## 1.6 quanta - Module

Units and quantities handling
include "quanta.g"

### Description

*Introduction*
A quantity is a value with a unit. For example, '5km/s', or '20Jy/pc2'. This module (the `quanta` module) enables you to create and manipulate such quantities. The types of functionality provided are:

- *Conversion* Conversion of quantities to different units
- *Calculation* Calculations with quantities
- *GUI* A GUI interface is also provided

To access the `quanta` module, include the the *quanta.g Glish* script. This will create a default Quanta `tool` called **dq**.

```
- include 'quanta.g'
- dq.type()              # Default tool created for you
quanta
```

The Quanta `tool` has no state (it doesn't remember anything), so there is generally no need to create your own `tool`. Just use the default one. However, if you do require one, perhaps in a script, then use the `quanta` constructor to make one.

```
- include 'quanta.g';
- myqt := quanta();
- myqt.type();
quanta
```

The Quanta `tool` manipulates quantities. A quantity is stored as a *Glish* record with two fields. These fields are named 'value' and 'unit'. As well as simple scalar quantites, one can also create quantities as vectors or arrays. For example, you may have a vector of values, which all have the same unit - there is no need to store a copy of the unit for each value. Access to the individual fields of a quantum should always be by using the *getvalue* and *getunit* methods, especially since the internal names can change or be not accessable at some stage.

### Example

```
- dq.quantity(5.4, 'km/s')
[value=5.4, unit=km/s]
-
- q1 := dq.quantity([8.57132661e+09, 1.71426532e+10], 'km/s')
-
- dq.convert(q1, 'pc/h');
[value=[1 2] , unit=pc/h]
```

In the first example, we make a simple scalar quantity. You can see that the quantity (which is actually a *Glish* record) has fields 'value' and 'unit'.
In the second example, we make a vector quantity and then convert it from units of km/s to pc/h.

**Example**

```
- dq.quantity('5.4km/s')
[value=5.4, unit=km/s]
- dq.quantity(dq.unit('5.4km/s'))
[value=5.4, unit=km/s]
```

In the first example, the value and unit were combined into one string (just saves a bit of typing). The second example shows that the function `unit` is an alias for `quantity`, and that you can create a quantity from another quantity.

**Example**

```
- q1 := dq.unit("5s 5.4km/s")
- length(q1)
2
- q1[1]
[value=5, unit=s]
- q1[2]
[value=5.4, unit=km/s]
```

Here we make a vector quantity by using the Glish string vector (double quote) feature. You can see that the resultant quantity (*Glish* record) is of length 2 and that each field of that vector quantity is a scalar quantity. So you see that `q1` itself does not have fields 'value' and 'unit', only the elements of `q1` have that.

**Example**

```
- q1 := dq.unit('5km');
- q2 := dq.unit('200m');
- dq.canonical(dq.add(q1,q2))
[value=5200, unit=m]
```

Here we make two quantities with consistent but different units, add them together and then convert the result to canonical units.

**Example**

```
- q1 := dq.quantity('6rad');
- q2 := dq.quantity('3deg');
- dq.compare(q1,q2)
T
- dq.compare(q1,dq.unit('3km'))
F
```

Here we make compare the dimensionality of the units of two quantities.

**Example**

```
- q1 := dq.unit(array("5s 5.4km/s", 3, 2))
- length(q1)
6
- q1[1]
[value=5, unit=s]
- q1[2]
[value=5.4, unit=km/s]
- q1[6]
[value=5.4, unit=km/s]
- q1::
[id=quant, shape=[3 2] ]
```

Here we fill a quantity with a 2-dimensional array. In this case, the array is an array of strings. and its shape is [3,2]. We see that the resultant quantity is a vector of length 6 (it does not preserve the array shape here). However, we see that the shape attribute of the quantity does preserve the original array shape though (see below for more information about attributes).

*Constants, time and angle formatting*
If you would like to see all the possible constants known to the Quanta `tool` you can issue the command `dq.map('const')`. You can get the value of any constant in that list with a command such as

```
- boltzmann := dq.constants('k')
- print 'Boltzmann constant is ', boltzmann
Boltzmann constant is  [value=1.3806578e-23, unit=J/K]
```

There are some extra handy ways you can manipulate strings when you are dealing with times or angles. The following list shows special strings and string formats which you can input to the `quantity` function. Something in square brackets is optional. There are examples after the list.

- time: [+-]hh:mm:ss.t... – This is the preferred time format (trailing fields can be omitted)
- time: [+-]hhHmmMss.t..[S] – This is an alternative time format (HMS case insensitive, trailing second fields can be omitted)
- angle: [+-]dd.mm.ss.t.. – This is the preferred angle format (trailing fields after second priod can be omitted; dd.. is valid)
- angle: [+-]ddDmmMss.t...[S] – This is an alternative angle format (DMS case insensitive, trailing fields can be omitted after M)
- today – The special string "today" gives the UTC time at the instant the command was issued.
- today/time – The special string "today" plus the specified time string gives the UTC time at the specified instant
- yyyy/mm/dd[/time] – gives the UTC time at the specified instant
- dd[-]mmm[-][cc]yy[/time] – gives the UTC time at the specified instant in calendat style notation (23-jun-1999)

Note that the standard unit for degrees is 'deg', and for days 'd'. Formatting is done in such a way that it interprets a 'd' as degrees if preceded by a value without a period and if any value following it is terminated with an 'm'. In other cases 'days' are assumed. Here are some examples.

```
- dq.quantity('today')
[value=50611.2108, unit=d]
- dq.quantity('5jul1998')
[value=50999, unit=d]
- dq.quantity('5jul1998/12:')
[value=50999.5, unit=d]
- dq.quantity('-30.12.2')
[value=-30.2005556, unit=deg]
- dq.quantity('2:2:10')
[value=30.5416667, unit=deg]
- dq.unit('23h3m2.2s')
[value=345.759167, unit=deg]
```

Angles and times can often be used interchangeably. Special functions (`dq.totime()` and `dq.toangle()`) are available to make them in the right units for the purpose. E.g. `dq.sin(time)` gives an error, whereas `dq.sin(dq.toangle(time))` works ok. See units for more information on units and quantities, and the map function for pre-defined units.

```
- print a := dq.quantity('today');        # 1
[value=51045.9972, unit=d];
- print b := dq.toangle(a);               # 2
[value=320731.459, unit=rad]
- dq.angle(dq.norm(dq.toangle(a)));       # 3
-001.00.56.308
- dq.angle(dq.norm(dq.toangle(a), 0));    # 4
+358.59.03.692
- dq.sub('today',a);                      # 5
[value=0.00192880747, unit=d]
```

1. Get the time now
2. Get the time as an angle
3. Get the time as a normalised angle (-pi to +pi) and show as dms
4. Get the time as a normalised angle (0 to 2pi) and show as dms
5. Get time since creation of a

*Attributes*
Quantities (that is, the *Glish* record) have attributes attached to them. All quantities have the attribute called 'id' which has the string value 'quant'. This can be used to identify that a record is in fact a valid quantity. When you make quantities from arrays, you will generally find an attribute called "shape" attached to the quantity indicating what shape the array was. Attributes are accessed via the "::" operator.

**Example**

```
- q1 := dq.unit(array("5s 5.4km/s", 3, 2))
- length(q1)
6
- q1::
[id=quant, shape=[3 2] ]
```

**Functions**

| | |
|---|---|
| is_quantity | Check if quantity |

| | |
|---|---|
| r_array | Create an array or vector of records |
| r_fill | Fill only a slice of a record array |
| r_index | Get index array for a record array |
| r_slice | Get a slice of a record array |

**Tools**

| | |
|---|---|
| quanta | quanta tool |
| quantagui | GUI code for quanta.g |

### 1.6.1 is_quantity - Function

**Package general**
**Module quanta**

Check if quantity

**Synopsis**
is_quantity(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value to be tested<br>Allowed:any |

**Returns**
Bool

**Description**
Checks if the operand is a correct quantity

**Example**

```
- a := dq.quantity("5Jy") # make a quantity
- a
[value=5, unit=Jy]
- is_quantity(a) # is it one?
T
- is_quantity("5Jy") # and this string?
F
```

### 1.6.2 r_array - Function

**Package general**
**Module quanta**

Create an array or vector of records

### Synopsis

r_array(a, ..., id)

### Arguments

| | | |
|---|---|---|
| a | in | initial filling value(s)<br>Allowed:any or array of any<br>Default: empty record |
| ... | in | coordinate lengths for each dimension<br>Allowed:non-zero integer |
| id | in | optional record id<br>Allowed:string |

### Returns

record array

### Description

r_array creates an array of records. Record arrays were introduced to be able to communicate arrays/vectors of arbitrary data types to and from C++ code. Most special records (quanta, measures, skycomponents etc.) will have an id attribute to identify them across the system, but this id is not essential. Arrays of records have always a shape attribute, to distinguish them from a 'scalar' record. They are created in the same way as normal Glish arrays (except using r_array). The function accepts an additional, optional, argument to set an id if wanted. A vector with length one is assumed to be a scalar (a multidimensional array with 1 element will be an array).

### Example

```
- print a := r_array(1:5, 2, 3) # create an array
[*3=1, *4=2, *5=3, *6=4, *7=5, *8=1]
- a:: # show the attributes
```

```
[shape=[2 3] ]
- print a := r_array(1:5, 2, 3, id='quant') # another
[*9=1, *10=2, *11=3, *12=4, *13=5, *14=1]
- a:: # and its attributes
[id=quant, shape=[2 3] ]
- r_array(a,1) # a 'scalar'
1
- r_array(a,2,1) # initial with record array
[*17=1, *18=2]
```

---

### 1.6.3 r_slice - Function
**Package general**
**Module quanta**

Get a slice of a record array

**Synopsis**
r_slice(v, ...)

**Arguments**

| | | |
|---|---|---|
| v | in | a record array from which to extract elements<br>Allowed:record (array) |
| ... | in | slicing dimensions<br>Allowed:integers, an index record, integer vectors |

**Returns**
record array

**Description**
r_slice creates an array of records as a slice from another array of records. Slice indexing is identical to all the normal Glish options (including using a record index). The result will be another array of records (a scalar if 1 element is selected) with the id preserved.

**Example**

```
- print a := r_array(1:5, 2, 3) # create array
[*31=1, *32=2, *33=3, *34=4, *35=5, *36=1]
- a:: # its shape
[shape=[2 3] ]
-  r_slice(a,5) # one element index
5
- r_slice(a,5):: # gives a scalar
[=]
- r_slice(a,1,) # a slice
[*39=1, *40=3, *41=5]
- r_slice(a,1,):: # gives a vector
[shape=3]
- r_slice(a,1,2:3) # and another
[*49=3, *50=5]
- r_slice(a,1,2:3)::
[shape=2]
[*17=1, *18=2]
- r_slice(a,1:2,):: # an array slice
[shape=[2 3] ]
```

---

### 1.6.4 r_fill - Function
**Package general**
**Module quanta**

Fill only a slice of a record array

**Synopsis**
r_fill(what, ...)

**Arguments**

| | | |
|---|---|---|
| what | in | value(s) that will be filled cyclically in array<br>Allowed:any or vector/array of any |
| ... | in | slicing dimensions<br>Allowed:integers, an index record, integer vectors |

**Returns**
record array

**Description**
r_fill will overwrite a slice of an array of records. It is comparable to an indexed write for standard Glish arrays. The value given as a filler will be re-used cyclically.

**Example**

```
- x := dq.unit('5km/s') # a record
- x::
[id=quant]
- print a := r_array(x,3,2) # a record array
[*40=[value=5, unit=km/s], *41=[value=5, unit=km/s], *42=[value=5, unit=km/s],
 *43=[value=5, unit=km/s], *44=[value=5, unit=km/s], *45=[value=5, unit=km/s]]
- a::
[id=quant, shape=[3 2] ]
- r_fill(dq.unit('2s'), a, 2, 1) # overwrite an element
T
- a
[*40=[value=5, unit=km/s], *41=[value=2, unit=s], *42=[value=5, unit=km/s],
 *43=[value=5, unit=km/s], *44=[value=5, unit=km/s], *45=[value=5, unit=km/s]]
- r_fill(dq.unit(), a, 3,) # a few more
T
- a
[*40=[value=5, unit=km/s], *41=[value=2, unit=s], *42=[value=1, unit=],
 *43=[value=5, unit=km/s], *44=[value=5, unit=km/s], *45=[value=1, unit=]]
- b := dq.unit('2s') # see what happens
- b::shape := 2 # if made an array
- r_fill(b, a, 3,)
T
- a
[*40=[value=5, unit=km/s], *41=[value=2, unit=s], *42=2,
 *43=[value=5, unit=km/s], *44=[value=5, unit=km/s], *45=s]
```

---

### 1.6.5 r_index - Function
**Package general**
**Module quanta**

Get index array for a record array

**Synopsis**

r_index(v, ...)

**Arguments**

| | | |
|---|---|---|
| v | in | a record array from which to extract elements<br>Allowed:record (array) |
| ... | in | slicing dimensions<br>Allowed:integers, an index record, integer vectors |

**Returns**
integer array

**Description**
r_index is a helper function for the record array functions. It returns an integer array with the selected elements.

**Example**

```
- print a := r_array(1:6,3,2) # create a record array
[*29=1, *30=2, *31=3, *32=4, *33=5, *34=6]
- r_index(a) # its index
[[1:3,]
    1 4
    2 5
    3 6]
- r_index(a):: # its shape
[shape=[3 2] ]
- r_index(a,2,)
[2 5]
- r_index(a,2,)::
[shape=2]
- r_index(a,2,2)
5
- r_index(a,2,2)::
[=]
```

---

### 1.6.6 quanta - Tool

**Package general**
**Module quanta**

quanta tool

**Constructors**

| | |
|---|---|
| quanta | Construct quanta tool |

**Functions**

| | |
|---|---|
| abs | absolute value of quantity |
| acos | arccosine of quantity |
| add | add quantities |
| angle | show an angle as a formatted string |
| asin | arcsine of quantity |
| atan | arctangent of quantity |
| atan2 | arctangent of two quantity |
| canon | get canonical value of quantity |
| canonical | get canonical value of quantity |
| ceil | ceil value of quantity |
| check | check for proper unit string |
| checkfreq | check for proper frequency unit |
| compare | compare dimensionality of units |
| constants | get a constant |
| convert | convert a quantity to another unit |
| convertdop | convert a doppler velocity quantity to another unit |
| convertfreq | convert a frequency quantity to another unit |
| cos | cosine of quantity |
| define | define a new unit name |
| div | divides quantities |
| done | free resources used by tool |
| eq | compare quantities |
| exp | exponential of quantity |
| fits | define some FITS units |
| floor | floor value of quantity |
| form.xxx | Format a quantity using globaly set formats |
| ge | compare quantities |
| getformat | get current output format |
| getunit | get the internal unit of a quantity |
| getvalue | get the internal value of a quantity |
| gt | compare quantities |
| gui | Use the quanta GUI interface |
| id | identification of tool |
| is_angle | check if valid angle or time quantity |
| le | compare quantities |
| log | logarithm of quantity |
| log10 | logarithm of quantity |
| lt | compare quantities |

| | |
|---|---|
| map | list known unit names and constants |
| mul | multiply quantities |
| ne | compare quantities |
| neg | negate quantities |
| norm | normalise angle |
| pow | raise quantity to power |
| quantity | make a quantity from a string or from a value and a unit string |
| setformat | set format for output of numbers |
| sin | sine of quantity |
| splitdate | split a date/time into a record |
| sqrt | square root of quantity |
| sub | subtract quantities |
| tan | tangent of quantity |
| time | show a time (or date) as a formatted string |
| toangle | convert a time (or an angle) to an angle |
| tos | convert quantity to string |
| totime | convert an angle (or a time) to a time |
| type | type of tool |

**quanta - Constructor**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

Construct quanta tool

**Synopsis**
quanta(host, forceneweserver)

**Arguments**

| | | |
|---|---|---|
| host | in | host on which to run tool<br>Allowed:valid host name string<br>Default: ” |
| forceneweserver | in | force the use of a new server<br>Allowed:bool<br>Default: F |

**Returns**
tool

**Description**
Create a quanta `tool`on the specified host (or by default the host you are running on).

---

**quanta.gui - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

Use the quanta GUI interface

**Synopsis**
gui()

**Arguments**

**Returns**
Bool

**Description**

Start a GUI interface for the handling of units and quanta.
A menu bar will appear with 3 sets of menu headings.
The File menu contains options to close the current window (start again with dq.gui()).
The last menu is the Help menu, which can drive to different parts of the Reference manual. The '?' menu selects how popup help can be displayed (initially hover for short-help, MB3 down for longer help).
A way of starting automatically a GUI interface is by adding the following to your .aipsrc file

```
quanta.default: gui
```

Note that all actions available on the screen, are also available as tool functions for the command line interface. They are each described separately. The screen has a top area, and three other areas. The *Map* menu in the top area allows you to check which units are recognised, and which constants are available.

The quantity calculator works as an *RPN* calculator. Values are typed in the *Value* input field, the top of the stack. Values can have either the units attached (e.g. 5km/s), or detached (e.g. 5 mm.s/A), or be of the recognised time/angle possibilities (e.g. today, 12:15:2.2, 12.3.5.7 12d3m5.7, 1996/12/2/12h15m). The top of the stack is always shown in canonical units in the window below the action buttons (the right side of the window) Some simple actions are available on the bar below the entry window:

$\wedge$ pops stack to enter window (e.g. for editing)

$\vee$ pushes top of stack down onto stack (like Return)

$x <> y$ interchanges top two values on stack

***hand*** allow edit of top of stack

***dup*** duplicates top of stack

***copy*** copy top of stack to clipboard

***paste*** paste clipboard to top of stack

Note that the internal precision of the numbers is not reflected in the output precision.
The bottom entry is used to enter units used in the *convert* and *define* actions. The Operators (right-hand) window works on the top, or top two, values of the stack. Most actions will be clear to the user. The last action is displayed in the action window. The actions in the left column and bottom row are:

**pop** pops top of stack to enter window

**clear** clears the whole stack

**convert** converts the top of stack value to the units entered in the Units window (do not forget to hit Return). Note that with the possibilities of entering angles as 5d11m, there is a difficulty having days as a unit. This is solved by having a decimal point in the number. I.e. 5d will be interpreted as 5 degrees, 5.d as 5 days.

**canon** convert top of stack to canonical units

**define** define the top of stack as your own unit in the Units field (e.g. define the value;1Jy as your unit:JY)

**const** put the selected constant on top of stack

**FITS** define some FITS units for use (check them with Map:User)

The functions available to the non-GUI user are the same as to the GUI user. They are, a.o.: *quantity, canonical, convert, define, map, fits, angle, time, add, sub, mul, div, neg, norm, sin, cos, tan, asin, acos, atan, atan2, abs, ceil, floor,*

*compare, check, pow, constants, totime, toangle, tos* (see list at top of tool description).
If at any stage you want to use a GUI result on the command line, the 'copy' button of the GUI will transfer the last result to the clipboard. Most of the fields in the GUI have individual popup help, which can be viewed hovering on the field.
Trigonometric functions expect either angle units, or no units. In the latter case the selected units in the bottom of the right-hand side are used.

---

**quanta.setformat - Function**

| | |
|---:|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

set format for output of numbers

**Synopsis**
setformat(t, v)

**Arguments**

| | | |
|---|---|---|
| t | in | type<br>Allowed: coded string indicating which format parameter to set<br>Default: '' |
| v | in | format parameter value<br>Allowed: numeric or coded string, depending on format type to be set<br>Default: F |

**Returns**
bool

**Description**
setformat allows to define output format parameters for a variety of value types. The *type* is indicated by the first parameter, and coded (see following list). The actual format parameter set depends on the value parameter *v* given. If this value is specified as the '...' string, the value will be asked interactively from the user. The formats specified (or the defaults) are used in the form functions. The formatting that can be specified:

**prec** set the output precision for general numbers. If the value given is an integer less than one, the precision will be set to 6.

**aprec** set the output prescion for angles. If the value given is an integer less than one, the precision will be set to 6. A value of 6 will output up to the seconds field, with decimal places if larger than 6.

**tprec** set the output prescion for times. If the value given is an integer less than one, the precision will be set to 6. A value of 6 will output up to the seconds field, with decimal places if larger than 6.

**long** longitude display. Valid formats: 'hms', 'dms', 'deg', 'rad', 'd', '+deg'.

**lat** latitude display. Valid formats: 'hms', 'dms', 'deg', 'rad', 'd'.

**len** length display. Valid format: any length dimension

**dtime** time type display: 'local', 'utc', 'last' or 'solar'

**elev** elevation limit rise/set: angle quantity

**auto** auto frame update interval: time quantity

**vel** velocity: velocity unit

**freq** wave characteristics: frequency, length, time, angle/time, /length, energy, impulse units

**dop** doppler type display: 'radio', 'opt', 'true'

**unit** default units: any valid unit

### Example

```
- 1.234567890123456 # note that quanta set precision to 9
1.23456789 # (as compared to default Glish 6)
- a:=dq.quantity(1.2345678901234567, 'km/s')
- dq.form.vel(a)                # display as velocity in default units
1.23456789 km/s
- dq.setformat('vel', 'yd/a')   # try another unit
T
- dq.form.vel(a)
4.26071737e+10 yd/a
- dq.setformat('prec',12)       # or another precision
T
- dq.form.vel(a)
42607173719.8 yd/a
- dq.setformat('unit', 'm/s')   # set generic units
```

```
T
- dq.form.unit(a)              # and see what it looks like
1234.56789012 m/s
- b:=dq.quantity('5d2m3')      # an angle
- dq.form.long(b)              # as longitude gives it in hms
00:20:08.200
- dq.setformat('long', 'deg')  # display longitude in degrees
                               # (rather than default hms)
T
- dq.form.long(b)              # and show it
5.03416667 deg
- dq.form.lat(b)               # as a latitude it is default dms
+005.02.03.000
-
```

---

**quanta.getformat - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get current output format

**Synopsis**
getformat(t)

**Arguments**

| | | |
|---|---|---|
| t | in | type<br>Allowed:coded string<br>Default: ” |

**Returns**
coded format

**Description**
getformat returns the current format value set for the different format possibilities. See the setformat function for the different format type descriptions. The known types are:
prec, aprec, tprec, long, lat, len, dtime, elev, auto, vel, freq, dop, unit.

**Example**

```
- dq.getformat('prec')
6
- dq.setformat('prec', 12) # set precision to 12 significant digits
T
- dq.getformat('prec')
12
- dq.getformat('long')
hms
```

---

**quanta.form.xxx - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

Format a quantity using globaly set formats

**Synopsis**
form.xxx(v, showform)

**Arguments**

| | | |
|---|---|---|
| v | in | value to be converted<br>Allowed:quantity |
| showform | in | show square brackets and separating , (not for all)<br>Allowed:bool<br>Default: F |

**Returns**
string
If a multi-dimensional value is given for the value $v$ in the case of *dtime*, *long* or *lat*, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the *showform* is $T$, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ','.

**Description**

form.xxx (xxx can be lat, long, len, vel, freq, dtime, unit) will format the input into a string using the global format information set by setformat().

**Example**

```
- dq.setformat('freq','cm')
T
- dq.form.freq(dq.quantity('5GHz'))
5.99584916 cm
```

---

**quanta.convertfreq - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert a frequency quantity to another unit

**Synopsis**
convertfreq(v, out)

**Arguments**

| | | |
|---|---|---|
| v | in | quantity to convert<br>Allowed:quantity or string<br>Default: '1Hz' |
| out | in | unit to convert to<br>Allowed:string or quantity<br>Default: 'Hz' |

**Returns**
quantity

**Description**
convertfreq converts a frequency quantity to another unit. All units allowed in the setformat are allowed.

**Example**

```
- dq.convertfreq('5GHz','cm')
[value=5.99584916, unit=cm]
- dq.convertfreq('5cm','GHz')
[value=5.99584916, unit=GHz]
```

---

### quanta.convertdop - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert a doppler velocity quantity to another unit

**Synopsis**
convertdop(v, out)

**Arguments**

| | | |
|---|---|---|
| v | in | quantity to convert<br>Allowed:quantity or string<br>Default: '0km/s' |
| out | in | unit to convert to<br>Allowed:string or quantity<br>Default: 'km/s' |

**Returns**
quantity

**Description**
convertfreq converts a velocity quantity to another unit. Units are either velocity or dimensionless.

**Example**

```
- dq.convertdop('1','km/s')
[value=299792.458, unit=km/s]
- dq.convertdop('10km/s','1')
[value=3.33564095e-05, unit=]
```

---

### quanta.quantity - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

make a quantity from a string or from a value and a unit string

**Synopsis**
quantity(v, name)

**Arguments**

| | | |
|---|---|---|
| v | in | value to convert to quantity<br>Allowed:quantity or string or numeric<br>Default: '1' or 1.0 |
| name | in | unit string if v numeric<br>Allowed:string<br>Default: '' |

**Returns**
quantity

**Description**
quantity makes a quantity from a string, or from a value and a string. Note that a function *unit* exists which is a synonym for quantity. If only a string is given, it can be a scalar string, a vector of strings, or an array of strings. The result will be a scalar quantity, or an array of quantities. If a numeric value and a string are given, the numeric value can be any numeric type, and can also be an array or a vector of them. See the introduction for an example of all the possibilities.

**Example**

```
- tu := dq.quantity('1Jy') # make quantity
- tu
[value=1, unit=Jy]
- dq.quantity(tu) # also accepts a quantity
[value=1, unit=Jy]
- tu := dq.unit('1Jy') # make quantity with synonym
```

```
- tu
[value=1, unit=Jy]
- dq.quantity(-1.3, 'Jy') # make quantity with separate value
[value=-1.3, unit=Jy]
```

---

**quanta.getvalue - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get the internal value of a quantity

**Synopsis**
getvalue(v)

**Arguments**

| | | |
|---|---|---|
| v | in | quantity<br>Allowed:quantity or string or numeric |

**Returns**
numeric scalar or array

**Description**
getvalue returns the internal value of a quantity. It also can handle an array of quantities.

**Example**

```
- tu := dq.quantity(-1.3, 'Jy')         # make quantity
- tu
[value=-1.3, unit=Jy]
- dq.getvalue(tu)
-1.3
- dq.getunit(tu)
Jy
- a := dq.quantity("5m/s 7A")
```

```
- a::
[id=quant, shape=2]
- dq.getvalue(a)
[5 7]
```

---

**quanta.getunit - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get the internal unit of a quantity

**Synopsis**
getunit(v)

**Arguments**

| | | |
|---|---|---|
| v | in | quantity<br>Allowed:quantity or string or numeric |

**Returns**
string scalar or array

**Description**
getunit returns the internal unit string of a quantity

**Example**

```
- tu := dq.quantity(-1.3, 'Jy')         # make quantity
- tu
[value=-1.3, unit=Jy]
- dq.getvalue(tu)
-1.3
- dq.getunit(tu)
Jy
```

---

**quanta.canonical - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get canonical value of quantity

**Synopsis**
canonical(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value to convert<br>Allowed:quantity or string<br>Default: '1' |

**Returns**
quantity

**Description**
canonical (with alias canon) gets the canonical value of a quantity

**Example**

```
- dq.canonical('1Jy') # canonical value of a string
[value=1e-26, unit=kg.s-2]
- dq.canon(dq.quantity('1Jy')) # canonical value of a unit
[value=1e-26, unit=kg.s-2]
```

---

**quanta.canon - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get canonical value of quantity

**Synopsis**
canon(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value to convert<br>Allowed:quantity or string<br>Default: '1' |

**Returns**
quantity

**Description**
canon gets the canonical value of a quantity

**Example**

```
- dq.canon('1Jy') # canonical value of a string
[value=1e-26, unit=kg.s-2]
- dq.canonical(dq.quantity('1Jy')) # canonical value of a unit
[value=1e-26, unit=kg.s-2]
```

---

**quanta.convert - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert a quantity to another unit

**Synopsis**
convert(v, out)

**Arguments**

| | | |
|---|---|---|
| v | in | quantity to convert<br>Allowed:quantity or string<br>Default: '1' |
| out | in | unit to convert to<br>Allowed:string or quantity<br>Default: " |

**Returns**
quantity

**Description**
convert converts a quantity to another unit. If no output unit given, conversion is to canonical units

**Example**

```
- tu := dq.quantity('5Mm/s') # specify a quantity
- tu
[value=5, unit=Mm/s]
- dq.convert(tu, 'pc/a') # convert it to parsec per year
[value=0.00511356, unit=pc/a]
- dq.convert(tu) # convert to canonical units
[value=5e+06, unit=m.s-1]
```

---

**quanta.define - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

define a new unit name

**Synopsis**
define(name, v)

**Arguments**

| | | |
|---|---|---|
| name | in | name of unit to define<br>Allowed:string |
| v | in | quantity value of new unit<br>Allowed:string or quantity<br>Default: '1' |

**Returns**
bool

**Description**
define defines the name and value of a user defined unit

**Example**

```
- dq.define('JY','1Jy') # your misspelling
T
- dq.define('VLAunit', '0.898 JY') # a special unit using it
T
- dq.quantity('5 VLAunit')  # check its use
[value=5, unit=VLAunit]
- dq.convert('5 VLAunit','Jy')
[value=4.49, unit=Jy]
```

---

**quanta.map - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

list known unit names and constants

**Synopsis**
map(v)

**Arguments**

| | | |
|---|---|---|
| v | in | type of information to list<br>Allowed:coded string<br>Default: 'all' |

**Returns**
string

**Description**
map lists the known mapping of units and constants. It has a single argument, which can be a coded string (no-case, minimax match):

**all** all of the following units (not constants): also the default

**Prefix** known decimal prefixes

**SI** known SI units

**Customary** a set of customary units known to programs

**User** units defined by the user

**Constants** known constants (note: only 'const', 'Const', 'constants' and 'Constants' recognised).

**Example**

```
- dq.map('pre') # list decimal prefixes
        == Prefix ==== 20 ====
        E         (exa)                         1e+18
        G         (giga)                        1000000000
        M         (mega)                        1000000
        P         (peta)                        1e+15
        T         (tera)                        1e+12
        Y         (yotta)                       1e+24
        Z         (zetta)                       1e+21
        a         (atto)                        1e-18
        c         (centi)                       0.01
        d         (deci)                        0.1
        da        (deka)                        10
        f         (femto)                       1e-15
        h         (hecto)                       100
        k         (kilo)                        1000
        m         (milli)                       0.001
        n         (nano)                        1e-09
        p         (pico)                        1e-12
        u         (micro)                       1e-06
        y         (yocto)                       1e-24
        z         (zepto)                       1e-21
- dq.map('Constants') # list known constants
```

```
== Constants ====
pi    3.14..                   3.14159
ee    2.71..                   2.71828
c     light vel.               2.99792e+08 m/s
G     grav. const              6.67259e-11 N.m2/kg2
h     Planck const             6.62608e-34 J.s
HI    HI line                  1420.41 MHz
R     gas const                8.31451 J/K/mol
NA    Avogadro #               6.02214e+23 mol-1
e     electron charge          1.60218e-19 C
mp    proton mass              1.67262e-27 kg
mp_me mp/me                    1836.15
mu0   permeability vac.        1.25664e-06 H/m
eps0  permittivity vac.        1.60218e-19 C
k     Boltzmann const          1.38066e-23 J/K
F     Faraday const            96485.3 C/mol
me    electron mass            9.10939e-31 kg
re    electron radius          2.8179e-15 m
a0    Bohr's radius            5.2918e-11 m
R0    solar radius             6.9599e+08 m
k2    IAU grav. const^2        0.000295912 AU3/d2/S0
```

---

**quanta.fits - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

define some FITS units

**Synopsis**
fits()

**Arguments**

**Returns**
bool

**Description**
fits defines some unit names used in reading and writing FITS files.

**Example**

```
- dq.fits()
T
- dq.map('user')
        == User ==== 19 ====
        BEAM      (dimensionless beam)        1 _
        DAYS      (day)                       86400 s
        DEG       (degree)                    0.0174532925199 rad
        DEGREES   (degree)                    0.0174532925199 rad
        HZ        (hertz)                     1 s-1
        JY        (jansky)                    1e-26 kg.s-2
        KELVIN    (kelvin)                    1 K
        KELVINS   (kelvin)                    1 K
        KM        (km)                        1000 m
        M         (meter)                     1 m
        METERS    (meter)                     1 m
        PASCAL    (pascal)                    1 m-1.kg.s-2
        PIXEL     (dimensionless pixel)       1 _
        S         (second)                    1 s
        SEC       (second)                    1 s
        SECONDS   (second)                    1 s
        VOLTS     (volt)                      1 m2.kg.s-3.A-1
        YEAR      (year)                      31557600 s
        YEARS     (year)                      31557600 s
```

---

**quanta.angle - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

show an angle as a formatted string

**Synopsis**
angle(v, prec, form, showform)

**Arguments**

| | | |
|---|---|---|
| v | in | value to output<br>Allowed:angle quantity or string |
| prec | in | number of digits shown<br>Allowed:numeric<br>Default: 0 |
| form | in | formatting information<br>Allowed:coded string array<br>Default: " |
| showform | in | show square brackets and separating ,<br>Allowed:bool<br>Default: F |

**Returns**
string

**Description**
angle converts an angle quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes +-ddd.mm.ss) and a string array of codes (no-case, minimax match): Codes include:

**clean** delete leading/trailing superfluous separators

**no_d** do not show degrees part

**no_dm** do not show degrees and minutes part

**dig2** show only 2 digits of degrees in angle format

**time** show as time (hh:mm:ss.ttt) rather than as angle

If a multi-dimensional value is given for the value $v$, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the *showform* is $T$, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ','.

**Example**

```
- tu := dq.quantity('5.7.12.345678') # define an angle
- tu
[value=5.1201, unit=deg]
- dq.angle(tu)     # default output
+005.07.12
```

```
- dq.angle(tu, prec=7) # 7 digits
+005.07.12.3
- dq.angle(tu, prec=4) # 4 digits
+005.07.
- dq.angle(tu, form="tim no_d") # as time, no hours shown
:20:29
```

---

**quanta.time - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

show a time (or date) as a formatted string

**Synopsis**
time(v, prec, form, showform)

**Arguments**

| | | |
|---|---|---|
| v | in | value to output<br>Allowed:time quantity or string |
| prec | in | number of digits shown<br>Allowed:numeric<br>Default: 0 |
| form | in | formatting information<br>Allowed:coded string array<br>Default: ” |
| showform | in | show square brackets and separating ,<br>Allowed:bool<br>Default: F |

**Returns**
string

**Description**
time converts a time quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes hh.mm.ss) and a string array of codes (no-case, minimax match): Codes include:

**clean** delete leading/trailing superfluous separators

**no_d** do not show hours part

**no_dm** do not show hours and minutes part

**ymd** include a date as yyyy/mm/dd (date is by default not shown)

**dmy** include a date as ddMMMyyyy (date is by default not shown)

**mjd** include a date as Modified Julian Day (date is by default not shown)

**fits** include a date and show time in FITS format: le from OS

**angle** show in angle (dd.mm.ss.ttt) rather than time format

**day** prefix day-of-week to output

**local** show local time rather than UTC (add timezone offset)

**no_time** suppress printing of time part

If a multi-dimensional value is given for the value $v$, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the *showform* is $T$, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ','.

#### Example

```
- tu := dq.quantity('today') # a time
- tu
[value=50450.2243, unit=d]
- dq.time(tu) # default format
05:23:01.000
- dq.time(tu,form="dmy")   # show date
02-Jan-1997/05:23:01.000
- dq.time(tu,form="ymd day") # and day
Thu-1997/01/02/05:23:01.000
- dq.time(tu,form="fits")               # FITS format
1997-01-02T05:23:01.000
- dq.time(tu,form="fits local")         # local FITS format
1997-01-02T15:23:01.000+10:00
- dq.time(tu,form="ymd local")          # local time
1997/01/02/15:23:01.000
```

---

### quanta.add - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

add quantities

**Synopsis**
add(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
quantity

**Description**
add adds two quantities

**Example**

```
- dq.add('5m', '2yd')
[value=6.8288, unit=m]
```

---

**quanta.sub - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

subtract quantities

**Synopsis**
sub(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
quantity

**Description**
sub subtracts two quantities

**Example**

```
- dq.sub('5m', '2yd')
[value=3.1712, unit=m]
```

---

**quanta.mul - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

multiply quantities

**Synopsis**
mul(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '1' |

**Returns**
quantity

**Description**
mul multiplies two quantities

**Example**

```
- dq.mul('5m', '3s')
[value=15, unit=m.s]
```

---

**quanta.div - Function**

| | |
|---:|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

divides quantities

**Synopsis**
div(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '1' |

**Returns**
quantity

**Description**
div divides two quantities

**Example**

```
- dq.div('5m', '3s')
[value=1.66667, unit=m/(s)]
```

---

**quanta.neg - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

negate quantities

**Synopsis**
neg(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string<br>Default: 1 |

**Returns**
quantity

**Description**
neg negates a quantity

**Example**

```
- dq.neg('5m')
[value=-5, unit=m]
```

---

**quanta.norm - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

normalise angle

**Synopsis**

norm(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle quantity or string |
| a | in | lower interval boundary<br>Allowed:numeric<br>Default: -0.5 |

**Returns**
quantity

**Description**
norm normalise angles in interval of $2\pi$ radians. The default interval is from -0.5 to +0.5 of a full interval (i.e. from -180 to +180 degrees). The lower end of the interval can be set as a fraction of $2\pi$

**Example**

```
- dq.norm('713deg') #default normalisation
[value=-7, unit=deg]
- dq.norm('713deg', -2.5)  # normalise to interval -900 - -540 deg
[value=-727, unit=deg]
```

---

**quanta.le - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
le(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
le compares two quantities for less than or equal.

**Example**

```
- dq.le('5m', '2yd')
F
```

---

**quanta.lt - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
lt(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
lt compares two quantities for less than.

**Example**

```
- dq.lt('5m', '2yd')
F
```

---

**quanta.eq - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
eq(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
eq compares two quantities for equality.

**Example**

```
- dq.eq('5m', '2yd')
F
```

---

**quanta.ne - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
ne(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
ne compares two quantities for non equality.

**Example**

```
- dq.ne('5m', '2yd')
T
```

---

**quanta.gt - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
gt(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
gt compares two quantities for greater than.

**Example**

```
- dq.gt('5m', '2yd')
T
```

---

**quanta.ge - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare quantities

**Synopsis**
ge(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string<br>Default: '0' |

**Returns**
boolean

**Description**
ge compares two quantities for greater than or equal.

**Example**

```
- dq.ge('5m', '2yd')
T
```

---

**quanta.sin - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

sine of quantity

**Synopsis**
sin(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle quantity or string |

**Returns**
quantity

**Description**
sin gives sine of angle quantity

**Example**

```
- dq.sin('7deg')
[value=0.121869, unit=]
```

---

**quanta.cos - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

cosine of quantity

**Synopsis**
cos(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle quantity or string |

**Returns**
quantity

**Description**
cos gives cosine of angle quantity

**Example**

```
- dq.cos('7deg')
[value=0.992546, unit=]
```

---

**quanta.tan - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

tangent of quantity

**Synopsis**
tan(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle quantity or string |

**Returns**
quantity

**Description**
tan gives tangent of angle quantity

**Example**

```
- dq.tan('7deg')
[value=0.122785, unit=]
```

---

**quanta.asin - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

arcsine of quantity

**Synopsis**
asin(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:non-dimensioned quantity or string |

**Returns**
quantity

**Description**
asin gives arcsine of non-dimensioned quantity

**Example**

```
- dq.convert(dq.asin(dq.sin('7deg')), 'deg')
[value=7, unit=deg]
```

---

**quanta.acos - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

arccosine of quantity

**Synopsis**
acos(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:non-dimensioned quantity or string |

**Returns**
quantity

**Description**
acos gives arccosine of non-dimensioned quantity

**Example**

```
- dq.convert(dq.acos(dq.cos('7deg')), 'deg')
[value=7, unit=deg]
```

---

**quanta.atan - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

arctangent of quantity

**Synopsis**
atan(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:non-dimensioned quantity or string |

**Returns**
quantity

**Description**
atan gives arctangent of non-dimensioned quantity

**Example**

```
- dq.convert(dq.atan(dq.tan('7deg')), 'deg')
[value=7, unit=deg]
```

---

**quanta.atan2 - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

arctangent of two quantity

**Synopsis**
atan2(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:non-dimensioned quantity or string |
| a | in | value<br>Allowed:non-dimensioned quantity or string |

**Returns**
quantity

**Description**
atan gives arctangent of two non-dimensioned quantity

**Example**

```
- dq.convert(dq.atan2(dq.sin('7deg'), dq.cos('7deg')), 'deg')
[value=7, unit=deg]
```

---

**quanta.abs - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

absolute value of quantity

**Synopsis**
abs(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |

**Returns**
quantity

**Description**
abs gives absolute value of quantity

**Example**

```
- dq.abs('-5km/s')
[value=5, unit=km/s]
```

---

**quanta.ceil - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

ceil value of quantity

**Synopsis**
ceil(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |

**Returns**
quantity

**Description**
ceil gives ceiling value of quantity

**Example**

```
- dq.ceil('5.1AU')
[value=6, unit=AU]
```

---

### quanta.floor - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

floor value of quantity

**Synopsis**
floor(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |

**Returns**
quantity

**Description**
floor gives flooring value of quantity

**Example**

```
- dq.floor('-5.1AU')
[value=-6, unit=AU]
```

---

### quanta.log - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

logarithm of quantity

**Synopsis**
log(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:dimensionless quantity or string |

**Returns**
quantity

**Description**
log gives natural logarithm of dimensionless quantity

**Example**

```
- dq.log('2')
[value=0.693147181, unit=]
```

---

**quanta.log10 - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

logarithm of quantity

**Synopsis**
log10(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:dimensionless quantity or string |

**Returns**
quantity

**Description**
log10 gives logarithm of dimensionless quantity

**Example**

```
- dq.log10('2')
[value=0.301029996, unit=]
```

---

**quanta.exp - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

exponential of quantity

**Synopsis**
exp(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:dimensionless quantity or string |

**Returns**
quantity

**Description**
exp gives exponential value of dimensionless quantity

**Example**

```
- dq.exp('2')
[value=7.3890561, unit=]
- dq.exp('2m')
SEVERE: Caught an exception! Event type=run exception=Quantum::exp
       illegal unit type 'm'
```

---

**quanta.sqrt - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

square root of quantity

**Synopsis**
sqrt(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:dimensionless quantity or string |

**Returns**
quantity

**Description**
sqrt gives square root of quantity with only even powered dimensions

**Example**

```
- dq.sqrt('2m2')
[value=1.41421356, unit=m]
- dq.sqrt('2s')
SEVERE: Caught an exception! Event type=run exception=UnitVal::UnitVal
        Illegal unit dimensions for root
```

---

**quanta.compare - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

compare dimensionality of units

**Synopsis**
compare(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | value<br>Allowed:quantity or string |

**Returns**
bool

**Description**
compare compares the dimensionality of units of two qauntities

**Example**

```
- dq.compare('5yd/a', '6m/s')   # equal dimensions
T
- dq.compare('5yd', '5s') # unequal dimensions
F
```

---

**quanta.check - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

check for proper unit string

**Synopsis**
check(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:string or quantity |

**Returns**
bool

**Description**
check checks if the argument has a properly defined unit string

**Example**

```
- dq.check('5AE/Jy.pc5/s')
T
- dq.check('7MYs')
F
```

---

**quanta.checkfreq - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

check for proper frequency unit

**Synopsis**
checkfreq(cm)

**Arguments**

| | | |
|---|---|---|
| cm | in | value<br>Allowed:string or quantity |

**Returns**
bool

**Description**
checkfreq checks if the argument has a properly defined frequency interpretable unit string

**Example**

```
- dq.checkfreq('5GHz')
T
- dq.checkfreq('5cm')
T
- dq.checkfreq('5cm/s2')
F
```

---

**quanta.pow - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

raise quantity to power

**Synopsis**
pow(v, a)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| a | in | power<br>Allowed:integer<br>Default: 1 |

**Returns**
quantity

**Description**
pow raises a quantity to an integer power

**Example**

```
- dq.pow('7.2km/s', -3)
[value=0.00267918, unit=(km/s)-3]
```

---

**quanta.constants - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

get a constant

**Synopsis**
constants(v)

**Arguments**

| | | |
|---|---|---|
| v | in | name<br>Allowed:coded string<br>Default: 'pi' |

**Returns**
quantity

**Description**
constants gets a named constant quantity. Names (no-case, minimax) are:

```
pi    3.14..              3.14159
ee    2.71..              2.71828
c     light vel.          2.99792e+08 m/s
G     grav. const         6.67259e-11 N.m2/kg2
h     Planck const        6.62608e-34 J.s
HI    HI line             1420.41 MHz
R     gas const           8.31451 J/K/mol
NA    Avogadro #          6.02214e+23 mol-1
```

```
e     electron charge          1.60218e-19 C
mp    proton mass              1.67262e-27 kg
mp_me mp/me                    1836.15
mu0   permeability vac.        1.25664e-06 H/m
eps0  permittivity vac.        1.60218e-19 C
k     Boltzmann const          1.38066e-23 J/K
F     Faraday const            96485.3 C/mol
me    electron mass            9.10939e-31 kg
re    electron radius          2.8179e-15 m
a0    Bohr's radius            5.2918e-11 m
R0    solar radius             6.9599e+08 m
k2    IAU grav. const^2        0.000295912 AU3/d2/S0
```

**Example**

```
- dq.constants()
[value=3.14159, unit=]
```

---

**quanta.is_angle - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

check if valid angle or time quantity

**Synopsis**
is_angle(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle/time quantity or string |

**Returns**
bool

**Description**
is_angle checks if the argument is a valid angle/time quantity.

**Example**

```
- dq.is_angle(dq.constants('pi'))
F
```

---

**quanta.totime - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert an angle (or a time) to a time

**Synopsis**
totime(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle/time quantity or string |

**Returns**
time quantity

**Description**
totime converts an angle quantity (or a time) to a time

**Example**

```
- dq.totime('2d5m')
[value=0.00578704, unit=d]
```

---

**quanta.toangle - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert a time (or an angle) to an angle

**Synopsis**
toangle(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle/time quantity or string |

**Returns**
angle quantity

**Description**
toangle converts a time quantity (or an angle) to an angle

**Example**

```
- dq.toangle('5h30m12.6')
[value=82.5525, unit=deg]
```

---

**quanta.splitdate - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

split a date/time into a record

**Synopsis**
splitdate(v)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:angle/time quantity or string |

**Returns**
record

**Description**
splitdate splits a date/time quantity into a record with constituent fields like year, yearday, month etc. All fields will be integer (to enable use as index and easy personal formatting), with the exception of the *s* field which is a double float. See the example for the fields returned.

**Example**

```
- dq.splitdate('today')
[mjd=52075.0361, year=2001, yearday=166, month=6, monthday=15,
  week=24, weekday=5, hour=0, min=52, sec=1,
   s=1.24799991, msec=247, usec=247999]
- dq.splitdate('183.33333333deg')
[mjd=0.509259259, year=1858, yearday=321, month=11, monthday=17,
   week=46, weekday=3, hour=12, min=13, sec=19,
    s=19.9999992, msec=999, usec=999999]
```

---

**quanta.tos - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

convert quantity to string

**Synopsis**
tos(v, prec)

**Arguments**

| | | |
|---|---|---|
| v | in | value<br>Allowed:quantity or string |
| prec | in | convert precision of value<br>Allowed:integer<br>Default: 0 |

**Returns**
string

**Description**
tos converts a quantity to a string with the precision defined with the *setformat('prec')* (which defaults to 9). If the optional *prec* argument is set to an integer value greater than 1, that precision is used in the conversion

**Example**

```
- print a := dq.quantity('2.56 yd/s')
[value=2.56, unit=yd/s]
- dq.tos(a)
2.56 yd/s
- a:=dq.quantity(1/7, 'km/s')
- dq.tos(a)
0.142857143 km/s
- dq.tos(a,2)
0.14 km/s
- dq.tos(a,20)
0.14285714285714284921 km/s
- dq.tos(a)
0.142857143 km/s
```

---

**quanta.id - Function**

| | |
|---:|:---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

identification of tool

**Synopsis**
id()

**Arguments**

**Returns**
object id

**Description**
id will return the object identification. Its use is meant for the toolmanager.

**Example**

```
- dq.id()
[sequence=1, pid=4774, time=1020735188, host=cetus, agentid=3]
```

---

**quanta.type - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

type of tool

**Synopsis**
type()

**Arguments**

**Returns**
string

**Description**
type will return the tool name.

**Example**

```
- dq.type()
quanta
```

---

**quanta.done - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |
| **Tool** | **quanta** |

free resources used by tool

**Synopsis**
done(kill)

**Arguments**

| | | |
|---|---|---|
| kill | in | force kill of the default tool<br>Allowed:bool<br>Default: F |

**Returns**
bool

**Description**
done will free the resources used by the tool. If the tool is the default tool (*dm*) the done will only be executed if the kill argument is true.

**Example**

```
- a:=quanta()
- a.done()
T
- dq.done()
F
- dq.done(kill=T)
NORMAL: Successfully closed empty server: quanta
T
```

---

### 1.6.7 quantagui - Tool

| | |
|---|---|
| **Package** | **general** |
| **Module** | **quanta** |

GUI code for quanta.g
include "quantagui.g"

### Description

quantagui provides the code for the GUI interface to the units and quantities (quanta) closure. It cannot be used (or included) on its own. It will be automatically included and used whenever a GUI is asked for by the quanta closure (using dq.gui()).