Unit System (unit)

Version 0.1.2

Alex Baker

https://github.com/ambaker1/unit

June 30, 2023

Abstract

The "unit" package provides a framework for defining consistent units for engineering analysis with base units of force, length, time and thermodynamic temperature. Units in the "unit" package are defined with both a conversion value and a dimension vector - which allows for definition of any base unit system and unit conversion that checks for consistent units. In addition to conversion utilities, conversion values to user-defined base units can be imported into an analysis as variables for use in Tcl expressions. For convenience, the "unit" package provides a built-in set of typically used units for engineering analysis, but also provides tools to add new units, combine units, and remove units. The "unit" package also introduces a new notation for expressing combinations of units, which allows for easy introspection and manipulation of the unit system.

General

The unit package supplies one command, *unit*, which has a variety of subcommands for unit manipulation.

```
unit $subcommand $arg1 $arg2 ...
```

`$subcommand`	Subcommand for main command. Options listed below:
`combine`	Combine units using unit string notation.
`convert`	Convert between unit strings (with dimensionality check).
`exists`	Check if a unit is defined.
`expr`	Get unit conversion value associated with unit string.
`import`	Import units as variables.
`info`	Get dictionary of conversion value and dimension for unit.
`names`	Get list of defined units, optionally with dimensions specified.
`new`	Create a unit, with specified dimensions.
`remove`	Remove units.
`string`	Get unit string for specified dimensions.
`system`	Define or redefine the base units for the system of units.
`type`	Define, remove, or query unit type dimensions.
`types`	Get unit types, optionally with dimensions specified.
`wipe`	Clear all units and forget unit system.
`$arg1 $arg2 ...`	Arguments for subcommand.

Defining the System of Units

The command *unit system* queries, defines, or redefines the base units (force, length, time and temperature). Note that force is not technically a base unit, but it is convenient to consider it as one for engineering analysis. If called with no input arguments, it simply queries the current base units. Otherwise, it defines the base units. If initializing a unit system (after calling *unit wipe*), the specified base units will be created and initialized with values of 1.0. If redefining the unit system, the specified base units must already be defined in the existing unit system.

```
unit system <default> <$forceUnit $lengthUnit $timeUnit <$tempUnit>>
```

default	Option to wipe the unit system and restore default built-in unit system (N m sec K).
$forceUnit	Base unit for force (built-in options: in ft mm cm m)
$lengthUnit	Base unit for length (built-in options: kip lbf kN N kgf tonf)
$timeUnit	Base unit for time (built-in options: msec sec min hr)
$tempUnit	Base unit for temperature (built-in options: K R) (default K)

Example 1: Defining base unit system

Code:

```
unit system kip in sec
puts [unit system]
```

Output:

```
kip in sec K
```

Note for OpenSees users: If all quantities in your model are defined with units, the base units can be changed and all output quantities will be in the new base units. This is especially useful if you need to have results in US and SI units. A note of warning, however, is that units can show up in unexpected places. For example, convergence tests are essentially in base units. *NormUnbalance* & *RelativeNormUnbalance* test tolerances are in force units, *NormDispIncr*, *RelativeNormDispIncr* & *RelativeTotalNormDispIncr* test tolerances are in length units, and *EnergyIncr* & *RelativeEnergyIncr* test tolerances are in energy (force*length) units. Care must be taken to ensure that model is completely defined with units if it is desired to be able to change default base units.

Unit Dimensions and Types

The unit package stores both a conversion value and a unit dimension vector for each unit and combination of units. The dimension vector contains four values, representing the exponent on force, length, time, and temperature, respectively. For example, a force unit would have the dimension vector "`{1 0 0 0}`", and an acceleration unit would have the dimension vector "`{0 1 -2 0}`". Many of the commands in this module require dimension input, so to make input easier, unit types can be used instead. A unit type is a string that represents a dimension vector, such as using "`force`" as an alias for "`{1 0 0 0}`". Additionally, rather than a single dimension vector or dimension type, dimension vectors and types can be combined with exponents, in a key-value pair fashion. For example, the inputs "`acceleration`", "`{0 1 -2 0}`", "`length 1 time -2`", and "`{0 1 0 0} 1 {0 0 1 0} -2`" are all equivalent.

Create, Query, and Remove Unit Types

The command *unit type* can be used to add new unit types, remove existing unit types, and query the dimension vectors associated with the unit type.

```
unit type $typeName <$dimArg1 $dimArg2 ...>
```

`$typeName`	Unit type to query or modify
`$dimArg1 $dimArg2 ...`	Unit dimension arguments. If empty, it simply queries the dimension of the unit type. If the input is the blank string "", the unit type will be removed. Otherwise, it will define (or redefine) the unit type.

Example 2: Defining a new unit type

Code:

```
unit type momentum mass 1 velocity 1
puts [unit type momentum]
```

Output:

```
1 0 1 0
```

Note: the base unit types (force, length, time, temp) and the "constant" unit type cannot be modified or removed.

Get Unit Types

The command *unit types* returns a list of all unit types, with the option to specify the dimension.

```
unit types <$dimArg1 $dimArg2 ...>
```

$dimArg1 $dimArg2 ...	Unit dimension arguments. By default, returns all unit types. If valid dimension arguments are provided, returns all unit types matching the provided dimensions.

Example 3: Query unit types

Code:

```
puts [unit types mass 1 accel 1]
puts [unit types mass 1 length 2]
```

Output:

```
force
rotMass inertia
```

Built-In Units

Upon loading in the unit module, the unit system will already be defined in SI units (N m sec K) and a built-in set of units will be provided. If a different base unit system is desired, the unit system can be redefined with the command *unit system*, and all the built-in units will be redefined in the new unit system. Most of the built-in units, with their conversion values to SI units, are shown below:

Force Units	
N	1
kN	1000
MN	1000000
kgf	9.80665
tonf	9806.65
lbf	4.4482216152605
kip	4448.2216152605

Length Units	
m	1
mm	0.001
cm	0.01
km	1000
in	0.0254
ft	0.3048
yd	0.9144

Time Units	
sec	1
msec	0.001
min	60
hr	3600

Temperature Units	
K	1
R	0.555555555555556

Standard Gravity	
g	9.80665

Mass Units	
kg	1
gram	0.001
Mg	1000
slug	14.5939029372064
lbm	0.45359237

Stress/Pressure Units	
ksi	6894757.29316836
ksf	47880.2589803358
psi	6894.75729316836
psf	47.8802589803358
Pa	1
kPa	1000
MPa	1000000
GPa	1000000000
atm	101325

Constants	
pi	3.14159265358979
e	2.71828182845905
rad	1
deg	0.0174532925199433
cycle	6.28318530717959

Other unit types also have built-in units, such as Hz for frequency and kJ for energy. For brevity, these additional units and their conversion values are omitted, but all built-in units and their conversion values can be easily queried within the program.

Note: If *unit wipe* is called, these built-in units will be removed, and the unit system will be blank. They can be restored by calling *unit system default*.

Unit System Information

Besides querying the base units with *unit system*, the unit module provides commands for getting a list of other defined units, checking existence of units, and querying unit conversion values and unit base unit dimensions.

Get Listing of All Units

The command *unit names* returns a list of all defined units, with the option to specify dimension type.

```
unit names <$dimArg1 $dimArg2 ...>
```

$dimArg1 $dimArg2 ...	Unit dimension arguments. By default, returns all unit names. If dimension arguments are provided, returns all matching units.

Checking if a Unit Exists

The existence of a unit can be queried with the command *unit exists*. This is the same as checking if a unit is in the result of *unit names*, but is a more efficient implementation.

```
unit exists $unitName
```

$unitName	Name of unit to query.

Get Unit Info

Information about a specific unit can be queried for any defined unit with the command *unit info*. Returns the "unit info", which is a Tcl dictionary with the following key/value pairs:

- value: Conversion value to base units
- dimension: Dimension vector (force, length, time, temperature)

```
unit info $unitName
```

$unitName	Name of unit to query.

Importing Units as Variables

Units can be imported as Tcl variables for use in Tcl expressions, using the command *unit import*. Variables created by the import command will be created the caller's scope (e.g. if called in a procedure, it will create local variables in that procedure), the variable names will match the unit names, and the variable values will represent the conversion from that unit to base units.

```
unit import $arg1 $arg2 ...
```

$arg1 $arg2 ...	Units to import. -all for all units.

Example 4: Importing selected units

Code:

```
unit system kip in sec
unit import ft ksi
puts $ft
puts $ksi
```

Output:

```
12.0
1.0
```

Example 5: Importing all units

Code:

```
unit import -all
```

Note: As a safety measure, variable traces are added to unit variables to display a warning if overridden. Also, be sure to import units after defining the proper unit system. Redefining the unit system does not affect imported variables.

Unit Strings

The unit module uses a custom syntax for representing combinations of units, hereafter referred to as unit strings. Unit strings may contain the following:

1. Numbers (integer, decimal, and scientific)
2. Unit names (alphanumeric and underscore characters)
3. Whitespace
4. Parentheses `()`
5. Exponents ^
6. Multiplication `*`
7. Division `/`

For example, a unit string could be in^2, or kip*sec^2/in, or 1000*N/m, or gram/(cm*sec). Note that this format differs from Tcl expressions. Tcl expression math would be $in**2, $kip*$sec**2/$in, 1000*$N/$m, and $gram/($cm*$sec). The main difference is a lack of variable substitution and use of the caret symbol (^), instead of double asterisk (**), for exponentiation. The caret symbol has a different meaning than exponentiation in Tcl expressions.

Getting Base Unit String for Unit Type

The base-unit representation of a unit type can be queried with the command *unit string*.

```
unit string $dimArg1 $dimArg2 ...
```

`$dimArg1 $dimArg2 ...` Unit dimension arguments.

Example 6: Getting unit string for a unit type

Code:

```
unit system kip in sec
puts [unit string disp]
puts [unit string area]
```

Output:

```
in
in^2
```

Reduce Unit String to Base Unit String

A unit string can be reduced to base-unit representation with the command *unit reduce*.

```
unit reduce $unitString
```

$unitString	Unit string to reduce.

Example 7: Reducing a unit string

Code:

```
unit system kip in sec
puts [unit reduce 20*kN/m^2]
```

Output:

```
0.0029007547546041853*ksi
```

Unit String Expressions

The conversion value of a unit string to base units can be obtained using the *unit expr* command. Note that the expression syntax is limited in comparison to Tcl expressions. Only multiplication, division, exponents, and parentheses are allowed, with the caret symbol (^) representing exponentiation.

```
unit expr $arg1 $arg2 ...
```

$arg1 $arg2 ...	Arguments that combine to form a unit string. Same style as Tcl *expr* command.

Example 8: Unit expressions, two ways

Code:

```
unit system N m sec
puts [unit expr {10*cm}]
unit import cm
puts [expr {10*$cm}]
```

Output:

```
0.1
0.1
```

Unit Conversion

The command *unit convert* converts between compatible unit strings (equivalent base unit exponents) by computing a new conversion factor from the base unit conversions. If the dimension vectors of the two unit strings are not compatible, it will throw an error.

```
unit convert <$values> $unitString1 $unitString2
```

$values	Tcl list of values to convert. Default 1.0
$unitString1	Unit string to convert from
$unitString2	Unit string to convert to

Example 9: Unit conversions

Code:

```
# Independent of unit system
unit system N m sec
puts [unit convert 5 lbm kg]
unit system kip in sec
puts [unit convert 5 lbm kg]
# Works with unit strings
puts [unit convert 10 ft^2 in^2]
# Multiple conversion
puts [unit convert {10 20 30} lbf*in N*m]
```

Output:

```
2.26796185
2.26796185
1440.0
1.129848290276167 2.259696580552334 3.3895448708285008
```

Modifying the Unit System

Although the unit module comes prepackaged with a set of built-in units, the unit set may be modified using the commands *unit new*, *unit combine*, *unit remove* and *unit wipe*.

Adding a New Unit

The command *unit new* creates a new unit with a base unit conversion value and base dimensions. By default, base unit dimensions are assumed to be zero (constant). Returns the unit info of the new unit.

```
unit new $unitName $value <$dimArg1 $dimArg2 ...>
```

$unitName	Name of unit to add
$value	Conversion of unit to base units
$dimArg1 $dimArg2 ...	Unit dimension arguments. By default, unit is a constant.

Combining Existing Units

The command *unit combine* creates an aggregate unit using a unit string. With *unit combine*, the base unit dimensions are automatically calculated. Returns the unit info of the unit combination.

```
unit combine $unitString <$unitName>
```

$unitString	Unit string defining the unit.
$unitName	Name of unit to add. By default, does not add a new unit but just returns the unit info.

Example 10: Two ways to create a new unit

Code:

```
unit system kip in sec
unit import kg m
puts [unit new rho [expr {$kg/$m**3}] mass 1 length -3]
puts [unit combine kg/m^3 rho]
```

Output:

```
value 9.357254687402184e-11 dimension {1 -4 2 0}
value 9.357254687402184e-11 dimension {1 -4 2 0}
```

Note that both *unit new* and *unit combine* can be used to create the unit "rho". However, since "kg" and "m" were already created as units, *unit combine* was a much simpler option, since base unit exponents did not need to be specified. Ideally, *unit new* should only be used for base units; all other units can be derived using *unit combine*.

Removing Units

Units can be removed with the command *unit remove*. Note that this will not unset any variable associated with the unit if the unit was imported.

```
unit remove $arg1 $arg2 ...
```

`$arg1 $arg2 ...`	Units to remove. -all for all units.

Wiping the Unit System

All units and the unit system can be cleared with the command *unit wipe*. Note that wiping the unit system will require that the command *unit system* be called again before using other unit commands.

```
unit wipe
```

Example 11: Defining unit system from scratch

Code:

```
unit wipe
unit system kip in sec
unit new ft 12.0 length
unit new ksi 1.0 stress
unit combine ksi/1000.0 psi
unit combine in/2.54 cm
puts [unit names]
```

Output:

```
kip in sec K ft ksi psi cm
```