

---

# **ETS Documentation**

*Release 4.1.3.dev0*

**Enthought**

October 09, 2017



---

## Contents

---

<b>1</b>	<b>SciMath Units Documentation</b>	<b>3</b>
1.1	Scimath Units User Manual Copyright Notice . . . . .	3
1.2	Introduction to SciMath Units . . . . .	4
1.3	Units with Numpy . . . . .	5
1.4	Unitted Functions . . . . .	6
1.5	Extending the Unit Parser . . . . .	7
1.6	Types of Units Available . . . . .	8
1.7	Scimath Units User Reference . . . . .	11
<b>2</b>	<b>SciMath Interpolate Documentation</b>	<b>13</b>
<b>3</b>	<b>SciMath Mathematics Documentation</b>	<b>15</b>
<b>4</b>	<b>Indices and tables</b>	<b>17</b>



The SciMath project includes packages to support scientific and mathematical calculations.

Contents:



Contents:

### Scimath Units User Manual Copyright Notice

**Authors** Tim Diller

**Version** Document Version 1

**Copyright** 2011 Enthought, Inc. All Rights Reserved.

Redistribution and use of this document in source and derived forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source or derived format (for example, Portable Document Format or Hypertext Markup Language) must retain the above copyright notice, this list of conditions and the following disclaimer.
- Neither the name of Enthought, Inc., nor the names of contributors may be used to endorse or promote products derived from this document without specific prior written permission.

THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

All trademarks and registered trademarks are the property of their respective owners.

Enthought, Inc.

515 Congress Avenue  
Suite 2100  
Austin TX 78701  
1.512.536.1057 (voice)  
1.512.536.1059 (fax)  
<http://www.enthought.com>  
[info@enthought.com](mailto:info@enthought.com)

## Introduction to SciMath Units

A large number of units used in science and engineering are available for use, and `scimath.units` makes working with and converting among them easy.

### Getting Started

We can import unit objects from the `scimath.units` submodules. (See *list of available units* for a listing by submodule.) There are submodules for many categories of physical quantities.

```
>>> from scimath.units.length import foot, inch, meter
>>> 3 * foot
0.9144000000000001*m
>>> foot / inch
12.0
>>> foot / meter
0.3048
```

Unit objects display natively in SI fundamental units and can be used to generate unit conversion factors or to see how units relate to one another.

```
>>> from scimath.units.pressure import pascal
>>> pascal * meter ** 2
1.0*m*kg*s**-2
```

**Caution:** If you are using this tool to produce conversion factors, remember that the factor is the *inverse* of what it looks like. Above, we divided one foot by one meter to get the ratio 0.3048. That is, `foot / meter` yields the number of meters per foot. Be careful to think through the logic clearly when developing conversion factors.

You can define your own arbitrary units and use them for calculating conversion factors:

```
>>> from scimath.units.length import inch
>>> from scimath.units.force import lbf
>>> from scimath.units.pressure import torr
>>> my_psi = 2 * lbf / inch ** 2
>>> my_psi / torr
103.44718363855331
```

Internally, they are stored as a *derivation* of fundamental physical quantities expressed in the SI system

```
>>> my_psi
13789.509579019157*m**-1*kg*s**-2
```

Conversions can be made between units with the same derivation using `convert()`:



```
>>> from scimath.units.api import convert
>>> from scimath.units.force import lbf, newton
>>> convert(1, lbf, newton)
4.44822
```

However, adding incompatible units raises an `IncompatibleUnits` exception:

```
>>> from scimath.units.electromagnetism import volt
>>> from scimath.units.mass import kilogram
>>> 1 * volt + 2 * kilogram
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "scimath/units/unit.py", line 62, in __add__
    raise IncompatibleUnits("add", self, other)
scimath.units.unit.IncompatibleUnits: Cannot add quantities with units of
↳ 'm**2*kg*s**-3*A**-1' and 'kg'
```

## Units with Numpy

For high-performance computation, `Scimath.units` includes two objects for adding units to `Numpy ndarray` objects: the `UnitScalar` and the `UnitArray`. `UnitScalars` and `UnitArrays` can be used directly in computations but are best handled with *united functions* constructed using the `has_units()` decorator.

### Working with UnitScalars and UnitArrays

#### UnitScalar example

As a basic example using scalar values, let's create two `UnitScalars` and add them. Say we were averaging the wing spans of African swallows (a) with that of European swallows (e).

```
>>> from scimath.units.api import UnitScalar
>>> a = UnitScalar(5, units="inches")
>>> e = UnitScalar(15, units="cm")
>>> (a + e) / 2
UnitScalar(5.452755905511811, units='0.025400000000000002*m')
```

Note that the result is a `UnitScalar` whose default units are *inches*, but it displays as a derivation from the SI fundamental unit *meters*. This is because `UnitScalar` values are stored internally as derivations of the SI system, thus if the value is assigned in a non-SI unit, part of the value may appear in the units. (See the section on *internal representation* for more detail.)

```
>>> UnitScalar(1, units="inch")
UnitScalar(1, units='0.025400000000000002*m')
```

A `UnitScalar` assumes the units that are first assigned to it. From the example above,

```
>>> (e + a) / 2
UnitScalar(13.85, units='0.01*m')
```

This does not fundamentally change the value of the variable, but it can lead to rounding errors and unexpected results:

```
>>> (e + a) == (a + e)
UnitScalar(False, units='None')
```

since

```
>>> (a + e) - (e + a)
UnitScalar(1.7763568394002505e-15, units='0.025400000000000002*m')
```

which is awfully close but not quite equal to zero.

## UnitArray example

A UnitArray uses a Numpy ndarray as its value.

```
>>> from numpy import linspace
>>> a = UnitArray(linspace(0, 5, 6), units="cm")
>>> a
UnitArray([ 0.,  1.,  2.,  3.,  4.,  5.], units='0.01*m')
```

UnitArrays can be multiplied by UnitScalars or other UnitArrays, as in NumPy.:

```
>>> a * b
UnitArray([0.,  1.,  2.,  3.,  4.,  5.], units='0.0030480000000000004*m**2')
```

Note that part of the value is contained in the unit string.

For high-performance computation with UnitArrays use *unitted functions*.

## Unitted Functions

A function which handles UnitArrays and UnitScalars is a unitted function. Unitted functions are created with the `has_units()` decorator. The units can be specified by passing arguments to the decorator or by constructing a special docstring.

### Decorator arguments

```
from numpy import array
from scimath.units.api import has_units
from scimath.units.length import feet, meter
@has_units(inputs="a:an array:units=ft;b:array:units=ft",
           outputs="result:an array:units=m")
def add(a,b):
    """ Add two arrays in ft and convert them to m.

    """
    return (a + b) * feet / meter
```

To use `has_units` with decorator arguments, pass string arguments “inputs” and (optionally) “outputs”. See the `has_units()` docstring or visit the [User Reference page](#) for details on the syntax.

### Formatted docstring

```
from scimath.units.api import has_units, UnitArray
@has_units
def add(a,b):
```

```

""" Add two arrays in ft and convert them to m.

Parameters
-----
a : array : units=ft
    An array
b : array : units=ft
    Another array

Returns
-----
c : array : units=m
    c = a + b
"""
return (a + b) * feet / meter

```

Using the `has_units` decorator with a docstring has the benefit of using a ReST-compatible format, so the uniting specification doubles as a documentation entry. See the `has_units()` docstring or visit the [User Reference page](#) for details on the syntax. The example above produces the following documentation entry when built with Sphinx:

## Unitted function output

In the examples above, we told `add()` to expect two values, `a` and `b` and convert them to feet for use in the function. Then we specified that the output would be in meters. Inside the function, `a` and `b` are not unitted, and the function is responsible for the conversion. (Remember our *caveat* regarding conversion factors.)

Unitted functions can accept either regular Python objects (of the appropriate type) or the equivalent unitted objects. The return type depends on what it was passed.

```

>>> add(1,2)
0.9144000000000001

```

In this case, `add()` accepted two integer arguments in feet, added them and returned an integer value in meters.

```

>>> add(UnitScalar(1, units="foot"), UnitScalar(2, units="foot"))
UnitScalar(0.9144000000000001, units='1.0*m')

```

In this case, `add()` accepted two `UnitScalar` arguments in feet and returned a `UnitScalar` in pure meters.

```

>>> add(UnitScalar(0.5, units="meter"), UnitScalar(50, units="cm"))
UnitScalar(1.0, units='1.0*m')

```

Finally, in this case, a conversion to feet was made for the calculation inside the function, and the value was converted back to meters when returned.

If no units are specified in the outputs or if no output string is given, then a regular scalar data type will be returned.

It may be useful to define new units for use in a project and have them available throughout an application. This is done by *extending the unit parser* to handle user-defined units, described in the next section.

## Extending the Unit Parser

If you define a series of units that you want to be available for import and use in unitted functions, then you must use the `unit_parser` as in the following file, imported later as `scimath/units/example_units`:

```
from scimath.units.api import unit_parser
from scimath.units.mass import kilogram

apple = 0.2 * kilogram
apple.label = 'label'

bread = 0.5 * kilogram
bread.label = 'loaf of bread'

very_small_rocks = 0.02 * kilogram
very_small_rocks.label = 'very small rocks'

a_duck = 1.3 * kilogram
a_duck.label = 'a duck'

import example_units as u
unit_parser.parser.extend(u)
```

When `example_units` is imported, `unit_parser` will be updated, and a unitted function can be built, as follows, from `scimath/units/example.py`:

```
from scimath.units.api import has_units
from scimath.units.example_units import a_duck

@has_units
def witch_test(mass_of_maiden):
    """ Test to determine if one or more young maidens is a witch.

    Parameters
    -----
    mass_of_maiden : array : units=a_duck
        array of masses to check against the weight of a duck

    Returns
    -----
    truth : bool
    """
    return mass_of_maiden >= 1
```

The behavior is as expected:

```
>>> from scimath.units.example import witch_test
>>> from scimath.units.mass import pound
>>> from scimath.units.api import UnitArray
>>> from numpy.random import randn
>>> maidens = UnitArray(randn(5)*15 + 100, units="pound")
>>> witch_test(maidens)
array([ True,  True,  True,  True,  True], dtype=bool)
```

## Types of Units Available

In the following sections, the units available from each `scimath.units` sub-module are listed. For convenience units are sometimes imported into a module from the module where they were defined.

## scimath.units.acceleration

f\_per\_s2, feet\_per\_second\_squared, ft\_per\_s2, m\_per\_s2, meters\_per\_second\_squared

## scimath.units.angle

circle, circles, deg, degree, degrees, gon, gons, grad, grads, math, mil, mils, minute, minutes, quadrant, quadrants, radian, radians, revolution, revolutions, right\_angle, right\_angles, second, seconds, sextant, sextants, sign, signs, turn, turns

## scimath.units.area

acre, barn, hectare, square\_centimeter, square\_foot, square\_inch, square\_meter, square\_mile

## scimath.units.density

g\_per\_c3, g\_per\_cc, g\_per\_cm3, gcc, gm\_per\_c3, gm\_per\_cc, gm\_per\_cm3, grams\_per\_cc, grams\_per\_cubic\_centimeter, kg\_per\_m3, kilograms\_per\_cubic\_meter, lb\_per\_gal, lb\_per\_gallon

## scimath.units.dimensionless

api, dimensionless, dim, fractional, fraction, gapi[#gamma\_ray]\_, one, parts\_per\_million, parts\_per\_one, pct, percent, percentage, ppm

## scimath.units.electromagnetism

amp, ampere, amps, amperes, coulomb, farad, henry, henrys, mho, micro\_farad, mA, milli\_ampere, milli\_amp, millivolts, mmho, mSiemen, mS, mv, ohms, ohmm, ohm\_m, ohm\_meter, ohms\_per\_m, ohms\_per\_meter, pf, pico\_farad, siemen, siemens\_per\_meter, siemens\_per\_m, tesla, teslas, uf, volts, v, weber, webers

## scimath.units.energy

Btu, Calorie, GeV, J, KeV, MJ, MeV, cal, calorie, eV, electron\_volt, erg, foot\_pound, horse\_power\_hour, joule, kJ, kcal, kilowatt\_hour

## scimath.units.force

lbf, lbs, N, newton

## scimath.units.frequency

Hz, RPM, hertz, hz, khz, kilohertz, rpm

## scimath.units.geo\_units

GPa, MPa, MPa\_per\_100f, MPa\_per\_100ft, MPa\_per\_f, MPa\_per\_ft, MPa\_per\_m, api<sup>1</sup>, apsi, becquerel, frac, fraction, fractional, g\_ft\_per\_cc\_s, g\_km\_per\_cc\_s, gapi<sup>1</sup>, gray, lb\_per\_gal, lb\_per\_gallon, parts\_per\_million, parts\_per\_one, pct, percent, percentage, ppg, ppm, psi\_per\_f, psi\_per\_ft, us\_fluid\_gallon, us\_per\_ft

## scimath.units.length

IN, angstrom, astronomical\_unit, centimeter, centimeters, cm, f, fathom, feet, fermi, foot, ft, inch, inches, kilometer, kilometers, km, light\_year, m, meter, meters, micrometer, micron, mile, millimeter, millimeters, mm, nanometer, nautical\_mile, nm, parsec, um, yard

## scimath.units.mass

centigram, cg, g, gm, gram, grams, kg, kilogram, kilograms, lb, lbs, metric\_ton, mg, milligram, ounce, pound, pounds, ton

## scimath.units.power

horsepower, kilowatt, kw, watt

## scimath.units.pressure

GPa, MPa, Pa, apsi, atm, atmosphere, bar, kPa, kbar, kbars, kilobar, millibar, pascal, punds\_per\_square\_inch, psi, psig, torr

## scimath.units.SI

ampere, atto, becquerel, candela, centi, copy, coulomb, deci, deka, dimensionless, exa, farad, femto, giga, gray, hecto, henry, hertz, joule, katal, kilo, kilogram, lumen, lux, mega, meter, micro, milli, mole, nano, newton, none, ohm, pascal, peta, pico, radian, second, siemens, sievert, steradian, tera, tesla, unit, volt, watt, weber, yocto, yotta, zepto, zetta

## scimath.units.speed

f\_per\_s, f\_per\_sec, feet\_per\_second, ft\_per\_s, ft\_per\_sec, kilometers\_per\_second, km\_per\_s, km\_per\_sec, knot, m\_per\_s, m\_per\_sec, meters\_per\_millisecond, meters\_per\_second, miles\_per\_hour

## scimath.units.substance

kmol, mol, mole

## scimath.units.temperature

K, celsius, degC, degF, degK, degc, degf, degk, fahrenheit, kelvin, rankine

---

<sup>1</sup> American Petroleum Institute units for gamma radiation

## scimath.units.time

day, hour, micro, microsecond, microseconds, milli, millisecond, milliseconds, minute, ms, msec, nano, nanosecond, ns, pico, picosecond, ps, s, sec, second, seconds, us, usec, year

## scimath.units.volume

barrel, bbl, c3, cc, centimeter, cm3, cubic\_centimeter, cubic\_foot, cubic\_inch, cubic\_meter, cuft, f3, ft3, gallon, gallons, liter, liters, m3, us\_fluid\_gallon, us\_fluid\_ounce, us\_fluid\_quart, us\_pint

# Scimath Units User Reference

## Internal Representation

Internally, a scimath unit is a unit object:

**class unit** (*value, derivation*)

### value

a scalar quantity which holds the magnitude of the unit, relative to the derivation in SI units.

### derivation

a 7-tuple holding the power of each fundamental quantity in the unit: (length, mass, time, electrical current, temperature, amount of substance, luminous intensity). The labels of the fundamental quantities are given in the attribute `_labels=('m', 'kg', 's', 'A', 'K', 'mol', 'cd')`

### label

the display name of the unit.

For example, the predefined unit Newton has the following attributes:

```

>>> from scimath.units.force import newton, lbf
>>> newton.value
1.0
>>> newton.derivation
(1, 1, -2, 0, 0, 0, 0)
>>> newton.label
'newton'
>>> lbf.value
4.44822
>>> lbf.derivation
(1, 1, -2, 0, 0, 0, 0)

```

## Limited API reference

### Convert

### HasUnits

### UnitScalar

UnitArray



## CHAPTER 2

---

### SciMath Interpolate Documentation

---

Contents:



## CHAPTER 3

---

### SciMath Mathematics Documentation

---

Contents:



## CHAPTER 4

---

### Indices and tables

---

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



**D**

derivation (unit attribute), 11

**L**

label (unit attribute), 11

**U**

unit (built-in class), 11

**V**

value (unit attribute), 11