

---

# **modelparameters Documentation**

***Release 1.0***

**Johan Hake, Henrik Finsberg**

**May 01, 2019**



---

## Programmers reference

---

<b>1</b>	<b>Demos</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>5</b>
<b>3</b>	<b>Source code</b>	<b>7</b>
<b>4</b>	<b>Contributors</b>	<b>9</b>
<b>5</b>	<b>License</b>	<b>11</b>
5.1	modelparameters package . . . . .	11
5.1.1	Submodules . . . . .	11
5.1.2	modelparameters.codegeneration module . . . . .	11
5.1.3	modelparameters.commands module . . . . .	13
5.1.4	modelparameters.config module . . . . .	14
5.1.5	modelparameters.logger module . . . . .	14
5.1.6	modelparameters.parameterdict module . . . . .	15
5.1.7	modelparameters.parameters module . . . . .	19
5.1.8	modelparameters.sympytools module . . . . .	25
5.1.9	modelparameters.utils module . . . . .	27
5.1.10	Module contents . . . . .	31
<b>6</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>



Modelparameters is a lightweight library to declare nested parameters in Python, and can be used as a module for providing parameter structure for physical models. It is mainly developed as a tool for [gotran](#), but certainly not limited to that.

```
from modelparameters import ScalarParam
# Define the gravitational acceleration
g = ScalarParam(9.81, name="g", unit="m/s**2",
                description="gravitational acceleration")
# Define the time (in milliseconds)
t = ScalarParam(100.0, unit="ms",
                description="Time spent")
# Compute the distance travelled in free fall
s = 0.5 * g * t**2
print(s)
```

Output:

```
49050.0  [-∞, ∞]
```



# CHAPTER 1

---

Demos

---

Will come later!





## CHAPTER 2

---

### Installation

---

You can install `modelparameters` either with `pip`:

```
pip install modelparameters
```

or if you want the latest features, you can install from `source`:

```
pip install git+https://finsberg@bitbucket.org/finsberg/modelparameters.git
```



## CHAPTER 3

---

### Source code

---

Modelparameters is originally developed by Johan Hake, and the original source code can be found in his [repoistory](#). The current maintained version can be foud [here](#).



## CHAPTER 4

---

### Contributors

---

The main contributors are

- Henrik Finsberg ([henriknf@simula.no](mailto:henriknf@simula.no))
- Johan Hake ([hake.dev@gmail.com](mailto:hake.dev@gmail.com))



GNU GPLv3

## 5.1 modelparameters package

### 5.1.1 Submodules

### 5.1.2 modelparameters.codegeneration module

`modelparameters.codegeneration.latex_unit` (*unit*)  
 Return sympified and LaTeX-formatted string describing given unit. E.g.: `>>> LatexCodeGenerator.format_unit("m/s**2")` `'\mathrm{\frac{m}{s^{2}}}'`

`modelparameters.codegeneration.ccode` (*expr, assign\_to=None, float\_precision='double'*)  
 Return a C-code representation of a sympy expression

`modelparameters.codegeneration.cppcode` (*expr, assign\_to=None, float\_precision='double'*)  
 Return a C++-code representation of a sympy expression

`modelparameters.codegeneration.pythoncode` (*expr, assign\_to=None, namespace='math'*)  
 Return a Python-code representation of a sympy expression

`modelparameters.codegeneration.sympycode` (*expr, assign\_to=None*)

`modelparameters.codegeneration.matlabcode` (*expr, assign\_to=None*)

`modelparameters.codegeneration.juliacode` (*expr, assign\_to=None*)

`modelparameters.codegeneration.latex` (*expr, \*\*settings*)  
 Convert the given expression to LaTeX representation.

```
>>> from sympy import latex, pi, sin, asin, Integral, Matrix, Rational
>>> from sympy.abc import x, y, mu, r, tau
```

```
>>> print(latex((2*tau)**Rational(7,2)))
8 \sqrt{2} \tau^{\frac{7}{2}}
```

Not using a print statement for printing, results in double backslashes for latex commands since that's the way Python escapes backslashes in strings.

```
>>> latex((2*tau)**Rational(7,2))
'8 \sqrt{2} \\\tau^{\frac{7}{2}}'
```

order: Any of the supported monomial orderings (currently “lex”, “grlex”, or “grevlex”), “old”, and “none”. This parameter does nothing for Mul objects. Setting order to “old” uses the compatibility ordering for Add defined in Printer. For very large expressions, set the ‘order’ keyword to ‘none’ if speed is a concern.

mode: Specifies how the generated code will be delimited. ‘mode’ can be one of ‘plain’, ‘inline’, ‘equation’ or ‘equation\*’. If ‘mode’ is set to ‘plain’, then the resulting code will not be delimited at all (this is the default). If ‘mode’ is set to ‘inline’ then inline LaTeX  $\$ \$$  will be used. If ‘mode’ is set to ‘equation’ or ‘equation\*’, the resulting code will be enclosed in the ‘equation’ or ‘equation\*’ environment (remember to import ‘amsmath’ for ‘equation\*’), unless the ‘itex’ option is set. In the latter case, the  $\$ \$$  syntax is used.

```
>>> print(latex((2*mu)**Rational(7,2), mode='plain'))
8 \sqrt{2} \mu^{\frac{7}{2}}
```

```
>>> print(latex((2*tau)**Rational(7,2), mode='inline'))
$8 \sqrt{2} \tau^{7 / 2}$
```

```
>>> print(latex((2*mu)**Rational(7,2), mode='equation*'))
\begin{equation*}8 \sqrt{2} \mu^{\frac{7}{2}}\end{equation*}
```

```
>>> print(latex((2*mu)**Rational(7,2), mode='equation'))
\begin{equation}8 \sqrt{2} \mu^{\frac{7}{2}}\end{equation}
```

itex: Specifies if itex-specific syntax is used, including emitting  $\$ \$$ .

```
>>> print(latex((2*mu)**Rational(7,2), mode='equation', itex=True))
$$8 \sqrt{2} \mu^{\frac{7}{2}}$$
```

fold\_frac\_powers: Emit “ $\{p/q\}$ ” instead of “ $\frac{p}{q}$ ” for fractional powers.

```
>>> print(latex((2*tau)**Rational(7,2), fold_frac_powers=True))
8 \sqrt{2} \tau^{7/2}
```

fold\_func\_brackets: Fold function brackets where applicable.

```
>>> print(latex((2*tau)**sin(Rational(7,2))))
\left(2 \tau\right)^{\sin\left(\frac{7}{2}\right)}
>>> print(latex((2*tau)**sin(Rational(7,2)), fold_func_brackets = True))
\left(2 \tau\right)^{\sin {\frac{7}{2}}}
```

fold\_short\_frac: Emit “ $p / q$ ” instead of “ $\frac{p}{q}$ ” when the denominator is simple enough (at most two terms and no powers). The default value is *True* for inline mode, False otherwise.

```
>>> print(latex(3*x**2/y))
\frac{3 x^{2}}{y}
>>> print(latex(3*x**2/y, fold_short_frac=True))
3 x^{2} / y
```



**long\_frac\_ratio:** The allowed ratio of the width of the numerator to the width of the denominator before we start breaking off long fractions. The default value is 2.

```
>>> print(latex(Integral(r, r)/2/pi, long_frac_ratio=2))
\frac{\int r\, dr}{2 \pi}
>>> print(latex(Integral(r, r)/2/pi, long_frac_ratio=0))
\frac{1}{2 \pi} \int r\, dr
```

**mul\_symbol:** The symbol to use for multiplication. Can be one of None, “ldot”, “dot”, or “times”.

```
>>> print(latex((2*tau)**sin(Rational(7,2)), mul_symbol="times"))
\left(2 \times \tau\right)^{\sin\left(\frac{7}{2} \right)}
```

**inv\_trig\_style:** How inverse trig functions should be displayed. Can be one of “abbreviated”, “full”, or “power”. Defaults to “abbreviated”.

```
>>> print(latex(asin(Rational(7,2))))
\operatorname{asin}\left(\frac{7}{2} \right)
>>> print(latex(asin(Rational(7,2)), inv_trig_style="full"))
\arcsin\left(\frac{7}{2} \right)
>>> print(latex(asin(Rational(7,2)), inv_trig_style="power"))
\sin^{-1}\left(\frac{7}{2} \right)
```

**mat\_str:** Which matrix environment string to emit. “smallmatrix”, “matrix”, “array”, etc. Defaults to “smallmatrix” for inline mode, “matrix” for matrices of no more than 10 columns, and “array” otherwise.

```
>>> print(latex(Matrix(2, 1, [x, y])))
\left[\begin{matrix}x\\y\end{matrix}\right]
```

```
>>> print(latex(Matrix(2, 1, [x, y]), mat_str = "array"))
\left[\begin{array}{c}x\\y\end{array}\right]
```

**mat\_delim:** The delimiter to wrap around matrices. Can be one of “[“, “(“, or the empty string. Defaults to “[“.

```
>>> print(latex(Matrix(2, 1, [x, y]), mat_delim="("))
\left(\begin{matrix}x\\y\end{matrix}\right)
```

**symbol\_names:** Dictionary of symbols and the custom strings they should be emitted as.

```
>>> print(latex(x**2, symbol_names={x: 'x_i'}))
x_i^{\{2\}}
```

latex also supports the builtin container types list, tuple, and dictionary.

```
>>> print(latex([2/x, y], mode='inline'))
$\left[ 2 / x, \quad y\right]$
```

modelparameters.codegeneration.octavecode (expr, assign\_to=None)

### 5.1.3 modelparameters.commands module

modelparameters.commands.get\_output (cmd, inp=None, cwd=None, env=None)

modelparameters.commands.get\_status\_output (cmd, inp=None, cwd=None, env=None)

modelparameters.commands.get\_status\_output\_errors (cmd, inp=None, cwd=None, env=None)

### 5.1.4 modelparameters.config module

`modelparameters.config.float_format()`

### 5.1.5 modelparameters.logger module

```
class modelparameters.logger.Logger (name)
    Bases: object

    add_log_indent (increment=1)
        Add to indentation level.

    add_logfile (filename=None, mode='a')

    begin_log (*message)
        Begin task: write message and increase indentation level.

    debug (*message)
        Write debug message.

    end_log ()
        End task: write a newline and decrease indentation level.

    error (*message, **kwargs)
        Write error message and raise an exception.

    flush_logger ()
        Flush the log handler

    get_log_handler ()
        Get handler for logging.

    get_log_level ()
        Get log level.

    get_logfile_handler (filename)

    get_logger ()
        Return message logger.

    info (*message)
        Write info message.

    info_blue (*message)
        Write info message in blue.

    info_green (*message)
        Write info message in green.

    info_red (*message)
        Write info message in red.

    log (level, *message)
        Write a log message on given log level

    pop_log_level ()
        Pop log level from the level stack, reverting to before the last push_level.

    push_log_level (level)
        Push a log level on the level stack.

    remove_logfile (filename)
```

**set\_default\_exception** (*exception*)

**set\_log\_handler** (*handler*)

Replace handler for logging.

To add additional handlers instead of replacing the existing, use `log.getLogger().addHandler(myhandler)`.

See the logging module for more details.

**set\_log\_indent** (*level*)

Set indentation level.

**set\_log\_level** (*level*)

Set log level.

**set\_log\_prefix** (*prefix*)

Set prefix for log messages.

**set\_raise\_error** (*value*)

**suppress\_logging** ()

Suppress all logging

**type\_error** (*\*message, \*\*kwargs*)

Write error message and raise a type error exception.

**value\_error** (*\*message, \*\*kwargs*)

Write error message and raise a value error exception.

**warning** (*\*message*)

Write warning message.

**wrap\_log\_message** (*message, symbol='\**)

## 5.1.6 modelparameters.parameterdict module

Contains the ParameterDict class, useful for defining recursive dictionaries of parameters and using attribute syntax for later access.

**class** `modelparameters.parameterdict.Param` (*value, name="", description="", \*\*kwargs*)

Bases: `object`

A simple type checking class for a single value

**check** (*value*)

Check the value using the type and any range check

**convert\_to** (*unit*)

Convert parameter to a different unit than the current one.

**Parameters** **unit** (*str*) – The new unit

**Returns** Return the same parameter with the new unit

**Return type** *Param*

### Example

```
>>> p_s = ScalarParam(1.0, unit="s")
>>> p_ms = p_s.convert_to('ms')
>>> print('value = {}, unit = {}'.format(p_ms.value), p_ms.unit))
value = 1000.0, unit = 'milliseconds'
```

**copy** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Return a copy of the parameter

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

#### description

**format\_data** (*value=None, not\_in=False, str\_length=0*)

Print a nice formatted version of the value and its range

#### Parameters

- **value** (*same as Param.value\_type (optional)*) – A value to be used in the forming. If not passed stored value is used.
- **not\_in** (*bool (optional)*) – If True return a not in version of the value
- **str\_length** (*int (optional)*) – Used to pad the str with blanks

**format\_width** ()

Return the width of the formatted str of value

**getvalue** ()

Return the value

**name**

**repr** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Returns an executable version of the Param including optional arguments

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

**setvalue** (*value, check=True*)

Try to set the value using the check

**update** (*value*)

**value**

Return the value

**class** modelparameters.parameterdict.**ScalarParam** (*value, ge=None, le=None, gt=None, lt=None, unit='1', name="", description=""*)

Bases: *modelparameters.parameters.Param*

A simple type and range checking class for a scalar value

**copy** (*include\_checkarg=True, include\_name=True, include\_description=True, include\_unit=True*)

Return a copy of the parameter

**Parameters**

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param
- **include\_unit** (*bool*) – If include unit in new Param

**get\_sym**()**name****repr** (*include\_checkarg=True, include\_name=True, include\_description=True, include\_unit=True*)

Returns an executable version of the Param including optional arguments

**Parameters**

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param
- **include\_unit** (*bool*) – If include unit in new Param

**sym****unit**

Return the unit

**update** (*param*)

Update parameter with value of new parameter. Take into account unit conversion if applicable.

**Parameters param** (*ScalarParameter or scalar*) – The parameter with the new value**class** modelparameters.parameterdict.**OptionParam** (*value, options, name="", description=""*)Bases: *modelparameters.parameters.Param*

A simple type and options checking class for a single value

**repr** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Returns an executable version of the Param including optional arguments

**Parameters**

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

**class** modelparameters.parameterdict.**ConstParam** (*value, name="", description=""*)Bases: *modelparameters.parameters.Param*

A Constant parameter which prevent any change of values

**class** modelparameters.parameterdict.**ArrayParam** (*value, size=None, ge=None, le=None, gt=None, lt=None, unit='1', name="", description=""*)Bases: *modelparameters.parameters.ScalarParam*

A numpy Array based parameter

**resize** (*newsize*)  
Change the size of the Array

**setvalue** (*value*)  
Set value of ArrayParameter

**value**  
Return the value

**class** modelparameters.parameterdict.**SlaveParam** (*expr*, *unit='1'*, *name=""*, *description=""*)  
Bases: `modelparameters.parameters.ScalarParam`

A slave parameter defined by other parameters

**expr**  
Return the stored expression

**format\_data** (*value=None*, *not\_in=False*, *str\_length=0*)  
Print a nice formatted version of the value and its range

**getvalue** ()  
Return a computed value of the Parameters

**setvalue** (*value*)  
A setvalue method which always fails

**value**  
Return a computed value of the Parameters

**class** modelparameters.parameterdict.**ParameterDict** (*\*\*params*)  
Bases: `dict`

A dictionary with attribute-style access, that maps attribute access to the real dictionary.

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

**copy** (*to\_dict=False*)  
Make a deep copy of self, including recursive copying of parameter subsets.

**Parameters to\_dict** (*bool* (*optional*)) – Return a dict with items representing the values of the Parameters

**format\_data** (*indent=None*)  
Make a recursive indented pretty-print string of self and parameter subsets.

**fromkeys** (*\*args*)  
Create a new dictionary with keys from iterable and values set to value.

**iterparameterdicts** ()  
Iterate over all ParameterDicts

**Parameters recurse** (*bool* (*optional*)) – If True each encountered ParameterDict will also be entered

**iterparams** (*recurse=False*)  
Iterate over all Param

**Parameters recurse** (*bool* (*optional*)) – If True each encountered ParameterDict will be entered

**optstr** ()  
Return a string with option set

An option string can be sent to a script using a parameter dict to set its parameters from command line options

**parse\_args** (*options=None, usage=""*)

Parse a list of options. use sys.argv as default

**Parameters** *options* (*list of str (optional)*) – List of options. By default sys.argv[1:] is used. This argument is mostly for debugging.

**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

**update** (*other*)

A recursive update that handles parameter subsets correctly unlike dict.update.

## 5.1.7 modelparameters.parameters module

modelparameters.parameters.**store\_symbol\_parameter** (*param*)

Store a symbol parameter

modelparameters.parameters.**value\_namespace** (*expr, include\_derivatives=False*)

Create a value name space for the included symbols in the expression

modelparameters.parameters.**symbols\_from\_expr** (*expr, include\_numbers=False, include\_derivatives=False*)

Returns a set of all symbols of an expression

### Parameters

- **expr** (*sympy expression*) – A sympy expression containing sympy.Symbols or sympy.AppliedUndef functions.
- **include\_numbers** (*bool*) – If True numbers will also be returned
- **include\_derivatives** (*bool*) – If True derivatives will be returned instead of its variables

modelparameters.parameters.**symbol\_to\_param** (*sym*)

Take a symbol or expression of symbols and returns the corresponding Parameters

modelparameters.parameters.**pythoncode** (*expr, assign\_to=None, namespace='math'*)

Return a Python-code representation of a sympy expression

modelparameters.parameters.**sympycode** (*expr, assign\_to=None*)

modelparameters.parameters.**float\_format** ()

**class** modelparameters.parameters.**Logger** (*name*)

Bases: `object`

**add\_log\_indent** (*increment=1*)

Add to indentation level.

**add\_logfile** (*filename=None, mode='a'*)

**begin\_log** (*\*message*)

Begin task: write message and increase indentation level.

**debug** (*\*message*)

Write debug message.

**end\_log** ()

End task: write a newline and decrease indentation level.

**error** (*\*message, \*\*kwargs*)

Write error message and raise an exception.

**flush\_logger** ()  
Flush the log handler

**get\_log\_handler** ()  
Get handler for logging.

**get\_log\_level** ()  
Get log level.

**get\_logfile\_handler** (*filename*)

**get\_logger** ()  
Return message logger.

**info** (*\*message*)  
Write info message.

**info\_blue** (*\*message*)  
Write info message in blue.

**info\_green** (*\*message*)  
Write info message in green.

**info\_red** (*\*message*)  
Write info message in red.

**log** (*level*, *\*message*)  
Write a log message on given log level

**pop\_log\_level** ()  
Pop log level from the level stack, reverting to before the last push\_level.

**push\_log\_level** (*level*)  
Push a log level on the level stack.

**remove\_logfile** (*filename*)

**set\_default\_exception** (*exception*)

**set\_log\_handler** (*handler*)  
Replace handler for logging.  
  
To add additional handlers instead of replacing the existing, use `log.get_logger().addHandler(myhandler)`.  
  
See the logging module for more details.

**set\_log\_indent** (*level*)  
Set indentation level.

**set\_log\_level** (*level*)  
Set log level.

**set\_log\_prefix** (*prefix*)  
Set prefix for log messages.

**set\_raise\_error** (*value*)

**suppress\_logging** ()  
Suppress all logging

**type\_error** (*\*message*, *\*\*kwargs*)  
Write error message and raise a type error exception.

**value\_error** (*\*message*, *\*\*kwargs*)  
Write error message and raise a value error exception.



**warning** (\*message)  
Write warning message.

**wrap\_log\_message** (message, symbol='\*')

modelparameters.parameters.**check\_arg** (arg, argtypes, num=-1, context=None, itemtypes=None, ge=None, le=None, gt=None, lt=None)

Type check for positional arguments

#### Parameters

- **arg** (any) – The argument to be checked
- **argtypes** (type, tuple) – The type of which arg should be
- **num** (int (optional)) – The argument positional number
- **context** (type, function/method (optional)) – The context of the check. If context is a class the check is assumed to be during creation. If a function/method the context is assumed to be a call to that function/method
- **itemtypes** (type (optional)) – If given argtypes must be a tuple or list and itemtypes forces each item to be a certain type
- **ge** (scalar (optional)) – Greater than or equal, range control of argument
- **le** (scalar (optional)) – Lesser than or equal, range control of argument
- **gt** (scalar (optional)) – Greater than, range control of argument
- **lt** (scalar (optional)) – Lesser than, range control of argument

modelparameters.parameters.**check\_kwarg** (kwarg, name, argtypes, context=None, itemtypes=None, ge=None, le=None, gt=None, lt=None)

Type check for keyword arguments

#### Parameters

- **kwarg** (any) – The keyword argument to be checked
- **name** (str) – The name of the keyword argument
- **argtypes** (type, tuple) – The type of which arg should be
- **context** (type, function/method (optional)) – The context of the check. If context is a class the check is assumed to be during creation. If a function/method the context is assumed to be a call to that function/method
- **itemtypes** (type (optional)) – If given argtypes must be a tuple or list and itemtypes forces each item to be a certain type
- **ge** (scalar (optional)) – Greater than or equal, range control of argument
- **le** (scalar (optional)) – Lesser than or equal, range control of argument
- **gt** (scalar (optional)) – Greater than, range control of argument
- **lt** (scalar (optional)) – Lesser than, range control of argument

modelparameters.parameters.**value\_formatter** (value, width=0)

Return a formatted string of a value

#### Parameters

- **value** (any) – The value which is formatted
- **width** (int) – A min str length value

**class** modelparameters.parameters.**Range** (*ge=None, le=None, gt=None, lt=None*)

Bases: `object`

A simple class for helping checking a given value is within a certain range

**format** (*value, width=0*)

Return a formatted range check of the value

**Parameters**

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

**format\_in** (*value, width=0*)

Return a formatted range check

**Parameters**

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

**format\_not\_in** (*value, width=0*)

Return a formatted range check

**Parameters**

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

modelparameters.parameters.**tuplewrap** (*arg*)

Wrap the argument to a tuple if it is not a tuple

**class** modelparameters.parameters.**Timer** (*task*)

Bases: `object`

Timer class

**classmethod** **timings** ()

Return all registered timings

**class** modelparameters.parameters.**Param** (*value, name="", description="", \*\*kwargs*)

Bases: `object`

A simple type checking class for a single value

**check** (*value*)

Check the value using the type and any range check

**convert\_to** (*unit*)

Convert parameter to a different unit than the current one.

**Parameters** **unit** (*str*) – The new unit

**Returns** Return the same parameter with the new unit

**Return type** *Param*

## Example

```
>>> p_s = ScalarParam(1.0, unit="s")
>>> p_ms = p_s.convert_to('ms')
>>> print('value = {}, unit = {}'.format(p_ms.value), p_ms.unit))
value = 1000.0, unit = 'milliseconds'
```

**copy** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Return a copy of the parameter

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

#### description

**format\_data** (*value=None, not\_in=False, str\_length=0*)

Print a nice formatted version of the value and its range

#### Parameters

- **value** (*same as Param.value\_type (optional)*) – A value to be used in the forming. If not passed stored value is used.
- **not\_in** (*bool (optional)*) – If True return a not in version of the value
- **str\_length** (*int (optional)*) – Used to pad the str with blanks

**format\_width** ()

Return the width of the formatted str of value

**getvalue** ()

Return the value

**name**

**repr** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Returns an executable version of the Param including optional arguments

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

**setvalue** (*value, check=True*)

Try to set the value using the check

**update** (*value*)

**value**

Return the value

**class** modelparameters.parameters.**OptionParam** (*value, options, name="", description=""*)

Bases: *modelparameters.parameters.Param*

A simple type and options checking class for a single value

**repr** (*include\_checkarg=True, include\_name=True, include\_description=True*)

Returns an executable version of the Param including optional arguments

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param

**class** modelparameters.parameters.**ConstParam** (*value*, *name*="", *description*="")

Bases: *modelparameters.parameters.Param*

A Constant parameter which prevent any change of values

**class** modelparameters.parameters.**TypelessParam** (*value*, *name*="", *description*="")

Bases: *modelparameters.parameters.Param*

A Typeless parameter allowing any change of value, including type changes

**class** modelparameters.parameters.**ScalarParam** (*value*, *ge*=None, *le*=None, *gt*=None, *lt*=None, *unit*='l', *name*="", *description*="")

Bases: *modelparameters.parameters.Param*

A simple type and range checking class for a scalar value

**copy** (*include\_checkarg*=True, *include\_name*=True, *include\_description*=True, *include\_unit*=True)

Return a copy of the parameter

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param
- **include\_unit** (*bool*) – If include unit in new Param

**get\_sym** ()

**name**

**repr** (*include\_checkarg*=True, *include\_name*=True, *include\_description*=True, *include\_unit*=True)

Returns an executable version of the Param including optional arguments

#### Parameters

- **include\_checkarg** (*bool*) – If include checkargs in new Param
- **include\_name** (*bool*) – If include name in new Param
- **include\_description** (*bool*) – If include description in new Param
- **include\_unit** (*bool*) – If include unit in new Param

**sym**

**unit**

Return the unit

**update** (*param*)

Update parameter with value of new parameter. Take into account unit conversion if applicable.

**Parameters param** (*ScalarParameter* or *scalar*) – The parameter with the new value

**class** modelparameters.parameters.**ArrayParam** (*value*, *size*=None, *ge*=None, *le*=None, *gt*=None, *lt*=None, *unit*='l', *name*="", *description*="")

Bases: *modelparameters.parameters.ScalarParam*

A numpy Array based parameter

**resize** (*newsize*)

Change the size of the Array

**setvalue** (*value*)

Set value of ArrayParameter

**value**

Return the value

**class** modelparameters.parameters.**SlaveParam** (*expr*, *unit='1'*, *name=""*, *description=""*)

Bases: *modelparameters.parameters.ScalarParam*

A slave parameter defined by other parameters

**expr**

Return the stored expression

**format\_data** (*value=None*, *not\_in=False*, *str\_length=0*)

Print a nice formatted version of the value and its range

**getvalue** ()

Return a computed value of the Parameters

**setvalue** (*value*)

A setvalue method which always fails

**value**

Return a computed value of the Parameters

modelparameters.parameters.**eval\_param\_expr** (*expr*, *param\_ns=None*, *include\_derivatives=False*, *ns=None*) *in-*

Eval an expression of symbols of ScalarParam

#### Parameters

- **expr** (*expression of ParamSymbols*) – The expression to be evaluated
- **param\_ns** (*dict (optional)*) – A namespace containing the parameters for which the expr should be evaluated with.
- **include\_derivatives** (*bool (optional)*) – If True not only symbols are evaluated but also derivatives
- **ns** (*dict (optional)*) – A namespace in which the expression will be evaluated in

### 5.1.8 modelparameters.sympytools module

modelparameters.sympytools.**check\_arg** (*arg*, *argtypes*, *num=-1*, *context=None*, *itemtypes=None*, *ge=None*, *le=None*, *gt=None*, *lt=None*)

Type check for positional arguments

#### Parameters

- **arg** (*any*) – The argument to be checked
- **argtypes** (*type, tuple*) – The type of which arg should be
- **num** (*int (optional)*) – The argument positional number
- **context** (*type, function/method (optional)*) – The context of the check. If context is a class the check is assumed to be during creation. If a function/method the context is assumed to be a call to that function/method

- **itemtypes** (*type optional*) – If given argtypes must be a tuple or list and itemtypes forces each item to be a certain type
- **ge** (*scalar optional*) – Greater than or equal, range control of argument
- **le** (*scalar optional*) – Lesser than or equal, range control of argument
- **gt** (*scalar optional*) – Greater than, range control of argument
- **lt** (*scalar optional*) – Lesser than, range control of argument

`modelparameters.sympytools.deprecated(func)`

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

`modelparameters.sympytools.Conditional(cond, true_value, false_value)`

Declares a conditional

#### Parameters

- **cond** (*A conditional*) – The conditional which should be evaluated
- **true\_value** (*Any model expression*) – Model expression for a true evaluation of the conditional
- **false\_value** (*Any model expression*) – Model expression for a false evaluation of the conditional

`modelparameters.sympytools.ContinuousConditional(cond, true_value, false_value, sigma=1.0)`

Declares a continuous conditional. Instead of a either or result the true and false values are weighted with a sigmoidal function which either evaluates to 0 or 1 instead of the true or false.

#### Parameters

- **cond** (*An InEquality conditional*) – An InEquality conditional which should be evaluated
- **true\_value** (*Any model expression*) – Model expression for a true evaluation of the conditional
- **false\_value** (*Any model expression*) – Model expression for a false evaluation of the conditional
- **sigma** (*float optional*) – Determines the sharpness of the sigmoidal function

`modelparameters.sympytools.store_symbol_parameter(param)`

Store a symbol parameter

`modelparameters.sympytools.symbol_to_params(sym)`

`modelparameters.sympytools.symbol_to_param(sym)`

Take a symbol or expression of symbols and returns the corresponding Parameters

`modelparameters.sympytools.symbols_from_expr(expr, include_numbers=False, include_derivatives=False)`

Returns a set of all symbols of an expression

#### Parameters

- **expr** (*sympy expression*) – A sympy expression containing sympy.Symbols or sympy.AppliedUndef functions.
- **include\_numbers** (*bool*) – If True numbers will also be returned

- **include\_derivatives** (*bool*) – If True derivatives will be returned instead of its variables

`modelparameters.sympytools.iter_symbol_params_from_expr(expr)`

Return an iterator over `sp.Symbols` from `expr`

`modelparameters.sympytools.symbol_params_from_expr(expr)`

Return a list of `Symbols` from `expr`

`modelparameters.sympytools.symbol_param_value_namespace(expr)`

Create a value name space for the included symbols in the expression

`modelparameters.sympytools.value_namespace(expr, include_derivatives=False)`

Create a value name space for the included symbols in the expression

`modelparameters.sympytools.add_pair_to_subs(subs, old, new)`

Add a pair of old and new symbols to `subs`. If a `subs` with `old` as a key already exist it will be removed before insertion.

### 5.1.9 modelparameters.utils module

`modelparameters.utils.ClassType`

alias of `builtins.type`

**class** `modelparameters.utils.Logger(name)`

Bases: `object`

**add\_log\_indent** (*increment=1*)

Add to indentation level.

**add\_logfile** (*filename=None, mode='a'*)

**begin\_log** (*\*message*)

Begin task: write message and increase indentation level.

**debug** (*\*message*)

Write debug message.

**end\_log** ()

End task: write a newline and decrease indentation level.

**error** (*\*message, \*\*kwargs*)

Write error message and raise an exception.

**flush\_logger** ()

Flush the log handler

**get\_log\_handler** ()

Get handler for logging.

**get\_log\_level** ()

Get log level.

**get\_logfile\_handler** (*filename*)

**get\_logger** ()

Return message logger.

**info** (*\*message*)

Write info message.

**info\_blue** (*\*message*)

Write info message in blue.

**info\_green** (*\*message*)

Write info message in green.

**info\_red** (*\*message*)

Write info message in red.

**log** (*level*, *\*message*)

Write a log message on given log level

**pop\_log\_level** ()

Pop log level from the level stack, reverting to before the last push\_level.

**push\_log\_level** (*level*)

Push a log level on the level stack.

**remove\_logfile** (*filename*)

**set\_default\_exception** (*exception*)

**set\_log\_handler** (*handler*)

Replace handler for logging.

To add additional handlers instead of replacing the existing, use `log.get_logger().addHandler(myhandler)`.

See the logging module for more details.

**set\_log\_indent** (*level*)

Set indentation level.

**set\_log\_level** (*level*)

Set log level.

**set\_log\_prefix** (*prefix*)

Set prefix for log messages.

**set\_raise\_error** (*value*)

**suppress\_logging** ()

Suppress all logging

**type\_error** (*\*message*, *\*\*kwargs*)

Write error message and raise a type error exception.

**value\_error** (*\*message*, *\*\*kwargs*)

Write error message and raise a value error exception.

**warning** (*\*message*)

Write warning message.

**wrap\_log\_message** (*message*, *symbol='\**)

`modelparameters.utils.float_format` ()

`modelparameters.utils.reduce` (*function*, *sequence*[, *initial*]) → value

Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value. For example, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates `((((1+2)+3)+4)+5)`. If *initial* is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty.

`modelparameters.utils.rjust` (*s*, *\*args*, *\*\*kwargs*)

`modelparameters.utils.VALUE_JUST` (*s*, *\*args*, *\*\*kwargs*)

`modelparameters.utils.param2value` (*param*)



`modelparameters.utils.value_formatter(value, width=0)`

Return a formatted string of a value

#### Parameters

- **value** (*any*) – The value which is formatted
- **width** (*int*) – A min str length value

**class** `modelparameters.utils.Range` (*ge=None, le=None, gt=None, lt=None*)

Bases: `object`

A simple class for helping checking a given value is within a certain range

**format** (*value, width=0*)

Return a formatted range check of the value

#### Parameters

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

**format\_in** (*value, width=0*)

Return a formatted range check

#### Parameters

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

**format\_not\_in** (*value, width=0*)

Return a formatted range check

#### Parameters

- **value** (*scalar*) – A value to be used in checking range
- **width** (*int*) – A min str length value

`modelparameters.utils.format_time(time)`

Return a formatted version of the time argument

**Parameters** **time** (*float*) – Time given in sections

**class** `modelparameters.utils.Timer` (*task*)

Bases: `object`

Timer class

**classmethod** `timings()`

Return all registered timings

`modelparameters.utils.list_timings()`

List all registered timings

`modelparameters.utils.clear_timings()`

Clear all registered timings

`modelparameters.utils.tic()`

Start timing

`modelparameters.utils.toc()`

Return timing since last toc/tic

`modelparameters.utils.is_iterable(obj)`

Test for iterable

**Parameters** `obj` (*any*) – Object which is beeing tested

`modelparameters.utils.add_iterable(iterable, initial=None)`

Sum the content of an iterable

`modelparameters.utils.camel_capitalize(name)`

Camel capitalize a str

`modelparameters.utils.tuplewrap(arg)`

Wrap the argument to a tuple if it is not a tuple

`modelparameters.utils.listwrap(arg)`

Wrap the argument to a list if it is not a list

`modelparameters.utils.check_arginlist(arg, lst, name='arg')`

Check that arg is in lst

`modelparameters.utils.check_arg(arg, argtypes, num=-1, context=None, itemtypes=None, ge=None, le=None, gt=None, lt=None)`

Type check for positional arguments

#### Parameters

- **arg** (*any*) – The argument to be checked
- **argtypes** (*type*, *tuple*) – The type of which arg should be
- **num** (*int* (*optional*)) – The argument positional number
- **context** (*type*, *function/method* (*optional*)) – The context of the check. If context is a class the check is assumed to be during creation. If a function/method the context is assumed to be a call to that function/method
- **itemtypes** (*type* (*optional*)) – If given argtypes must be a tuple or list and itemtypes forces each item to be a certain type
- **ge** (*scalar* (*optional*)) – Greater than or equal, range control of argument
- **le** (*scalar* (*optional*)) – Lesser than or equal, range control of argument
- **gt** (*scalar* (*optional*)) – Greater than, range control of argument
- **lt** (*scalar* (*optional*)) – Lesser than, range control of argument

`modelparameters.utils.check_kwarg(kwarg, name, argtypes, context=None, itemtypes=None, ge=None, le=None, gt=None, lt=None)`

Type check for keyword arguments

#### Parameters

- **kwarg** (*any*) – The keyword argument to be checked
- **name** (*str*) – The name of the keyword argument
- **argtypes** (*type*, *tuple*) – The type of which arg should be
- **context** (*type*, *function/method* (*optional*)) – The context of the check. If context is a class the check is assumed to be during creation. If a function/method the context is assumed to be a call to that function/method
- **itemtypes** (*type* (*optional*)) – If given argtypes must be a tuple or list and itemtypes forces each item to be a certain type
- **ge** (*scalar* (*optional*)) – Greater than or equal, range control of argument

- `le (scalar (optional))` – Lesser than or equal, range control of argument
- `gt (scalar (optional))` – Greater than, range control of argument
- `lt (scalar (optional))` – Lesser than, range control of argument

`modelparameters.utils.quote_join (list_of_str)`

Join a list of strings with quotes and commans

`modelparameters.utils.deprecated (func)`

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

### 5.1.10 Module contents



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### m

- `modelparameters`, [31](#)
- `modelparameters.codegeneration`, [11](#)
- `modelparameters.commands`, [13](#)
- `modelparameters.config`, [14](#)
- `modelparameters.logger`, [14](#)
- `modelparameters.parameterdict`, [15](#)
- `modelparameters.parameters`, [19](#)
- `modelparameters.sympytools`, [25](#)
- `modelparameters.utils`, [27](#)





## A

`add_iterable()` (in module `modelparameters.utils`), 30  
`add_log_indent()` (`modelparameters.logger.Logger` method), 14  
`add_log_indent()` (`modelparameters.parameters.Logger` method), 19  
`add_log_indent()` (`modelparameters.utils.Logger` method), 27  
`add_logfile()` (`modelparameters.logger.Logger` method), 14  
`add_logfile()` (`modelparameters.parameters.Logger` method), 19  
`add_logfile()` (`modelparameters.utils.Logger` method), 27  
`add_pair_to_subs()` (in module `modelparameters.sympytools`), 27  
`ArrayParam` (class in `modelparameters.parameters.parameterdict`), 17  
`ArrayParam` (class in `modelparameters.parameters`), 24

## B

`begin_log()` (`modelparameters.logger.Logger` method), 14  
`begin_log()` (`modelparameters.parameters.Logger` method), 19  
`begin_log()` (`modelparameters.utils.Logger` method), 27

## C

`camel_capitalize()` (in module `modelparameters.utils`), 30  
`ccode()` (in module `modelparameters.codegeneration`), 11  
`check()` (`modelparameters.parameterdict.Param` method), 15  
`check()` (`modelparameters.parameters.Param` method), 22

`check_arg()` (in module `modelparameters.parameters`), 21  
`check_arg()` (in module `modelparameters.sympytools`), 25  
`check_arg()` (in module `modelparameters.utils`), 30  
`check_arginlist()` (in module `modelparameters.utils`), 30  
`check_kwarg()` (in module `modelparameters.parameters`), 21  
`check_kwarg()` (in module `modelparameters.utils`), 30  
`ClassType` (in module `modelparameters.utils`), 27  
`clear()` (`modelparameters.parameterdict.ParameterDict` method), 18  
`clear_timings()` (in module `modelparameters.utils`), 29  
`Conditional()` (in module `modelparameters.sympytools`), 26  
`ConstParam` (class in `modelparameters.parameterdict`), 17  
`ConstParam` (class in `modelparameters.parameters`), 24  
`ContinuousConditional()` (in module `modelparameters.sympytools`), 26  
`convert_to()` (`modelparameters.parameterdict.Param` method), 15  
`convert_to()` (`modelparameters.parameters.Param` method), 22  
`copy()` (`modelparameters.parameterdict.Param` method), 16  
`copy()` (`modelparameters.parameterdict.ParameterDict` method), 18  
`copy()` (`modelparameters.parameterdict.ScalarParam` method), 16  
`copy()` (`modelparameters.parameters.Param` method), 23  
`copy()` (`modelparameters.parameters.ScalarParam` method), 24

`cppcode()` (in module `modelparameters.codegeneration`), 11

## D

`debug()` (`modelparameters.logger.Logger` method), 14

`debug()` (`modelparameters.parameters.Logger` method), 19

`debug()` (`modelparameters.utils.Logger` method), 27

`deprecated()` (in module `modelparameters.sympytools`), 26

`deprecated()` (in module `modelparameters.utils`), 31

`description` (`modelparameters.parameterdict.Param` attribute), 16

`description` (`modelparameters.parameters.Param` attribute), 23

## E

`end_log()` (`modelparameters.logger.Logger` method), 14

`end_log()` (`modelparameters.parameters.Logger` method), 19

`end_log()` (`modelparameters.utils.Logger` method), 27

`error()` (`modelparameters.logger.Logger` method), 14

`error()` (`modelparameters.parameters.Logger` method), 19

`error()` (`modelparameters.utils.Logger` method), 27

`eval_param_expr()` (in module `modelparameters.parameters`), 25

`expr` (`modelparameters.parameterdict.SlaveParam` attribute), 18

`expr` (`modelparameters.parameters.SlaveParam` attribute), 25

## F

`float_format()` (in module `modelparameters.config`), 14

`float_format()` (in module `modelparameters.parameters`), 19

`float_format()` (in module `modelparameters.utils`), 28

`flush_logger()` (`modelparameters.logger.Logger` method), 14

`flush_logger()` (`modelparameters.parameters.Logger` method), 19

`flush_logger()` (`modelparameters.utils.Logger` method), 27

`format()` (`modelparameters.parameters.Range` method), 22

`format()` (`modelparameters.utils.Range` method), 29

`format_data()` (`modelparameters.parameterdict.Param` method), 16

`format_data()` (`modelparameters.parameterdict.ParameterDict` method), 18

`format_data()` (`modelparameters.parameterdict.SlaveParam` method), 18

`format_data()` (`modelparameters.parameters.Param` method), 23

`format_data()` (`modelparameters.parameters.SlaveParam` method), 25

`format_in()` (`modelparameters.parameters.Range` method), 22

`format_in()` (`modelparameters.utils.Range` method), 29

`format_not_in()` (`modelparameters.parameters.Range` method), 22

`format_not_in()` (`modelparameters.utils.Range` method), 29

`format_time()` (in module `modelparameters.utils`), 29

`format_width()` (`modelparameters.parameterdict.Param` method), 16

`format_width()` (`modelparameters.parameters.Param` method), 23

`fromkeys()` (`modelparameters.parameterdict.ParameterDict` method), 18

## G

`get_log_handler()` (`modelparameters.logger.Logger` method), 14

`get_log_handler()` (`modelparameters.parameters.Logger` method), 20

`get_log_handler()` (`modelparameters.utils.Logger` method), 27

`get_log_level()` (`modelparameters.logger.Logger` method), 14

`get_log_level()` (`modelparameters.parameters.Logger` method), 20

`get_log_level()` (`modelparameters.utils.Logger` method), 27

`get_logfile_handler()` (`modelparameters.logger.Logger` method), 14

`get_logfile_handler()` (`modelparameters.parameters.Logger` method), 20

`get_logfile_handler()` (`modelparameters.utils.Logger` method), 27

`get_logger()` (`modelparameters.logger.Logger` method), 14

`get_logger()` (`modelparameters.parameters.Logger` method), 20

`get_logger()` (`modelparameters.utils.Logger` method), 27

`get_output()` (in module `modelparameters.commands`), 13

`get_status_output()` (in module `modelparameters.commands`), 13

`get_status_output_errors()` (in module `modelparameters.commands`), 13

`get_sym()` (`modelparameters.parameterdict.ScalarParam` method), 17

`get_sym()` (`modelparameters.parameters.ScalarParam` method), 24

`getvalue()` (`modelparameters.parameterdict.Param` method), 16

`getvalue()` (`modelparameters.parameterdict.SlaveParam` method), 18

`getvalue()` (`modelparameters.parameters.Param` method), 23

`getvalue()` (`modelparameters.parameters.SlaveParam` method), 25

## I

`info()` (`modelparameters.logger.Logger` method), 14

`info()` (`modelparameters.parameters.Logger` method), 20

`info()` (`modelparameters.utils.Logger` method), 27

`info_blue()` (`modelparameters.logger.Logger` method), 14

`info_blue()` (`modelparameters.parameters.Logger` method), 20

`info_blue()` (`modelparameters.utils.Logger` method), 27

`info_green()` (`modelparameters.logger.Logger` method), 14

`info_green()` (`modelparameters.parameters.Logger` method), 20

`info_green()` (`modelparameters.utils.Logger` method), 28

`info_red()` (`modelparameters.logger.Logger` method), 14

`info_red()` (`modelparameters.parameters.Logger` method), 20

`info_red()` (`modelparameters.utils.Logger` method), 28

`is_iterable()` (in module `modelparameters.utils`), 29

`iter_symbol_params_from_expr()` (in module `modelparameters.sympytools`), 27

`iterparameterdicts()` (`modelparameters.parameterdict.ParameterDict` method), 18

`iterparams()` (`modelparameters.parameterdict.ParameterDict` method), 18

## J

`juliacode()` (in module `modelparameters.codegeneration`), 11

## L

`latex()` (in module `modelparameters.codegeneration`), 11

`latex_unit()` (in module `modelparameters.codegeneration`), 11

`list_timings()` (in module `modelparameters.utils`), 29

`listwrap()` (in module `modelparameters.utils`), 30

`log()` (`modelparameters.logger.Logger` method), 14

`log()` (`modelparameters.parameters.Logger` method), 20

`log()` (`modelparameters.utils.Logger` method), 28

`Logger` (class in `modelparameters.logger`), 14

`Logger` (class in `modelparameters.parameters`), 19

`Logger` (class in `modelparameters.utils`), 27

## M

`matlabcode()` (in module `modelparameters.codegeneration`), 11

`modelparameters` (module), 31

`modelparameters.codegeneration` (module), 11

`modelparameters.commands` (module), 13

`modelparameters.config` (module), 14

`modelparameters.logger` (module), 14

`modelparameters.parameterdict` (module), 15

`modelparameters.parameters` (module), 19

`modelparameters.sympytools` (module), 25

`modelparameters.utils` (module), 27

## N

`name` (`modelparameters.parameterdict.Param` attribute), 16

`name` (`modelparameters.parameterdict.ScalarParam` attribute), 17

`name` (`modelparameters.parameters.Param` attribute), 23

`name` (`modelparameters.parameters.ScalarParam` attribute), 24

## O

`octavecode()` (in module `modelparameters.codegeneration`), 13

`OptionParam` (class in `modelparameters.parameterdict`), 17

`OptionParam` (class in `modelparameters.parameters`), 23

`optstr()` (`modelparameters.parameterdict.ParameterDict` method), 18

## P

`Param` (class in `modelparameters.parameterdict`), 15

`Param` (class in `modelparameters.parameters`), 22

param2value() (in module modelparameters.utils),  
28

ParameterDict (class in modelparameters.parameterdict), 18

parse\_args() (modelparameters.parameterdict.ParameterDict method),  
18

pop() (modelparameters.parameterdict.ParameterDict method), 19

pop\_log\_level() (modelparameters.logger.Logger method), 14

pop\_log\_level() (modelparameters.parameters.Logger method), 20

pop\_log\_level() (modelparameters.utils.Logger method), 28

push\_log\_level() (modelparameters.logger.Logger method), 14

push\_log\_level() (modelparameters.parameters.Logger method), 20

push\_log\_level() (modelparameters.utils.Logger method), 28

pythoncode() (in module modelparameters.codegeneration), 11

pythoncode() (in module modelparameters.parameters), 19

## Q

quote\_join() (in module modelparameters.utils), 31

## R

Range (class in modelparameters.parameters), 21

Range (class in modelparameters.utils), 29

reduce() (in module modelparameters.utils), 28

remove\_logfile() (modelparameters.logger.Logger method), 14

remove\_logfile() (modelparameters.parameters.Logger method), 20

remove\_logfile() (modelparameters.utils.Logger method), 28

repr() (modelparameters.parameterdict.OptionParam method), 17

repr() (modelparameters.parameterdict.Param method), 16

repr() (modelparameters.parameterdict.ScalarParam method), 17

repr() (modelparameters.parameters.OptionParam method), 23

repr() (modelparameters.parameters.Param method), 23

repr() (modelparameters.parameters.ScalarParam method), 24

resize() (modelparameters.parameterdict.ArrayParam method),  
17

resize() (modelparameters.parameters.ArrayParam method), 25

rjust() (in module modelparameters.utils), 28

## S

ScalarParam (class in modelparameters.parameterdict), 16

ScalarParam (class in modelparameters.parameters),  
24

set\_default\_exception() (modelparameters.logger.Logger method), 14

set\_default\_exception() (modelparameters.parameters.Logger method), 20

set\_default\_exception() (modelparameters.utils.Logger method), 28

set\_log\_handler() (modelparameters.logger.Logger method), 15

set\_log\_handler() (modelparameters.parameters.Logger method), 20

set\_log\_handler() (modelparameters.utils.Logger method), 28

set\_log\_indent() (modelparameters.logger.Logger method), 15

set\_log\_indent() (modelparameters.parameters.Logger method), 20

set\_log\_indent() (modelparameters.utils.Logger method), 28

set\_log\_level() (modelparameters.logger.Logger method), 15

set\_log\_level() (modelparameters.parameters.Logger method), 20

set\_log\_level() (modelparameters.utils.Logger method), 28

set\_log\_prefix() (modelparameters.logger.Logger method), 15

set\_log\_prefix() (modelparameters.parameters.Logger method), 20

set\_log\_prefix() (modelparameters.utils.Logger method), 28

set\_raise\_error() (modelparameters.logger.Logger method), 15

set\_raise\_error() (modelparameters.parameters.Logger method), 20

set\_raise\_error() (modelparameters.utils.Logger method), 28

setvalue() (modelparameters.parameterdict.ArrayParam method),  
18

setvalue() (modelparameters.parameterdict.Param method), 16

setvalue() (modelparameters.parameterdict.SlaveParam method),  
18

`setvalue()` (*modelparameters.parameters.ArrayParam method*), 25  
`setvalue()` (*modelparameters.parameters.Param method*), 23  
`setvalue()` (*modelparameters.parameters.SlaveParam method*), 25  
`SlaveParam` (class in *modelparameters.parameters.parameterdict*), 18  
`SlaveParam` (class in *modelparameters.parameters*), 25  
`store_symbol_parameter()` (*in module modelparameters.parameters*), 19  
`store_symbol_parameter()` (*in module modelparameters.sympytools*), 26  
`suppress_logging()` (*modelparameters.logger.Logger method*), 15  
`suppress_logging()` (*modelparameters.parameters.Logger method*), 20  
`suppress_logging()` (*modelparameters.utils.Logger method*), 28  
`sym` (*modelparameters.parameterdict.ScalarParam attribute*), 17  
`sym` (*modelparameters.parameters.ScalarParam attribute*), 24  
`symbol_param_value_namespace()` (*in module modelparameters.sympytools*), 27  
`symbol_params_from_expr()` (*in module modelparameters.sympytools*), 27  
`symbol_to_param()` (*in module modelparameters.parameters*), 19  
`symbol_to_param()` (*in module modelparameters.sympytools*), 26  
`symbol_to_params()` (*in module modelparameters.sympytools*), 26  
`symbols_from_expr()` (*in module modelparameters.parameters*), 19  
`symbols_from_expr()` (*in module modelparameters.sympytools*), 26  
`sympycode()` (*in module modelparameters.codegeneration*), 11  
`sympycode()` (*in module modelparameters.parameters*), 19

## T

`tic()` (*in module modelparameters.utils*), 29  
`Timer` (class in *modelparameters.parameters*), 22  
`Timer` (class in *modelparameters.utils*), 29  
`timings()` (*modelparameters.parameters.Timer class method*), 22  
`timings()` (*modelparameters.utils.Timer class method*), 29  
`toc()` (*in module modelparameters.utils*), 29  
`tuplewrap()` (*in module modelparameters.parameters*), 22

`tuplewrap()` (*in module modelparameters.utils*), 30  
`type_error()` (*modelparameters.logger.Logger method*), 15  
`type_error()` (*modelparameters.parameters.Logger method*), 20  
`type_error()` (*modelparameters.utils.Logger method*), 28  
`TypelessParam` (class in *modelparameters.parameters*), 24

## U

`unit` (*modelparameters.parameterdict.ScalarParam attribute*), 17  
`unit` (*modelparameters.parameters.ScalarParam attribute*), 24  
`update()` (*modelparameters.parameterdict.Param method*), 16  
`update()` (*modelparameters.parameterdict.ParameterDict method*), 19  
`update()` (*modelparameters.parameterdict.ScalarParam method*), 17  
`update()` (*modelparameters.parameters.Param method*), 23  
`update()` (*modelparameters.parameters.ScalarParam method*), 24

## V

`value` (*modelparameters.parameterdict.ArrayParam attribute*), 18  
`value` (*modelparameters.parameterdict.Param attribute*), 16  
`value` (*modelparameters.parameterdict.SlaveParam attribute*), 18  
`value` (*modelparameters.parameters.ArrayParam attribute*), 25  
`value` (*modelparameters.parameters.Param attribute*), 23  
`value` (*modelparameters.parameters.SlaveParam attribute*), 25  
`value_error()` (*modelparameters.logger.Logger method*), 15  
`value_error()` (*modelparameters.parameters.Logger method*), 20  
`value_error()` (*modelparameters.utils.Logger method*), 28  
`value_formatter()` (*in module modelparameters.parameters*), 21  
`value_formatter()` (*in module modelparameters.utils*), 28  
`VALUE_JUST()` (*in module modelparameters.utils*), 28  
`value_namespace()` (*in module modelparameters.parameters*), 19

`value_namespace()` (*in module `modelparameters.sympytools`*), [27](#)

## W

`warning()` (*modelparameters.logger.Logger method*), [15](#)

`warning()` (*modelparameters.parameters.Logger method*), [20](#)

`warning()` (*modelparameters.utils.Logger method*), [28](#)

`wrap_log_message()` (*modelparameters.logger.Logger method*), [15](#)

`wrap_log_message()` (*modelparameters.parameters.Logger method*), [21](#)

`wrap_log_message()` (*modelparameters.utils.Logger method*), [28](#)