsys-*  
*tor.utils.TypeVariableSymbol* method), 66

`fps()` (*trnsystor.utils.TypeVariableSymbol* method), 66

`free_symbols()` (*trnsystor.utils.TypeVariableSymbol*  
property), 66

`from_component()` (*trnsystor.studio.StudioHeader*  
class method), 29

`from_dict()` (*trnsys-*  
*tor.collections.ExternalFileCollection* class  
method), 41

`from_dict()` (*trnsystor.collections.InputCollection*  
class method), 42

`from_dict()` (*trnsystor.collections.OutputCollection*  
class method), 43

`from_dict()` (*trnsys-*  
*tor.collections.ParameterCollection* class  
method), 44

`from_dict()` (*trnsystor.collections.VariableCollection*  
class method), 42

`from_expression()` (*trnsystor.statement.Constant*  
class method), 1

`from_expression()` (*trnsystor.statement.Equation*  
class method), 2

`from_string()` (*trnsystor.statement.Version* class  
method), 16

`from_symbolic_expression()` (*trnsys-*  
*tor.statement.Equation* class method), 2

`from_tag()` (*trnsystor.externalfile.ExternalFile* class  
method), 26

`from_tag()` (*trnsystor.statement.Equation* class  
method), 4

`from_tag()` (*trnsystor.trnsysmodel.MetaData* class  
method), 23

`from_tag()` (*trnsystor.typecycle.TypeCycle* class  
method), 28

`from_tag()` (*trnsystor.typevariable.Derivative* class  
method), 15

`from_tag()` (*trnsystor.typevariable.Input* class  
method), 13

`from_tag()` (*trnsystor.typevariable.Output* class  
method), 14

`from_tag()` (*trnsystor.typevariable.Parameter* class  
method), 12

`from_tag()` (*trnsystor.typevariable.TypeVariable* class  
method), 28

`from_xml()` (*trnsystor.TrnsysModel* class method), 8

`from_xml()` (*trnsystor.trnsysmodel.MetaData* class  
method), 23

`fromiter()` (*trnsystor.utils.TypeVariableSymbol* class  
method), 66

`func()` (*trnsystor.utils.TypeVariableSymbol* property),  
66

## G

`gammassimp()` (*trnsystor.utils.TypeVariableSymbol*  
method), 67

`get()` (*trnsystor.collections.ConstantCollection*  
method), 34

`get()` (*trnsystor.collections.EquationCollection*  
method), 38

`get()` (*trnsystor.collections.ExternalFileCollection*  
method), 41

`get()` (*trnsystor.collections.InputCollection* method),  
43

`get()` (*trnsystor.collections.OutputCollection* method), 43  
`get()` (*trnsystor.collections.ParameterCollection* method), 44  
`get()` (*trnsystor.collections.VariableCollection* method), 42  
`get_color()` (*trnsystor.linkstyle.LinkStyle* method), 30  
`get_int_from_rgb()` (in module *trnsystor.utils*), 45  
`get_linestyle()` (*trnsystor.linkstyle.LinkStyle* method), 30  
`get_linewidth()` (*trnsystor.linkstyle.LinkStyle* method), 30  
`get_octo_pts_dict()` (*trnsystor.anchorpoint.AnchorPoint* method), 31  
`get_rgb_from_int()` (in module *trnsystor.utils*), 45  
`getn()` (*trnsystor.utils.TypeVariableSymbol* method), 67  
`getO()` (*trnsystor.utils.TypeVariableSymbol* method), 67  
`graph()` (*trnsystor.deck.Deck* property), 7

## H

`has()` (*trnsystor.utils.TypeVariableSymbol* method), 67

## I

`idx()` (*trnsystor.statement.Equation* property), 4  
`idx()` (*trnsystor.typevariable.Derivative* property), 15  
`idx()` (*trnsystor.typevariable.Input* property), 13  
`idx()` (*trnsystor.typevariable.Output* property), 14  
`idx()` (*trnsystor.typevariable.Parameter* property), 12  
`idx()` (*trnsystor.typevariable.TypeVariable* property), 28  
`idxs()` (*trnsystor.typecycle.TypeCycle* property), 29  
`iloc()` (*trnsystor.collections.ComponentCollection* property), 40  
`index()` (*trnsystor.collections.ComponentCollection* method), 40  
`index()` (*trnsystor.collections.CycleCollection* method), 41  
`initial_input_values()` (*trnsystor.TrnsysModel* property), 9  
`Input` (class in *trnsystor.typevariable*), 13  
`InputCollection` (class in *trnsystor.collections*), 42  
`inputs()` (*trnsystor.collections.ConstantCollection* property), 34  
`inputs()` (*trnsystor.collections.EquationCollection* property), 38  
`inputs()` (*trnsystor.component.Component* property), 24  
`inputs()` (*trnsystor.TrnsysModel* property), 10  
`insert()` (*trnsystor.collections.ComponentCollection* method), 40  
`insert()` (*trnsystor.collections.CycleCollection* method), 41  
`integrate()` (*trnsystor.utils.TypeVariableSymbol* method), 68  
`invalidate_connections()` (*trnsystor.collections.ConstantCollection* method), 34  
`invalidate_connections()` (*trnsystor.collections.EquationCollection* method), 38  
`invalidate_connections()` (*trnsystor.component.Component* method), 26  
`invalidate_connections()` (*trnsystor.TrnsysModel* method), 10  
`invert()` (*trnsystor.utils.TypeVariableSymbol* method), 68  
`is_algebraic_expr()` (*trnsystor.utils.TypeVariableSymbol* method), 68  
`is_connected()` (*trnsystor.collections.ConstantCollection* property), 34  
`is_connected()` (*trnsystor.collections.EquationCollection* property), 38  
`is_connected()` (*trnsystor.component.Component* property), 26  
`is_connected()` (*trnsystor.statement.Equation* property), 4  
`is_connected()` (*trnsystor.TrnsysModel* property), 10  
`is_connected()` (*trnsystor.typevariable.Derivative* property), 15  
`is_connected()` (*trnsystor.typevariable.Input* property), 13  
`is_connected()` (*trnsystor.typevariable.Output* property), 14  
`is_connected()` (*trnsystor.typevariable.Parameter* property), 12  
`is_connected()` (*trnsystor.typevariable.TypeVariable* property), 28  
`is_constant()` (*trnsystor.utils.TypeVariableSymbol* method), 69  
`is_polynomial()` (*trnsystor.utils.TypeVariableSymbol* method), 70  
`is_question()` (*trnsystor.typecycle.TypeCycle* property), 29  
`is_rational_function()` (*trnsystor.utils.TypeVariableSymbol* method), 71  
`items()` (*trnsystor.collections.ConstantCollection* method), 34  
`items()` (*trnsystor.collections.EquationCollection* method), 38  
`items()` (*trnsystor.collections.ExternalFileCollection* method), 41  
`items()` (*trnsystor.collections.InputCollection*

method), 43  
 items() (trnsystor.collections.OutputCollection  
 method), 43  
 items() (trnsystor.collections.ParameterCollection  
 method), 44  
 items() (trnsystor.collections.VariableCollection  
 method), 42

## K

keys() (trnsystor.collections.ConstantCollection  
 method), 34  
 keys() (trnsystor.collections.EquationCollection  
 method), 38  
 keys() (trnsystor.collections.ExternalFileCollection  
 method), 41  
 keys() (trnsystor.collections.InputCollection method),  
 43  
 keys() (trnsystor.collections.OutputCollection  
 method), 43  
 keys() (trnsystor.collections.ParameterCollection  
 method), 44  
 keys() (trnsystor.collections.VariableCollection  
 method), 42

## L

leadterm() (trnsystor.utils.TypeVariableSymbol  
 method), 71  
 limit() (trnsystor.utils.TypeVariableSymbol method),  
 72  
 Limits (class in trnsystor.statement), 19  
 link\_styles() (trnsys-  
 tor.collections.ConstantCollection property),  
 34  
 link\_styles() (trnsys-  
 tor.collections.EquationCollection property),  
 38  
 link\_styles() (trnsystor.component.Component  
 property), 24  
 link\_styles() (trnsystor.TrnsysModel property), 10  
 LinkStyle (class in trnsystor.linkstyle), 29  
 List (class in trnsystor.statement), 18  
 loc() (trnsystor.collections.ComponentCollection prop-  
 erty), 40  
 lseries() (trnsystor.utils.TypeVariableSymbol  
 method), 72

## M

Map (class in trnsystor.statement), 20  
 match() (trnsystor.utils.TypeVariableSymbol method),  
 72  
 matches() (trnsystor.utils.TypeVariableSymbol  
 method), 73  
 MetaData (class in trnsystor.trnsysmodel), 22

model() (trnsystor.collections.ConstantCollection  
 property), 34  
 model() (trnsystor.collections.EquationCollection  
 property), 37  
 model() (trnsystor.component.Component property),  
 24  
 model() (trnsystor.TrnsysModel property), 10

## N

n() (trnsystor.utils.TypeVariableSymbol method), 73  
 NaNCheck (class in trnsystor.statement), 17  
 NoCheck (class in trnsystor.statement), 20  
 NoList (class in trnsystor.statement), 20  
 nseries() (trnsystor.utils.TypeVariableSymbol  
 method), 73  
 nsimplify() (trnsystor.utils.TypeVariableSymbol  
 method), 74

## O

one\_based\_idx() (trnsystor.statement.Equation  
 property), 4  
 one\_based\_idx() (trnsystor.typevariable.Derivative  
 property), 15  
 one\_based\_idx() (trnsystor.typevariable.Input prop-  
 erty), 13  
 one\_based\_idx() (trnsystor.typevariable.Output  
 property), 14  
 one\_based\_idx() (trnsystor.typevariable.Parameter  
 property), 12  
 one\_based\_idx() (trnsys-  
 tor.typevariable.TypeVariable property),  
 28  
 Output (class in trnsystor.typevariable), 14  
 OutputCollection (class in trnsystor.collections),  
 43  
 outputs() (trnsystor.collections.ConstantCollection  
 property), 35  
 outputs() (trnsystor.collections.EquationCollection  
 property), 38  
 outputs() (trnsystor.component.Component prop-  
 erty), 24  
 outputs() (trnsystor.TrnsysModel property), 10  
 OverwriteCheck (class in trnsystor.statement), 17

## P

Parameter (class in trnsystor.typevariable), 12  
 ParameterCollection (class in trnsys-  
 tor.collections), 44  
 parameters() (trnsystor.TrnsysModel property), 9  
 path() (trnsystor.linkstyle.LinkStyle property), 30  
 plot() (trnsystor.TrnsysModel method), 9  
 pop() (trnsystor.collections.ComponentCollection  
 method), 40



- pop () (trnsystor.collections.ConstantCollection method), 35
- pop () (trnsystor.collections.CycleCollection method), 41
- pop () (trnsystor.collections.EquationCollection method), 38
- pop () (trnsystor.collections.ExternalFileCollection method), 41
- pop () (trnsystor.collections.InputCollection method), 43
- pop () (trnsystor.collections.OutputCollection method), 43
- pop () (trnsystor.collections.ParameterCollection method), 44
- pop () (trnsystor.collections.VariableCollection method), 42
- popitem () (trnsystor.collections.ConstantCollection method), 35
- popitem () (trnsystor.collections.EquationCollection method), 38
- popitem () (trnsystor.collections.ExternalFileCollection method), 41
- popitem () (trnsystor.collections.InputCollection method), 43
- popitem () (trnsystor.collections.OutputCollection method), 43
- popitem () (trnsystor.collections.ParameterCollection method), 44
- popitem () (trnsystor.collections.VariableCollection method), 42
- powsimp () (trnsystor.utils.TypeVariableSymbol method), 74
- predecessor () (trnsystor.statement.Equation property), 4
- predecessor () (trnsystor.typevariable.Derivative property), 15
- predecessor () (trnsystor.typevariable.Input property), 13
- predecessor () (trnsystor.typevariable.Output property), 14
- predecessor () (trnsystor.typevariable.Parameter property), 12
- predecessor () (trnsystor.typevariable.TypeVariable property), 28
- predecessors () (trnsystor.collections.ConstantCollection property), 35
- predecessors () (trnsystor.collections.EquationCollection property), 38
- predecessors () (trnsystor.component.Component property), 26
- predecessors () (trnsystor.TrnsysModel property), 10
- primitive () (trnsystor.utils.TypeVariableSymbol method), 74
- print\_my\_latex () (in module trnsystor.utils), 46
- ## R
- radsimp () (trnsystor.utils.TypeVariableSymbol method), 75
- ratsimp () (trnsystor.utils.TypeVariableSymbol method), 75
- rcall () (trnsystor.utils.TypeVariableSymbol method), 75
- read\_file () (trnsystor.deck.Deck class method), 7
- refine () (trnsystor.utils.TypeVariableSymbol method), 75
- remove () (trnsystor.collections.ComponentCollection method), 40
- remove () (trnsystor.collections.CycleCollection method), 41
- remove\_models () (trnsystor.deck.Deck method), 7
- removeO () (trnsystor.utils.TypeVariableSymbol method), 75
- replace () (trnsystor.utils.TypeVariableSymbol method), 75
- return\_equation\_or\_constant () (trnsystor.deck.Deck method), 8
- reverse () (trnsystor.collections.ComponentCollection method), 41
- reverse () (trnsystor.collections.CycleCollection method), 42
- reverse\_anchor\_points () (trnsystor.anchorpoint.AnchorPoint property), 31
- reverse\_anchor\_points () (trnsystor.TrnsysModel property), 9
- rewrite () (trnsystor.utils.TypeVariableSymbol method), 78
- round () (trnsystor.utils.TypeVariableSymbol method), 78
- ## S
- save () (trnsystor.deck.Deck method), 7
- separate () (trnsystor.utils.TypeVariableSymbol method), 79
- series () (trnsystor.utils.TypeVariableSymbol method), 79
- set\_canvas\_position () (trnsystor.collections.ConstantCollection method), 35
- set\_canvas\_position () (trnsystor.collections.EquationCollection method), 38
- set\_canvas\_position () (trnsystor.component.Component method), 24
- set\_canvas\_position () (trnsystor.TrnsysModel method), 10



set\_color() (*trnsystor.linkstyle.LinkStyle* method), 30  
 set\_component\_layer() (*trnsystor.collections.ConstantCollection* method), 35  
 set\_component\_layer() (*trnsystor.collections.EquationCollection* method), 39  
 set\_component\_layer() (*trnsystor.component.Component* method), 24  
 set\_component\_layer() (*trnsystor.TrnsysModel* method), 11  
 set\_global\_settings() (*trnsystor.utils.DeckFilePrinter* class method), 46  
 set\_linestyle() (*trnsystor.linkstyle.LinkStyle* method), 30  
 set\_linewidth() (*trnsystor.linkstyle.LinkStyle* method), 30  
 set\_link\_style() (*trnsystor.collections.ConstantCollection* method), 35  
 set\_link\_style() (*trnsystor.collections.EquationCollection* method), 39  
 set\_link\_style() (*trnsystor.component.Component* method), 25  
 set\_link\_style() (*trnsystor.TrnsysModel* method), 11  
 set\_statement() (*trnsystor.controlcards.ControlCards* method), 6  
 set\_typevariable() (*trnsystor.deck.Deck* static method), 8  
 setdefault() (*trnsystor.collections.ConstantCollection* method), 36  
 setdefault() (*trnsystor.collections.EquationCollection* method), 39  
 setdefault() (*trnsystor.collections.ExternalFileCollection* method), 41  
 setdefault() (*trnsystor.collections.InputCollection* method), 43  
 setdefault() (*trnsystor.collections.OutputCollection* method), 43  
 setdefault() (*trnsystor.collections.ParameterCollection* method), 44  
 setdefault() (*trnsystor.collections.VariableCollection* method), 42  
 simplify() (*trnsystor.utils.TypeVariableSymbol* method), 80  
 Simulation (class in *trnsystor.statement*), 18  
 size() (*trnsystor.collections.ConstantCollection* property), 33  
 size() (*trnsystor.collections.EquationCollection* property), 37  
 size() (*trnsystor.collections.InputCollection* property), 42  
 size() (*trnsystor.collections.OutputCollection* property), 43  
 size() (*trnsystor.collections.ParameterCollection* property), 44  
 size() (*trnsystor.collections.VariableCollection* property), 42  
 Solver (class in *trnsystor.statement*), 21  
 sort\_key() (*trnsystor.utils.TypeVariableSymbol* method), 80  
 special\_cards() (*trnsystor.TrnsysModel* property), 9  
 Statement (class in *trnsystor.statement*), 16  
 studio\_anchor() (*trnsystor.anchorpoint.AnchorPoint* method), 30  
 studio\_anchor\_mapping() (*trnsystor.anchorpoint.AnchorPoint* property), 31  
 studio\_anchor\_reverse\_mapping() (*trnsystor.anchorpoint.AnchorPoint* property), 31  
 StudioHeader (class in *trnsystor.studio*), 29  
 subs() (*trnsystor.utils.TypeVariableSymbol* method), 81  
 successors() (*trnsystor.collections.ConstantCollection* property), 36  
 successors() (*trnsystor.collections.EquationCollection* property), 39  
 successors() (*trnsystor.component.Component* property), 26  
 successors() (*trnsystor.TrnsysModel* property), 11  
 successors() (*trnsystor.typevariable.Output* property), 14

## T

taylor\_term() (*trnsystor.utils.TypeVariableSymbol* method), 83  
 TimeReport (class in *trnsystor.statement*), 18  
 to\_file() (*trnsystor.deck.Deck* method), 7  
 together() (*trnsystor.utils.TypeVariableSymbol* method), 83  
 Tolerances (class in *trnsystor.statement*), 19  
 trigsimp() (*trnsystor.utils.TypeVariableSymbol* method), 83  
 TrnsysModel (class in *trnsystor*), 8  
 type\_number() (*trnsystor.collections.ConstantCollection* property),

36  
type\_number() (trnsystor.collections.EquationCollection property), 39  
type\_number() (trnsystor.component.Component property), 24  
type\_number() (trnsystor.TrnsysModel property), 11  
TypeCycle (class in trnsystor.typecycle), 28  
TypeVariable (class in trnsystor.typevariable), 27  
TypeVariableSymbol (class in trnsystor.utils), 46

## U

unit\_name() (trnsystor.collections.ConstantCollection property), 36  
unit\_name() (trnsystor.collections.EquationCollection property), 37  
unit\_name() (trnsystor.component.Component property), 24  
unit\_name() (trnsystor.TrnsysModel property), 11  
unit\_number() (trnsystor.collections.ConstantCollection property), 33  
unit\_number() (trnsystor.collections.EquationCollection property), 39  
unit\_number() (trnsystor.component.Component property), 24  
unit\_number() (trnsystor.statement.Equation property), 4  
unit\_number() (trnsystor.TrnsysModel property), 11  
update() (trnsystor.collections.ConstantCollection method), 33  
update() (trnsystor.collections.EquationCollection method), 37  
update() (trnsystor.collections.ExternalFileCollection method), 41  
update() (trnsystor.collections.InputCollection method), 43  
update() (trnsystor.collections.OutputCollection method), 43  
update() (trnsystor.collections.ParameterCollection method), 44  
update() (trnsystor.collections.VariableCollection method), 42  
update\_meta() (trnsystor.TrnsysModel method), 9  
update\_models() (trnsystor.deck.Deck method), 7

## V

values() (trnsystor.collections.ConstantCollection method), 36  
values() (trnsystor.collections.EquationCollection method), 39

values() (trnsystor.collections.ExternalFileCollection method), 41  
values() (trnsystor.collections.InputCollection method), 43  
values() (trnsystor.collections.OutputCollection method), 43  
values() (trnsystor.collections.ParameterCollection method), 44  
values() (trnsystor.collections.VariableCollection method), 42  
VariableCollection (class in trnsystor.collections), 42  
Version (class in trnsystor.statement), 16

## X

xreplace() (trnsystor.utils.TypeVariableSymbol method), 83