Example

```
include 'msplot.g';
msplotapplyflags('3C273XC1.ms.flags.12')
```

This example applies the flags in the table called `3C273XC1.flags.12` to a measurements set called `3C273XC1.ms` in the same directory.

1.4 measures - Module

Measures handling
include "measures.g"

Description

A measure is a quantity with a specified reference frame (e.g. UTC, J2000, mars). The measures module provides an interface to the handling of measures. The basic functionality provided is:

- *Conversion* Conversion of measures, especially between different frames (e.g. UTC to LAST)
- *Calculation* Calculation of e.g. a rest frequency from a velocity and a frequency.

This functionality is provided in both a command line interface and a *GUI* interface.
To access the `measures` module, include the *measures.g Glish* script. This will create a default Measures `tool` called **dm**.
Altough a Measures `tool` has state (it remembers the frame you are working in), there is only in very special cases the need to create your own `tool` . Just use the default **dm** one. However, if you do require one, perhaps to distinguish working in two different environment frames for an input observation and and output display, then use the `measures` constructor to make one:

```
- include 'measures.g';
- mydm := measures(); # measures tool created for you
- mydm.type();
measures
```

In normal use, the **measures** `tool` will be operated using a *GUI*. All operations can be done from either the command line or the *GUI*. The aipsrc mechanism is used by the measures `tool` to specify certain operational details (like using DE405 rather than the default DE200 for planetary positions).

Measures

Measures are e.g. an epoch, or coordinates, which have, in addition to values (as quantities), also a reference specification, and possibly an offset. They are represented as records with fields describing the various entities embodied in the measure. These entities can be obtained by the access methods *gettype*, *getref*, *getoffset*. *getvalue*.

Each measure has its own list of reference codes (see the individual methods for creating them, like *direction*). If an empty or no code reference code is given, the default code for that type of measure will be used (e.g. it is *J2000* for a direction). If an unknown code is given, this default is also returned, but with a warning message.

The values of a measure (like the right-ascension for a direction) are given as quantities. Each of them can be either a scalar quantity with a scalar or vector for its actual value (see the following example). The values can also be given as an r_array, in which case each element of that array will be used as a value. E.g a vector of length 2 of quanta will be seen in a direction constructor as a longitude and a latitude.

```
- dm.epoch('utc','today') # note that your value will be different
[type=epoch, refer=UTC, m0=[value=51350.0778, unit=d]]
- dm.direction('j2000','5h20m','-30.2deg')
[type=direction, refer=J2000, m1=[value=-0.527089434, unit=rad],
  m0=[unit=rad, value=1.3962634]]
- a := dm.direction('j2000','5h20m','-30.2deg')
- dm.gettype(a)
direction
- dm.getoffset(a)
F
- dm.getref(a)
J2000
- dm.getvalue(a)
[*53=[unit=rad, value=1.3962634], *54=[value=-0.527089434, unit=rad]]
- dm.getvalue(a)[1]
[unit=rad, value=1.3962634]
- dm.getvalue(a)[2]
[value=-0.527089434, unit=rad]
- # try as a scalar quantity with multiple values
- a := dm.direction('j2000', dq.quantity([10,20],'deg'),
+ dq.quantity([30,40], 'deg'))
- dm.getvalue(a)[1]
[unit=rad, value=[0.174532925 0.34906585] ]
- dq.getvalue(dm.getvalue(a)[1])[2]
0.34906585
- a
[type=direction, refer=J2000, m1=[value=[0.523598776 0.698131701] , unit=rad],
 m0=[unit=rad, value=[0.174532925 0.34906585] ]]
```

```
- # try as an r_array
- b := r_array(,2)
- r_fill(dq.quantity([10,20],'deg'), b, 1)
T
- r_fill(dq.quantity([30,40],'deg'), b, 2)
T
- a := dm.direction('j2000', b)
- dm.getvalue(a)
[*61=[unit=rad, value=[0.174532925 0.34906585] ],
 *62=[value=[0.523598776 0.698131701] , unit=rad]]
```

Known measures are:

- epoch: an instance in time (internally expressed as MJD or MGSD)
- direction: a direction towards an astronomical object (including planets, sun, moon)
- position: a position on Earth
- frequency: electromagnetic wave energy
- radialvelocity: radial velocity of astronomical object
- doppler: doppler shift (i.e. radial velocity in non-velocity units like 'Optical', 'Radio'. The radialvelocity GUI caters for Doppler coding as well)
- baseline: interferometer baseline
- uvw: UVW coordinates
- earthmagnetic: Earth' magnetic field

In addition to the reference code (like J2000), a measure needs sometimes more information to be convertable to another reference code (e.g. a time and position to convert it to an azimuth/elevation). This additional information is called the reference *frame*, and can specify one or more of 'where am i', 'when is it', 'what direction", 'how fast'.
The frame values can be set by the doframe tool function, or by a menu on the measure guis.
Since you would normally work from a fixed position, the position frame element ('where you are'), can be specified in your .aipsrc if its name is in the Observatory list (obslist) tool function. You can set your preferred position by adding to your *.aipsrc* file:

```
measures.default.observatory: atca
```

More information on the individual measures can be found in the GUI descriptions for epoch; direction; position; frequency; radialvelocity; doppler.

Example

Suppose we have an object with a J2000 right ascension and declination, and want to know the time of rising above the horizon of a certain astronomical direction. In practice you would do this using the special application in the gui, but the following shows how to do it the hard and long way.

```
- include 'measures.g'# 1
T
- tim := dm.epoch('utc','today') # 2
- tim
[type=epoch, refer=UTC, m0=[value=51350.0934, unit=d]]
- pos := dm.observatory('ATCA') # 3
- dm.doframe(tim)  # 4
T
- dm.doframe(pos)                # 5
T
- coord := dm.direction('J2000', '5h20m30.2', '-30d15m12.5')  # 6
- hadec := dm.measure(coord,'hadec') # 7
- last := dm.measure(tim,'last') # 8
- sdec := dq.sin(dm.getvalue(hadec)[2]) # 9
- cdec := dq.cos(dm.getvalue(hadec)[2])
- clat := dq.cos(dm.getvalue(pos)[2])
- slat := dq.sin(dm.getvalue(pos)[2])
- ha := dq.acos(dq.neg(dq.div(dq.mul(sdec, slat),
 dq.mul(cdec, clat))))  #10
- dq.norm(dq.add(ha, dm.getvalue(coord)[1]), 0) #11
[value=189.926855, unit=deg]
- dq.time(dq.norm(dq.add(ha, dm.getvalue(coord)[1]), 0)) #12
12:39:42.445
- rt := dq.totime(dq.norm(dq.add(ha, dm.getvalue(coord)[1]))) #13
- rt
[value=-0.472425402, unit=d]
- rtoff := dm.epoch('r_utc', dm.getvalue(tim)[1]) #14
- rtoff
[type=epoch, refer=R_UTC, m0=[value=51350, unit=d]]
- rte := dm.epoch('last', rt, off=rtoff) #15
- rte
[type=epoch, refer=LAST, m0=[value=58062.5276, unit=d]]
- dm.measure(rte, 'utc') #16
[type=epoch, refer=UTC, m0=[value=51349.3674, unit=d]]
- dq.time(dm.getvalue(dm.measure(rte, 'utc'))[1], form="ymd time")
1999/06/20/08:48:59.738
- dq.time(dm.getvalue(dm.measure(dm.measure(rte, 'utc'), 'last'))[1]) #17
12:39:42.445
- # try it another way
- tim := dm.epoch('utc', 'today') #18
```

```
- tim := dm.measure(tim,'tai') #19
- dm.doframe(tim) #20
T
- dm.showframe(F) #21
position: 09:58:12.033 -030.08.43.736 6372960.26 m ITRF
epoch: 2002/10/18/00:15:49 UTC
T
- is_measure(tim) #22
T
- sun:=dm.direction('sun') #23
- dm.rise(sun) #24
[rise=[value=20.4413559, unit=deg], set=[value=158.042775, unit=deg]]
- dm.rise(sun,'10deg')  #25
[rise=[value=27.3170864, unit=deg], set=[value=151.167045, unit=deg]]
- dq.form.long(dm.rise(sun).rise) #26
01:21:45.925
- dq.setformat('long','dms') #27
T
- dq.form.long(dm.rise(sun).rise)
+020.26.28.881
```

1. Gives access to the measures, with a default server dm
2. Get the time now as an epoch measure
3. Get the position of an observatory
4. Set the time in reference frame. Note that if you are working in a GUI environment, the frame set will be displayed. If working in a pure CLI environment, the current frame can be shown with the showframe tool function.
5. Set the position in reference frame
6. Get the coordinates as a measure
7. Convert the coordinates to hour-angle, declination
8. Get the sidereal time for now (just for the fun)
9. Calculate the sines/cosines of declination and latitudes
10. Get the hour-angle for zero elevation
11. Get the sidereal time of setting (rising?), normalised between 0 and 360 degrees
12. Show it in a time format
13. Save it as a time (remember, we calculated it as an angle)

14. We would like to convert this sidereal time (rte) back to UTC. To be able to do that, we most now when (for which date) the sidereal time is valid. In general we do not know the Sidereal date, else it would be easy. We can however, specify an offset with a measure. Special reference codes are available (called 'raze'), which will after conversion only retain the integral part. This line says that we specify tim as an epoch in UTC, to be razed after conversion. If we now use this as an offset for a sidereal time, the offset will be automatically converted to sidereal time (since it has to be added to a sidereal time), razed, i.e. giving the Greenwich sidereal date, and hence the complete epoch is known. (Note that the indecision over the 4 minutes per day has to be handled in special cases).

15. We can now define our sidereal time rt as a real epoch

16. We now convert the sidereal time rte, to an UTC, and show it as a time as well

17. We do not trust this funny razing, and convert the UTC obtained straight back to a sidereal time. And, indeed it worked (compare line 12)

18. Another time

19. Convert it to TAI (just to show how to do it)

20. Frame it

21. Show the current frame (the F argument indicates to not use GUI if one present) (note that the position is set to ATCA, as defined by the .aipsrc variable). If an error happens, the command

    ```
    dm.doframe(dm.observatory('atca'))
    ```

    will solve it.

22. to check if we really got a measure, or an error occurred.

23. define the position of the sun (the actual position will be at the frame time)

24. get the sidereal time of rise/set of the sun (at the default elevation of 5 deg)

25. try again for an elevation of 10 deg

26. display the rise sidereal time as a 'longitude'. A variety of dq.form.x routines exist, using a global format setting mechanism (in general you have your preferred way to display something)

27. change the format, and show again

The above could be done using the GUI (start it with dm.gui()), setting your position in the Frame menu option, and using the rise/set application.
Another option would be to use the direction GUI (from the Tool menu in the general GUI or starting dm.directiongui() from the command line); selecting the Sun as planet; setting the position in the frame menu and the time; Convert to J2000, and select rise/set from the Info menu.
If at one stage you want to use a GUI result on the command line, the 'copy' button will transfer the last result to the clipboard (and read it with the normal clipboard operation

```
- dcb.paste()
[type=direction, refer=J2000, m1=[value=0.669116295, unit=rad], m0=[unit=rad, value=0.0259745717]]
```

Tools

measures	measures tool
measuresgui	gui code for measures.g

Functions

is_measure	Check if measure

1.4.1 is_measure - Function

Package general
Module measures

Check if measure

Synopsis
is_measure(v)

Arguments

v	in	value to be tested Allowed:any

Returns
Bool

Description
Checks if the operand is a correct measure

1.4.2 measures - Tool

Package general
Module measures

measures tool

Constructors

measures	Construct measures tool

Functions

addxvalue	get some additional measure information
asbaseline	define a baseline from a position measure
baseline	define a baseline measure
cometname	get the current comet name
comettopo	get the current comet table coordinates
comettype	get the current comet table type
direction	define a direction measure
directiongui	GUI for manipulating direction measures
dirshow	Format a direction using globally set formats
doframe	save a measure as frame reference
done	free resources used by tool
doppler	define a doppler measure
dopplergui	GUI for manipulating doppler measures
doshowauto	save an automatic conversion reference
earthmagnetic	define an earthmagnetic measure
epoch	define an epoch measure
epochgui	GUI for manipulating epoch measures
expand	expand n positions to $n*(n-1)/2$ baselines
frameauto	update the currently active frame time automatically
framecomet	set the current comet table
framenoauto	stop an active automatic frame
framenow	set the active frame time at now
frequency	define a frequency measure
frequencygui	GUI for manipulating frequency measures
getoffset	get the offset of a measure
getref	get the reference code of a measure
gettype	get the type of a measure
getvalue	get the value of a measure
gui	Use the measures GUI interface
id	identification of tool
linelist	get a list of known spectral lines
listcodes	get known reference codes
measure	convert a measure to another reference

observatory	get position of an observatory
obslist	get a list of known observatories
posangle	get position angle of two directions
position	define a position measure
positiongui	GUI for manipulating position measures
radialvelocity	define a radialvelocity measure
radialvelocitygui	GUI for manipulating radial velocity measures
rise	get rise and set sidereal time
riseset	get rise and set times
separation	get separation angle between two directions
show	Format a measure using globally set formats
showauto	automatically update current conversions
showframe	show the currently active frame reference
source	get direction of a source
sourcelist	get a list of known sources
spectralline	get frequency of a spectral line
todoppler	convert a frequency or radialvelocity measure to a doppler measure
tofrequency	convert a doppler type value to a frequency
toradialvelocity	convert a doppler type value to a real radial velocity
torestfrequency	convert a frequency and doppler measure to a rest frequency
touvw	calculate a uvw measure from a baseline
type	type of tool
uvw	define a uvw measure

measures - Constructor

Package	**general**
Module	**measures**
Tool	**measures**

Construct measures tool

Synopsis

measures(host, forceneweserver)

Arguments

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

Returns
tool

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

---

measures.gui - Function

Package	**general**
Module	**measures**
Tool	**measures**

Use the measures GUI interface

Synopsis
gui(parent)

Arguments

parent	in	parent window Allowed:window or bool Default: F

Returns
Bool

Description
Start a GUI interface for the handling of measures.
A menu bar will appear with 3 sets of menu headings.
The File menu contains options to close the current window or all measures windows (start again with dm.gui()).
The Format menu can set the available global display formats (and other information like elevation limit for rise/set). Note that the formats can be set at many places throughout the display.
The Frame menu can be used to set frame information (other possibilities are the 'Frame it' buttons on the individual tools). One special option of the Frame menu is the When—Auto option. Selecting this will update the frame time and all the last presses of the 'Convert' button (see below) at specified intervals (see Format—Auto-interval) (to track e.g. the Azimuth of Venus).

These three menu buttons will appear on all major GUI windows.
The next set, only available on the main bar, are Tool, which starts GUI's for individual measure handling.
The Application menu will give you time conversions (LAST, local, sidereal, solar (what is the approximate civil time at another place...)), and clocks of the same; rise/set times etc.
The last menu is the Help menu, which can drive to different parts of the Reference manual.
Selecting a measures type from the Tool menu will bring up a separate GUI to manipulate the selected measure (e.g. a direction).
The overall layout of all measure tools is identical. A top bar with File, Format, Frame and Help menus as described above.
Below that is an Input part (left-side) and an Output part. On the input side you can either type the relevant values for your measure in the top part, and select the correct reference code from the menu next to the Convert button; or use the special menus provided in some cases on the middle bar (Source, Planet, OBS, etc).
The bottom half of the display is reserved to display the input values in (measure dependent) different formats for the same entity (e.g. wavelength, energy for a frequency measure). The bottom half is filled in if a carriage return (or Tab) is entered in an entry field, or one of the action buttons in the middle bar is used.
The middle bar also contains a 'Frame it' button. This will transfer the input to the global environment frame, and an Offset button.
The Offset button has a 'set' menu item. If an offset is set (it is set to the input value, including the reference code), any subsequent entries in the input field are interpreted as having this offset (actuation is indicated by turning the button text red). E.g. if in the epoch GUI you enter 'today' (or 'now'), and press 'Offset—set', the button will turn red, and the bottom field will show:

```
Fri-1998/03/13/01:17:53.991
50885/01:17:53.991
50885.0541 d
```

If you now enter '1h' and hit return, the bottom half will show:

```
Fri-1998/03/13/02:17:53.991
50885/02:17:53.991
50885.0958 d
```

(check what it means by holding MB3 down). Note that in the direction GUI the offset is maybe contrary to what you expect. Internally directions are direction cosines. Hence offsets are direction cosines as well: a vector with a length of 1, and not offsets in longitude and latitude.
If you now hit 'Frame it', a Frame display will appear showing the last entered time.
The 'Convert' button will convert the input with the selected reference code to the reference code selected on the right-hand side of the display. You can

select any output reference code you want from the menu given (although in some cases additional frame information is needed, e.g. the position if you want to convert an UTC to a sidereal time: set it with the Frame menu on the top bar). The top part of the Result window shows the result of the conversion operator in the globally selected format. If you want to see it in another format, in most cases menu buttons next to the displayed result will let you select another format for the display (epoch is the only exception: only displayed in hms). The bottom half of the display shows the result in a variety of fixed formats. Selecting at this stage 'TAI' as output code, and pressing 'Convert' will show on all fields (including top line):

```
02:18:24.991
Fri-1998/03/13/02:18:24.991
50885/02:18:24.991
50885.0961 d
```

The middle bar of the Result contains an Offset button. If 'Offset—set' pressed, the INPUT value will be used as offset to display the result. E.g. following the example and setting Offset, if we now type 2h1m in the Input entry and press 'Convert', the result will be:

```
01:00:60.000        -- rounding on my machine
Wed-1858/11/17/01:00:60.000
0/01:00:60.000
0.0423611111 d
```

Note that the result is 2h1m, and not the 30 odd seconds difference you would maybe expect. The offset was given as the time in UTC, but the result is in TAI: the internal mechanism will convert the Offset to the same reference frame(IAT) before applying it, resulting again in a time difference of 2h1m with the offset specified (Note that the bottom conversion of the date is, of course, nonsense).
The Copy button will export the value in the Result window (as a measure) to the clipboard. On the command line you could then use:

```
- dcb.paste()
[type=epoch, refer=TAI, m0=[value=0.0423611111, unit=d]]
- dq.time(dm.getvalue(dcb.paste())[1])
01:00:60.000
```

Some result bars have also an Info button. It gives a range of selections of additional information that can be displayed. In addition to some fixed ones (e.g. radial velocity in the frequency frame) it also can have a 'User..' one. This allows you to select your own action on the value in the result frame. You have to make a function. The argument of that function is a measure, the result should be a quantity. Let us take an example with the direction frame. We make a function that will take the tan of the right ascension, and multiply it with the HI line frequency (hence the result is a frequency):

```
- myact := function(cr) {
+ return dq.mul(dq.constants('HI'), dq.tan(dm.getvalue(cr)[1]));
+ }
```

On the direction tool GUI we select the SUN as Planet, press Convert. I got (and that depends on the time frame of course, -7 and -3 degrees). Press 'Info—User..', and select 'freq' and type 'myact' in the window that appears, and hit OK. The result is: -175.400989 MHz (play with the button on the right of this to get: -170.91834 cm, -7.25400475e-10 keV, -5.53908615e-17 pc). A way of starting automatically a GUI interface is by adding the following to your .aipsrc file

```
measures.default: gui
```

---

measures.dirshow - Function

Package	**general**
Module	**measures**
Tool	**measures**

Format a direction using globally set formats

Synopsis
dirshow(v)

Arguments

v	in	value to be converted to string Allowed:a direction measure

Returns
string

Description
dirshow will convert a direction to a string, using the formats set globally for longitude and latitude (see setformat).

Example

```
- dm.dirshow(dm.direction('venus'))
00:00:00.000 +090.00.00.000 venus
```

---

measures.show - Function

Package	**general**
Module	**measures**
Tool	**measures**

Format a measure using globally set formats

Synopsis
show(v, refcode)

Arguments

v	in	value to be converted to string Allowed:any measure
refcode	in	add the reference code to output Allowed:boolean Default: T

Returns
string

Description
show will convert a measure to a string, using the formats set globally for various types of variables (see setformat). All measures are catered for (at this moment *direction, position, epoch, radialvelocity, frequency, doppler, baseline, uvw, earthmagnetic*).

Example

```
- dm.show(dm.frequency('lsrk', dq.constants('HI')))
1420.40575 MHz LSRK
- dq.setformat('freq', 'keV')
T
- dm.show(dm.frequency('lsrk', dq.constants('HI')))
```

```
5.87432838e-09 keV LSRK
- dm.show(dm.frequency('lsrk', dq.constants('HI')), refcode=F)
5.87432838e-09 keV
```

---

measures.epochgui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating epoch measures

Synopsis
epochgui()

Arguments

Returns
Bool

Description
epochgui starts a GUI for manipulating astronomical epochs. It can be started from the command line, or from the main GUI interface (see gui function), which also describes the layout and use of the different fields in the GUI. Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

position LAST, LMST

---

measures.epoch - Function

Package	**general**
Module	**measures**
Tool	**measures**

define an epoch measure

Synopsis
epoch(rf, v0, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	epoch value Allowed:time quantity Default: 0.0d
off	in	optional offset Allowed:epoch measure Default: F

Returns
epoch measure

Description
epoch defines an epoch measure from the CLI. It has to specify a reference code, an epoch quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be an epoch. Allowable reference codes are: *UTC TAI LAST LMST GMST1 GAST UT1 UT2 TDT TCG TDB TCB*.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.epoch())
[normal=LAST LMST GMST1 GAST UT1 UT2 UTC TAI TDT TCG TDB TCB IAT
        GMST TT ET UT, extra=]
```

See quantity for possible time formats.

Example

```
- dm.epoch('utc','today')
[type=epoch, refer=utc, m0=[value=50451, unit=d]]
```

---

measures.directiongui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating direction measures

Synopsis
directiongui()

Arguments

Returns
Bool

Description
directiongui starts a GUI for manipulating astronomical directions. It can be started from the command line, or from the main GUI interface (see gui function), which also describes the layout and use of the GUI.
Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

epoch all but J2000, B1950, GALACTIC, SUPERGAL, ECLIPTIC

position HADEC, AZEL

---

measures.direction - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a direction measure

Synopsis
direction(rf, v0, v1, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	longitude Allowed:quantity Default: 0..
v1	in	latitude Allowed:quantity Default: 90..
off	in	optional offset Allowed:direction measure Default: F

Returns
direction measure

Description
direction defines a direction measure from the CLI. It has to specify a reference code, direction quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a direction. Allowable reference codes are: *J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL SUPERGAL ECLIPTIC MECLIPTIC TECLIPTIC MERCURY VENUS MARS JUPITER SATURN URANUS NEPTUNE PLUTO MOON SUN COMET*.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.direction())
[normal=J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
        AZELSW AZELNE AZELGEO AZELSWGEO AZELNEGEO JNAT ECLIPTIC
        MECLIPTIC TECLIPTIC SUPERGAL ITRF TOPO,
 extra=MERCURY VENUS MARS JUPITER SATURN URANUS NEPTUNE PLUTO SUN
       MOON COMET]
```

The direction quantity values should be longitude(angle) and latitude(angle) (none needed for planets: the frame epoch defines coordinates). See quantity for possible angle formats.

Example

```
- dm.direction('j2000','30deg','40deg')
[type=direction, refer=j2000, m0=[value=0.523599, unit=rad],
m1=[value=0.698132, unit=rad]]
- dm.direction('mars')
```

```
[type=direction, refer=mars, m0=[value=0, unit=rad],
m1=[value=1.57079633, unit=rad]]
```

---

measures.getvalue - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the value of a measure

Synopsis
getvalue(v)

Arguments

v	in	measure Allowed:array of measures

Returns
array of quantities

Description
getvalue gets the actual implementation value of the measure.

Example

```
- b:=dm.direction('j2000','0deg','80deg');
- dm.getvalue(b)
[*55=[unit=rad, value=0], *56=[value=1.3962634, unit=rad]]
- dm.getvalue(b)::
[id=quant, shape=[2] ]
```

---

measures.gettype - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the type of a measure

Synopsis
gettype(v)

Arguments

v	in	measure Allowed:array of measures

Returns
string

Description
gettype gets the actual type of the measure.

Example

```
- b:=dm.direction('j2000','0deg','80deg');
- dm.getvalue(b)
[*55=[unit=rad, value=0], *56=[value=1.3962634, unit=rad]]
- dm.getvalue(b)::
[id=quant, shape=[2] ]
- dm.gettype(b)
direction
```

---

measures.getref - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the reference code of a measure

Synopsis
getref(v)

Arguments

v	in	measure Allowed:array of measures

Returns
string

Description
gettype gets the actual reference code of the measure.

Example

```
- b:=dm.direction('j2000','0deg','80deg');
- dm.getvalue(b)
[*55=[unit=rad, value=0], *56=[value=1.3962634, unit=rad]]
- dm.getvalue(b)::
[id=quant, shape=[2] ]
- dm.gettype(b)
direction
- dm.getref(b)
J2000
-
```

---

measures.getoffset - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the offset of a measure

Synopsis
getoffset(v)

Arguments

v	in	measure Allowed:array of measures

Returns
measure or boolean

Description
getoff gets the actual offset of the measure (as a measure) or F if no offset given.

Example

```
- b:=dm.direction('j2000','0deg','80deg');
- dm.getvalue(b)
[*55=[unit=rad, value=0], *56=[value=1.3962634, unit=rad]]
- dm.getvalue(b)::
[id=quant, shape=[2] ]
- dm.gettype(b)
direction
- dm.getref(b)
J2000
- dm.getoffset(b)
F
```

---

measures.cometname - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the current comet name

Synopsis
cometname()

Arguments

Returns
string or fail

Description
cometname gets the name of the current comet (if any).

Example

```
- dm.cometname()
SEVERE: Method cometname fails!
No Comet table present
<fail>: Method cometname fails!
No Comet table present
        File:   servers.g, Line 1004
        Stack:  .()
                .()
```

---

measures.comettype - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the current comet table type

Synopsis
comettype()

Arguments

Returns
string

Description
comettype gets the comet table type (apparent or topocentric)

Example

```
- dm.comettype()
none
```

---

measures.comettopo - Function

Package	**general**
Module	**measures**
Tool	**measures**

get the current comet table coordinates

Synopsis
comettopo()

Arguments

Returns
position measure or fail

Description
comettopo gets the comet table's topographic coordinates used.

Example

```
- dm.comettopo()
SEVERE: Method comettopo fails!
No Topocentric Comet table present
<fail>: Method comettopo fails!
No Topocentric Comet table present
        File:   servers.g, Line 1004
        Stack:  .()
                .()
                .()
```

---

measures.framecomet - Function

Package	**general**
Module	**measures**
Tool	**measures**

set the current comet table

Synopsis
framecomet(v)

Arguments

v	in	name of a table Allowed:table name Default: ”

Description
framecomet will put the specified comet table in the frame.

Example

```
- dm.framecomet('VGEO')
T
- dm.showframe()
position: 09:58:12.033 -030.08.43.736 6372960.26 m ITRF
epoch: 2002/10/18/00:15:49 UTC
direction: 00:00:00.000 +090.00.00.000 VENUS
T
- dm.cometname()
VENUS
- dm.comettype()
APP
- dm.doframe(dm.epoch('et', dq.quantity('1997/12/20/17:30:0')))
T
- dm.measure(dm.direction('comet'), 'app')
[type=direction, refer=APP, m1=[value=-0.347102565, unit=rad],
  m0=[unit=rad, value=-0.949364859]]
```

Returns
bool

measures.positiongui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating position measures

Synopsis
positiongui()

Arguments

Returns
Bool

Description
positiongui starts a GUI for manipulating geographical positions. It can be started from the command line, or from the main GUI interface (see gui function), which also describes the basic use of the GUI.
Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

none none needed

---

measures.position - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a position measure

Synopsis
position(rf, v0, v1, v2, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	longitude or x Allowed:quantity Default: 0..
v1	in	latitude or y Allowed:quantity Default: 90..
v2	in	height or z Allowed:quantity Default: 0m
off	in	optional offset Allowed:position measure Default: F

Returns
string

Description
position defines a position measure from the CLI. It has to specify a reference code, position quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a position. Allowable reference codes are: *WGS84 ITRF* (World Geodetic System and International Terrestrial Reference Frame).
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.position())
[normal=ITRF WGS84, extra=]
```

The position quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Example

```
- dm.position('wgs84','30deg','40deg','10m')
[type=position, refer=wgs84, m2=[value=10, unit=m],
m0=[value=0.523599, unit=rad], m1=[value=0.698132, unit=rad]]
- dm.observatory('ATCA')
[type=position, refer=ITRF, m2=[value=6.37296e+06, unit=m],
m0=[value=2.61014, unit=rad], m1=[value=-0.526138, unit=rad]]
```

---

measures.observatory - Function

Package	**general**
Module	**measures**
Tool	**measures**

get position of an observatory

Synopsis
observatory(name)

Arguments

name	in	observatory name Allowed:coded string - case insensitive Default: 'ATCA'

Returns
position measure

Description
observatory will give you the position of an observatory as given in the system. At the time of writing the following observatories are recognised (but check e.g. the position GUI for currently known ones, or the dm.obslist() tool function): *ALMA ATCA BIMA CLRO DRAO DWL GB JCMT MOPRA NRAO12M PKS VLA WSRT*.

Example

```
- dm.observatory('ATCA')
[type=position, refer=ITRF, m2=[value=6.37296e+06, unit=m],
m0=[value=2.61014, unit=rad], m1=[value=-0.526138, unit=rad]]
```

---

measures.obslist - Function

Package	**general**
Module	**measures**
Tool	**measures**

get a list of known observatories

Synopsis
obslist()

Returns
position measure

Description
obslist will give you a string with the space separated list of observatories known in the Observatories table (see measuresdata module).

Example

```
- dm.obslist()
ALMA ATCA BIMA CLRO DRAO DWL GB JCMT MOPRA NRAO12M PKS VLA WSRT
```

---

measures.linelist - Function

Package	**general**
Module	**measures**
Tool	**measures**

get a list of known spectral lines

Synopsis
linelist()

Returns
string

Description
linelist will give you a string with the space separated list of spectral lines known in the Lines table (see measuresdata module). A number of lines is available now, but tables with many lines are already online, and will be interfaced once a nomenclature can be defined for the tens of thousands of lines.

Example

```
- dm.linelist()
HI
```

---

measures.spectralline - Function

Package	**general**
Module	**measures**
Tool	**measures**

get frequency of a spectral line

Synopsis
spectralline(name)

Arguments

name	in	name Allowed:coded string Default: 'HI'

Returns
quantity

Description
spectralline will give you the frequency of a spectral line. The known list can be obtained by dm.linelist().

Example

```
- dm.spectralline('HI')
[value=1420.40575, unit=MHz]
```

---

measures.sourcelist - Function

Package	**general**
Module	**measures**
Tool	**measures**

get a list of known sources

Synopsis
sourcelist()

Returns
string

Description
sourcelist will give you a string with the space separated list of sources known in the Sources table (see measuresdata module).

Example

```
- dm.sourcelist()
0002-478 0003+380 0003-066 0007+106 0007+171 0008-264 0008-421
......
```

---

measures.source - Function

Package	**general**
Module	**measures**
Tool	**measures**

get direction of a source

Synopsis
source(name)

Arguments

name	in	name Allowed:coded string Default: 'PKS1934-638'

Returns
direction measure

Description
source will give you the direction of a source. The known list can be obtained by dm.sourcelist().

Example

```
- dm.source()
[type=direction, refer=j2000, m0=[value=-1.13700885, unit=rad],
 m1=[value=-1.1119962, unit=rad]]
```

---

measures.frequencygui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating frequency measures

Synopsis
frequencygui()

Arguments

Returns
Bool

Description
frequencygui starts a GUI for manipulating electromagnetic wave characteristics. It can be started from the command line, or from the main GUI interface (see gui function), which also describes the use of the GUI. Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

epoch TOPO, GEO

position TOPO

direction all

radialvelocity REST

---

measures.frequency - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a frequency measure

Synopsis
frequency(rf, v0, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	frequency/wavelength/... Allowed:quantity Default: 0Hz
off	in	optional offset Allowed:frequency measure Default: F

Returns
frequency measure

Description
frequency defines a frequency measure from the CLI. It has to specify a reference code, frequency quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a frequency. Allowable reference codes are: *REST LSRK LSRD BARY GEO TOPO GALACTO.*
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.frequency())
[normal=REST LSRK LSRD BARY GEO TOPO GALACTO, extra=]
```

The frequency quantity values should be in one of the recognised units (examples all give same frequency):

- value with time units: a period (0.5s)
- value as frequency: 2Hz
- value in angular frequency: 720deg/s
- value as length: 149896km
- value as wave number: 4.19169e-8m-1
- value as enery (h.nu): 8.27134e-9ueV
- value as momentum: 4.42044e-42kg.m

Example

```
- dm.frequency('lsrk', '5GHz')
[type=frequency, refer=lsrk, m0=[value=5e+09, unit=Hz]]
- dm.frequency('lsrk', '21cm')
[type=frequency, refer=lsrk, m0=[value=1.42758e+09, unit=Hz]]
```

---

measures.dopplergui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating doppler measures

Synopsis
dopplergui()

Arguments

Returns
Bool

Description
dopplergui starts a GUI for manipulating Doppler shifts in astronomical type units (like 'OPTICAL', 'RADIO'). The functionality is also available in the radialvelocitygui. It can be started from the command line, or from the main GUI interface (see gui function), which also decsribes the use of the GUI. Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

none none needed

---

measures.doppler - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a doppler measure

Synopsis
doppler(rf, v0, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	doppler ratio/velocity Allowed:quantity Default: 0
off	in	optional offset Allowed:doppler measure Default: F

Returns
doppler measure

Description
doppler defines a doppler measure from the CLI. It has to specify a reference code, doppler quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a

doppler. Allowable reference codes are: *RADIO OPTICAL Z RATIO RELATIVISTIC BETA GAMMA*.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.doppler())
[normal=RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC, extra=]
```

The doppler quantity values should be either non-dimensioned to specify a ratio of the light velocity, or in velocity. (examples all give same doppler):

Example

```
- dm.doppler('radio', 0.4)
[type=doppler, refer=radio, m0=[value=1.19917e+08, unit=m/s]]
- dm.doppler('radio',dm.mul(dm.quantity(0.4),dm.constants('c')))
[type=doppler, refer=radio, m0=[unit=m/s, value=1.19917e+08]]
```

---

measures.radialvelocitygui - Function

Package	**general**
Module	**measures**
Tool	**measures**

GUI for manipulating radial velocity measures

Synopsis
radialvelocitygui()

Arguments

Returns
Bool

Description
radialvelocitygui starts a GUI for manipulating astronomical radial velocities. It can be started from the command line, or from the main GUI interface (see gui function), which also describes the basic use of the GUI.

Conversion will sometimes need additional frame information. If either reference code is one of the following, the frame information needed is:

epoch TOPO, GEO

position TOPO

direction all

---

measures.radialvelocity - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a radialvelocity measure

Synopsis
radialvelocity(rf, v0, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	radial velocity Allowed:quantity Default: 0m/s
off	in	optional offset Allowed:radialvelocity measure Default: F

Returns
radialvelocity measure

Description
radialvelocity defines a radialvelocity measure from the CLI. It has to specify a reference code, radialvelocity quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a radialvelocity. Allowable reference codes are: *LSRK LSRD BARY GEO TOPO GALACTO*.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.radialvelocity())
[normal=LSRK LSRD BARY GEO TOPO GALACTO, extra=]
```

The radialvelocity quantity values should be given as velocity.

Example

```
- dm.radialvelocity('lsrk','20km/s')
[type=radialvelocity, refer=lsrk, m0=[value=20000, unit=m/s]]
```

---

measures.uvw - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a uvw measure

Synopsis

uvw(rf, v0, v1, v2, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	longitude or x Allowed:quantity Default: 0..
v1	in	latitude or y Allowed:quantity
v2	in	height or z Allowed:quantity
off	in	optional offset Allowed:uvw measure Default: F

Returns

uvw measure

Description
uvw defines a uvw measure from the CLI. It has to specify a reference code, uvw quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a uvw. Allowable reference codes are ITRF and the direction ones.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.uvw())
[normal=J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
        AZELSW AZELNE AZELGEO AZELSWGEO AZELNEGEO JNAT ECLIPTIC
        MECLIPTIC TECLIPTIC SUPERGAL ITRF TOPO, extra=]
```

The uvw quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Example

```
- dm.uvw('itrf','30deg','40deg','10m')
[type=uvw, refer=ITRF, m2=[value=10, unit=m], m1=[unit=rad,
   value=0.698131701], m0=[unit=rad, value=0.523598776]]
- dm.measure(dm.uvw('itrf','30deg','40deg','10m'), 'j2000')
[type=uvw, refer=J2000, m2=[value=10, unit=m],
    m1=[unit=rad, value=0.698073433],
     m0=[unit=rad, value=-1.74254263]]
```

---

measures.touvw - Function

Package	**general**
Module	**measures**
Tool	**measures**

calculate a uvw measure from a baseline

Synopsis
touvw(v, dot, xyz)

Arguments

v	in	baseline measure Allowed:baseline measure
dot	out	uvw-dot Allowed:quantity array Default: unset
xyz	out	uvw Allowed:quantity array

Returns
uvw measure

Description
touvw calculates a uvw measure from a baseline. The baseline can consist of a vector of actual baseline positions. Note that the baseline does not have to be a proper *baseline*, but can be a series of positions (to call positions baselines see asbaseline) for speed reasons: operations are linear and can be done on positions, which are converted to baseline values at the end (with expand).
Whatever the reference code of the baseline, the returned *uvw* will be given in J2000. If the *dot* argument is given, that variable will be filled with a quantity array consisting of the time derivative of the uvw (note that only the sidereal rate is taken into account; not precession, earth tides and similar variations, which are much smaller). If the *xyz* variable is given, it will be filled with the quantity values of the uvw measure.
The values of the input baselines can be given as either a quantity vector per x, y or z value; or as an array as explained in the introduction.
uvw coordinates are calculated for a certain direction in the sky; hence the frame has to contain the direction for the calculation to work. Since the baseline and the sky rotate with respect of each other, the time should be specified as well.

Example

```
- dm.doframe(dm.observatory('atca'))
- dm.doframe(dm.source('pks1934-638'))
- dm.doframe(dm.epoch('utc',dq.unit('today')))
- b := dm.baseline('itrf', '10m', '20m', '30m')
- dm.touvw(b)
[type=uvw, refer=J2000, m2=[value=37.4165739, unit=m],
    m1=[unit=rad, value=-0.751775754], m0=[unit=rad,
    value=2.51551809]]
- dm.touvw(b,d,x); print d; print x;
[type=uvw, refer=J2000, m2=[value=37.4165739, unit=m],
```

```
    m1=[unit=rad, value=-0.751775754], m0=[unit=rad,
    value=2.51551809]]
[value=[[1:3,]
    0.0002206593
    0.00144013975
    0.000711365081], unit=m/s]
[value=[[1:3,]
    -22.1480126
    16.0156625
    -25.5531621], unit=m]
- dm.getvalue(dm.touvw(b))
[*7=[unit=rad, value=2.51551809], *8=[unit=rad, value=-0.751775754], *9=[value=37.4165739, unit=m]]
- dm.getvalue(dm.touvw(b))[1]
[unit=rad, value=2.51551809]
```

An example with more than one value:

```
- sb := dm.baseline('itrf',dq.unit([10,50],'m'),dq.unit([20,100],'m'),
dq.unit([30,150],'m'))
- dm.touvw(sb,d,x); print d; print x;
[type=uvw, refer=J2000, m2=[value=[37.4165739 187.082869] , unit=m],
    m1=[unit=rad, value=[-0.743811234 -0.743811234] ],
    m0=[unit=rad, value=[2.50094148 2.50094148] ]]
[value=[[1:3,]
    0.00025643414  0.0012821707
    0.00143537137  0.00717685684
    0.000709009712 0.00354504856], unit=m/s]
[value=[[1:3,]
    -22.0746793 -110.373397
    16.45792    82.2895998
    -25.334668  -126.67334], unit=m]
- dm.getvalue(dm.touvw(sb))
[*7=[unit=rad, value=[2.50094148 2.50094148] ],
        *8=[unit=rad, value=[-0.743811234 -0.743811234] ],
 *9=[value=[37.4165739 187.082869] , unit=m]]
- dm.getvalue(dm.touvw(sb))[1]
[unit=rad, value=[2.50094148 2.50094148] ]
- dq.getvalue(dm.getvalue(dm.touvw(sb))[1])[2]
2.50094148
- dm.doframe(dm.epoch('utc','today'))
T
- dm.expand(sb)
[type=baseline, refer=ITRF, m2=[value=149.666295, unit=m],
 m1=[unit=rad, value=0.930274014],
 m0=[unit=rad, value=1.10714872]]
- dm.expand(sb,x)
[type=baseline, refer=ITRF, m2=[value=149.666295, unit=m],
```

```
m1=[unit=rad, value=0.930274014],
m0=[unit=rad, value=1.10714872]]
- x
[value=[[1:3,]
    40
    80
    120], unit=m]
- dm.expand(dm.touvw(sb),x); x
[type=uvw, refer=J2000, m2=[value=149.666295, unit=m],
   m1=[unit=rad, value=-0.654614537],
   m0=[unit=rad, value=2.32532487]]
[value=[[1:3,]
    -81.3219596
    86.5043397
    -91.124849], unit=m]
- dm.touvw(dm.expand(sb),xyz=x); x
[type=uvw, refer=J2000, m2=[value=149.666295, unit=m],
   m1=[unit=rad, value=-0.654614537],
   m0=[unit=rad, value=2.32532487]]
[value=[[1:3,]
    -81.3219596
    86.5043397
    -91.124849], unit=m]
- a := dm.touvw(sb, xyz=x); x
[value=[[1:3,]
    -20.3304899 -101.652449
    21.6260849  108.130425
    -22.7812122 -113.906061], unit=m]
-
```

---

measures.expand - Function

Package	**general**
Module	**measures**
Tool	**measures**

expand n positions to $n*(n-1)/2$ baselines

Synopsis
expand(v, xyz)

Arguments

v	in	measure Allowed:baseline, position or uvw measure
xyz	out	uvw Allowed:quantity array

Returns
measure

Description
expand calculates the differences between a series of given measure values: it calculates baseline values from position values. The returned value is a measure, but the value of the optional output variable *xyz* will be set to an array of values.

Example

```
- b := dm.baseline('itrf', '10m', '20m', '30m')
- sb := dm.baseline('itrf',dq.unit([10,50],'m'),dq.unit([20,100],'m'),
+    dq.unit([30,150],'m'))
- dm.expand(b,x); x
[type=baseline, refer=ITRF, m2=[value=37.4165739, unit=m],
m1=[unit=rad, value=0.930274014],
m0=[unit=rad, value=1.10714872]]
[value=[[1:3,]
    10
    20
    30], unit=m]
- dm.expand(sb,x); x
[type=baseline, refer=ITRF, m2=[value=149.666295, unit=m],
m1=[unit=rad, value=0.930274014],
m0=[unit=rad, value=1.10714872]]
[value=[[1:3,]
    40
    80
    120], unit=m]
-
```

---

measures.earthmagnetic - Function

Package	**general**
Module	**measures**
Tool	**measures**

define an earthmagnetic measure

Synopsis
earthmagnetic(rf, v0, v1, v2, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	longitude or x Allowed:quantity Default: 0G
v1	in	latitude or y Allowed:quantity Default: 0..
v2	in	height or z Allowed:quantity Default: 90..
off	in	optional offset Allowed:earthmagnetic measure Default: F

Returns
earthmagnetic measure

Description
earthmagnetic defines an earthmagnetic measure from the CLI. It needs a reference code, earthmagnetic quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given) if the reference code is not for a model, and optionally it can specify an offset, which in itself has to be a earthmagnetic. In general you specify a model (IGRF is the default and the only one known) and convert it to an explicit field. (See

`http://fdd.gsfc.nasa.gov/IGRF.html`

for information on the International Geomagnetic Reference Field). The earthmagnetic quantity values should be either longitude (angle), latitude(angle) and length(field strength); or x,y,z (field). See quantity for possible angle formats.

Example

```
- dm.earthmagnetic('igrf')
[type=earthmagnetic, refer=IGRF, m2=[value=1e-06, unit=nT],
      m1=[unit=nT, value=0], m0=[unit=nT, value=6.123234e-23]]
- dm.measure(dm.earthmagnetic('igrf'), 'j2000')
[type=earthmagnetic, refer=J2000, m2=[value=47232.741, unit=nT],
      m1=[unit=nT, value=-14627.3775],
       m0=[unit=nT, value=25892.4992]]
```

---

measures.baseline - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a baseline measure

Synopsis
baseline(rf, v0, v1, v2, off)

Arguments

rf	in	reference code Allowed:coded string
v0	in	longitude or x Allowed:quantity Default: 0..
v1	in	latitude or y Allowed:quantity
v2	in	height or z Allowed:quantity
off	in	optional offset Allowed:baseline measure Default: F

Returns
baseline measure

Description
baseline defines a baseline measure from the CLI. It has to specify a reference code, baseline quantity values (see introduction for the action on a scalar

quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a baseline. Allowable reference codes are ITRF and the direction ones.
Note that additional ones may become available. Check in Glish with:

```
- dm.listcodes(dm.baseline())
[normal=J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
        AZELSW AZELNE AZELGEO AZELSWGEO AZELNEGEO JNAT ECLIPTIC
        MECLIPTIC TECLIPTIC SUPERGAL ITRF TOPO, extra=]
```

The baseline quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Example

```
- dm.baseline('itrf','30deg','40deg','10m')
[type=baseline, refer=ITRF, m2=[value=10, unit=m],
 m1=[unit=rad, value=0.698131701], m0=[unit=rad, value=0.523598776]]
- dm.measure(dm.baseline('itrf','30deg','40deg','10m'), 'j2000')
[type=baseline, refer=J2000, m2=[value=10, unit=m],
 m1=[unit=rad, value=0.698073433],
  m0=[unit=rad, value=-1.74254263]]
```

---

measures.asbaseline - Function

Package	**general**
Module	**measures**
Tool	**measures**

define a baseline from a position measure

Synopsis

asbaseline(pos)

Arguments

pos	in	position measure Allowed:position measure

Returns

baseline measure

Description
asbaseline converts a position measure into a baseline measure. No actual baseline is calculated, since operations can be done on positions, with subtractions to obtain baselines at a later stage.

Example

```
- b := dm.position('itrf', '10m', '20m', '30m');
- sb := dm.position('itrf',dq.unit([10,50],'m'),dq.unit([20,100],'m'),
+ dq.unit([30,150],'m'));
- print b; print sb;
[type=position, refer=ITRF, m2=[value=37.4165739, unit=m],
 m1=[unit=rad, value=0.930274014],
 m0=[unit=rad, value=1.10714872]]
[type=position, refer=ITRF, m2=[value=[37.4165739 187.082869] , unit=m],
 m1=[unit=rad, value=[0.930274014 0.930274014] ],
 m0=[unit=rad, value=[1.10714872 1.10714872] ]]
- print b; print dm.asbaseline(b); print sb; print dm.asbaseline(sb)
[type=position, refer=ITRF, m2=[value=37.4165739, unit=m],
m1=[unit=rad, value=0.930274014],
m0=[unit=rad, value=1.10714872]]
[type=baseline, refer=ITRF, m2=[value=37.4165739, unit=m],
m1=[unit=rad, value=0.930274014],
m0=[unit=rad, value=1.10714872]]
[type=position, refer=ITRF, m2=[value=[37.4165739 187.082869] , unit=m],
 m1=[unit=rad, value=[0.930274014 0.930274014] ],
 m0=[unit=rad, value=[1.10714872 1.10714872] ]]
[type=baseline, refer=ITRF, m2=[value=[37.4165739 187.082869] , unit=m],
 m1=[unit=rad, value=[0.930274014 0.930274014] ],
 m0=[unit=rad, value=[1.10714872 1.10714872] ]]
```

---

measures.listcodes - Function

Package	**general**
Module	**measures**
Tool	**measures**

get known reference codes

Synopsis
listcodes(ms)

Arguments

ms	in	the measure type for which to list Allowed:a measure

Returns
record with two string vectors

Description
listcodes will produce the known reference codes for a specified measure type. It will return a record with two entries. The first is a string vector of all normal codes; the second a string vector (maybe empty) with all extra codes (like planets).

Example

```
# Generate some direction
# Note that an empty or non-specified reference code will produce the
# measure with the default code for that measure type
- a:=dm.direction()
- dm.getref(a)
J2000
- is_measure(a)
T
# Get the known reference codes for direction
- dm.listcodes(a)
[normal=J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL AZELSW
AZELNE JNAT ECLIPTIC MECLIPTIC TECLIPTIC SUPERGAL ITRF TOPO,
extra=MERCURY VENUS MARS JUPITER SATURN URANUS NEPTUNE
       PLUTO SUN MOON COMET]
```

---

measures.measure - Function

Package	**general**
Module	**measures**
Tool	**measures**

convert a measure to another reference

Synopsis

measure(v, rf, off, qv)

Arguments

v	in	measure to be converted Allowed:measure
rf	in	output reference code Allowed:coded string
off	in	optional output offset Allowed:measure Default: F
qv	in	optional quantity (with multiple values possible) Allowed:quaantity Default: F

Returns

measure

Description

measure converts measures (epoch, direction etc.) from one reference to another. It will, for instance, convert a direction from J2000 to AZEL representation.
Its arguments are a measure, an output reference code (see the individual measures for the allowable codes (direction, position, epoch, frequency, doppler, radialvelocity, baseline, uvw, earthmagnetic)), and an optional offset of the same type as the main measure. The offset will be subtracted from the result before it is returned.
In some cases (see the individual measures for when), more information than just a reference code is necessary. E.g. the above example of a conversion to AZEL, needs to know for when, and where on Earth we want it. This information is stored in a reference frame. Measures are set in the reference frame with the doframe function. The frame is tool wide.
An optional argument em qv can contain a quantity with a vector of values and a unit. When present, the vector will be converted (and returned) in the same way as the main reference. Not all Measures accept this argument yet. If needed please ask an enhancement.

Example

```
- print a := dm.epoch('utc', 'today')  # a time
[type=epoch, refer=utc, m0=[value=50451.2, unit=d]]
```

```
- dm.measure(a, 'tai') # convert to IAT
[type=epoch, refer=tai, m0=[value=50451.2, unit=d]]
- dm.doframe(a) # set time in frame
T
- dm.doframe(dm.observatory('ATCA')) # set position in frame
T
- print b:=dm.direction('j2000', dm.toangle('0h'), '-30deg')  # a direction
[type=direction, refer=j2000, m0=[value=0, unit=rad],
m1=[value=-0.523599, unit=rad]]
- dm.measure(b, 'azel') # convert to AZEL
[type=direction, refer=azel, m0=[value=-1.35242, unit=rad],
m1=[value=0.846695, unit=rad]]
- print dm.angle(dm.getvalue(dm.measure(b, 'azel'))[1]),
+  dm.angle(dm.getvalue(dm.measure(b, 'azel'))[2]) # show as angles
-077.29.18 +048.30.43
```

In the following the *qv* argument is used.

```
# Fill the frame with necessary information
- dm.doframe(dm.epoch('utc','today'))
T
- dm.doframe(dm.observatory('atca'))
T
- dm.doframe(dm.direction('mars'))
T
# Make a list of frequencies to be converted
- a:=dq.unit([1,1.1,1.2,1.3],'GHz')
- a
[value=[1 1.1 1.2 1.3] , unit=GHz]
# Make a frequency measure. Although any can be used, it is advisable
# to use an element of the list, to make sure all units are correct
- m:=dm.frequency('lsrk',dq.unit(dq.getvalue(a)[1], dq.getunit(a)))
# Convert all
- dm.measure(m,'lsrd',qv=a)
[type=frequency, refer=LSRD, m0=[value=999995118, unit=Hz]]
# And check
- a
[value=[0.999995118 1.09999463 1.19999414 1.29999365] , unit=GHz]
```

---

measures.doframe - Function

Package	**general**
Module	**measures**
Tool	**measures**

save a measure as frame reference

Synopsis
doframe(v)

Arguments

v	in	measure to be set in frame Allowed:measure

Returns
bool

Description
doframe will set the measure specified as part of a frame. The *Frame it* button on the different measure GUI's has the same effect.
If conversion from one type to another is necessary (either in the GUI with the *Convert*$->$ button or with the measure function), the following frames should be set if one of the reference types involved in the conversion is as in the following lists.
Epoch

```
UTC
TAI
LAST position
LMST  position
GMST1
GAST
UT1
UT2
TDT
TCG
TDB
TCD
```

Direction

```
J2000
JMEAN epoch
JTRUE  epoch
APP  epoch
B1950
BMEAN  epoch
```

```
BTRUE  epoch
GALACTIC
HADEC  epoch position
AZEL epoch  position
SUPERGALACTIC
ECLIPTIC
MECLIPTIC      epoch
TECLIPTIC epoch
PLANET epoch   [position]
```

Position

```
WGS84
ITRF
```

Radial Velocity

```
LSRK  direction
LSRD  direction
BARY  direction
GEO  direction epoch
TOPO  direction epoch position
GALACTO direction
```

Doppler

```
RADIO
OPTICAL
Z
RATIO
RELATIVISTIC
BETA
GAMMA
```

Frequency

```
REST  direction radialvelocity
LSRK  direction
LSRD  direction
BARY  direction
GEO  direction epoch
TOPO  direction epoch position
GALACTO
```

Example

```
- print a := dm.epoch('utc', 'today')  # a time
[type=epoch, refer=utc, m0=[value=50451.2, unit=d]]
- dm.doframe(a) # set time in frame
T
```

---

measures.framenow - Function

Package	**general**
Module	**measures**
Tool	**measures**

set the active frame time at now

Synopsis
framenow()

Arguments

Returns
bool

Description
framenow will fill the active frame time with the current date and time. The different frame values necessary are described in the doframe function

Example

```
- dm.framenow() # specify now as frame reference
T
- dm.showframe(F) # and show the current frame
epoch: 2002/10/18/00:15:49 UTC
T
```

---

measures.showframe - Function

Package	**general**
Module	**measures**
Tool	**measures**

show the currently active frame reference

Synopsis
showframe(g)

Arguments

g	in	use gui if available Allowed:boolean Default: T

Returns
bool

Description
showframe will display the currently active reference frame values in either a separate screen (if working in GUI environment), or on the terminal if no GUI active, or a forcing argument F is given. The different frame values necessary are described in the doframe function. The frame is displayed on the terminal using the formatting as done for the show function.

Example

```
- dm.doframe(dm.epoch('utc','today')) # specify now as frame reference
T
- dm.showframe(F) # and show the current frame
epoch: 2002/10/18/00:15:49 UTC
T
```

---

measures.frameauto - Function

Package	**general**
Module	**measures**
Tool	**measures**

update the currently active frame time automatically

Synopsis

frameauto(v)

Arguments

v	in	update interval Allowed:time string Default: 2s

Returns
bool

Description
frameauto will set the active frame to now, and update it regularly with the specified amount (default is 5s). It will also show the updated conversion value of any conversion you have done using a *Convert* button or the doshowauto tool function. The different frame values necessary are described in the doframe function.

Example

```
- dm.frameauto('2s') # specify the frame reference to be updated every 2s
T
- dm.showframe(F) # and show the current frame
epoch: 2002/10/18/00:15:49 UTC
T
```

---

measures.framenoauto - Function

Package	**general**
Module	**measures**
Tool	**measures**

stop an active automatic frame

Synopsis
framenoauto()

Arguments

Returns
bool

Description
framenoauto will stop the update of an active automatic frame update.

Example

```
- dm.frameauto() # specify frame reference updated every 5s
T
- dm.framenoauto() # stop update
T
- dm.showframe(F) # and show the current frame
epoch: 2002/10/18/00:15:49 UTC
T
```

---

measures.doshowauto - Function

Package	**general**
Module	**measures**
Tool	**measures**

save an automatic conversion reference

Synopsis
doshowauto(v, r, of)

Arguments

v	in	measure to be auto converted Allowed:measure
r	in	reference code to be autoconverted to Allowed:valid reference code string
of	in	possible output offset Allowed:measure Default: F

Returns
bool

Description
doshowauto will set the conversion specified as part of the automatic update display. Normally this is done automatically by the *Convert*$->$ button. Note that in the case of an Epoch the time to be converted is taken from the automatic frame, rather than from the definition.

Example

```
- dm.doshowauto(dm.direction('venus'), 'j2000') # auto update Venus position
T
- dm.showauto(F) # and show it on terminal
[type=direction, refer=j2000, m0=[value=-1.00354548, unit=rad],
 m1=[value=-0.383760227, unit=rad]]
T
```

---

measures.showauto - Function

Package	**general**
Module	**measures**
Tool	**measures**

automatically update current conversions

Synopsis
showauto(g)

Arguments

g	in	use gui if available Allowed:boolean Default: T

Returns
bool

Description
showauto will display (and update automatically if an automatic frame update is active) any active conversions. It normally will start autonomously when specifying an autoframe, and if conversions are active. Manual filling of conversions that can be updated can be done with the doshowauto function.

Example

```
- dm.doshowauto(dm.epoch('utc','today'), 'last') # auto update to LAST
T
- dm.showauto(F) # and show it on terminal
[type=epoch, refer=LAST, m0=[value=57505.5613, unit=d]]
T
```

---

measures.toradialvelocity - Function

Package	**general**
Module	**measures**
Tool	**measures**

convert a doppler type value to a real radial velocity

Synopsis
toradialvelocity(rf, v0)

Arguments

rf	in	radial velocity reference type Allowed:string
v0	in	doppler value Allowed:doppler measure

Returns
radialvelocity measure

Description
toradialvelocity will convert a Doppler type value (e.g. in radio mode) to a real radialvelocity. The type of velocity (e.g. LSRK) should be specified

Example

```
- print a := dm.doppler('radio',0.4)
[type=doppler, refer=radio, m0=[unit=m/s, value=1.19917e+08]]
- dm.toradialvelocity('topo',a)
[type=radialvelocity, refer=TOPO, m0=[unit=m/s, value=1.41079e+08]]
```

---

measures.tofrequency - Function

Package	**general**
Module	**measures**
Tool	**measures**

convert a doppler type value to a frequency

Synopsis
tofrequency(rf, v0, rfq)

Arguments

rf	in	frequency reference type Allowed:string
v0	in	doppler value Allowed:doppler measure
rfq	in	rest frequency Allowed:frequency measure or frequency quantity

Returns
frequency measure

Description
tofrequency will convert a Doppler type value (e.g. in radio mode) to a frequency. The type of frequency (e.g. LSRK) and a rest frequency (either as a frequency quantity (e.g. dm.constants('HI')) or a frequency measure (e.g. dm.frequency('rest','5100MHz')) should be specified

Example

```
- print a := dm.doppler('radio',0.4)
[type=doppler, refer=radio, m0=[unit=m/s, value=1.19917e+08]]
- dm.tofrequency('lsrk',a,dm.constants('HI'))
[type=frequency, refer=LSRK, m0=[unit=Hz, value=8.52243e+08]]
```

---

measures.todoppler - Function

Package	**general**
Module	**measures**
Tool	**measures**

convert a frequency or radialvelocity measure to a doppler measure

Synopsis
todoppler(rf, v0, rfq)

Arguments

rf	in	doppler reference type Allowed:string
v0	in	radial velocity or frequency Allowed:radialvelocity or frequency measure
rfq	in	rest frequency Allowed:frequency measure or frequency quantity Default: F

Returns
frequency measure

Description
todoppler will convert a radialvelocity measure or a frequency measure to a doppler measure. In the case of a frequency, a rest frequency has to be specified. The type of doppler wanted (e.g. RADIO) has to be specified.

Example

```
- print f := dm.frequency('lsrk','1410MHz')     # specify a frequency
[type=frequency, refer=lsrk, m0=[value=1.41e+09, unit=Hz]]
- dm.todoppler('radio', f, dm.constants('HI')) # give doppler, using HI rest
[type=doppler, refer=radio, m0=[value=2196249.84, unit=m/s]]
```

---

measures.torestfrequency - Function

Package	**general**
Module	**measures**
Tool	**measures**

convert a frequency and doppler measure to a rest frequency

Synopsis
torestfrequency(v0, d0)

Arguments

v0	in	frequency reference type Allowed:string
d0	in	doppler value Allowed:doppler measure

Returns
frequency measure

Description
torestfrequency will convert a frequency measure and a doppler measure (e.g. obtained from another spectral line with a known rest frequency) to a rest frequency.

Example

```
- print dp := dm.doppler('radio', '2196.24984km/s')  # a measured doppler speed
[type=doppler, refer=radio, m0=[value=2196249.84, unit=m/s]]
- print f := dm.frequency('lsrk','1410MHz')    # a measured frequency
[type=frequency, refer=lsrk, m0=[value=1.41e+09, unit=Hz]]
- dm.torestfrequency(f, dp)                   # the corresponding rest frequency
[type=frequency, refer=REST, m0=[value=1.42040575e+09, unit=Hz]]
```

---

measures.rise - Function

Package	**general**
Module	**measures**
Tool	**measures**

get rise and set sidereal time

Synopsis
rise(crd, ev)

Arguments

crd	in	direction of source Allowed:direction measure
ev	in	elevation limit Allowed:angle quantity Default: '5deg'

Returns
record with rise and set siodereal time quantities; or a 2 element string with below or above

Description
rise will give the rise/set hour-angles of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used.

Example

```
- dm.rise(dm.direction('sun'))
[rise=[value=267.12445, unit=deg], set=[value=439.029964, unit=deg]]
- dq.form.long(dm.rise(dm.direction('sun')).rise)
17:48:29.868
```

---

measures.riseset - Function

Package	**general**
Module	**measures**
Tool	**measures**

get rise and set times

Synopsis
riseset(crd, ev)

Arguments

crd	in	direction of source Allowed:direction measure
ev	in	elevation limit Allowed:angle quantity Default: '5deg'

Returns
record with rise and set epoch or strings

Description
rise will give the rise/set times of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used. The returned value is a record with a 'solved' field, which is F if the source is always below or above the horizon. In that case the rise and set fields will all have a string value. The record also returns a rise and set record, with 'last' and 'utc' fields showing the rise and set times as epochs.

Example

```
- dm.riseset(dm.direction('sun'))
[solved=T,
 rise=[last=[type=epoch, refer=LAST, m0=[value=0.0731388605, unit=d]],
       utc=[type=epoch, refer=UTC, m0=[value=52085.8964, unit=d]]],
 set=[last=[type=epoch, refer=LAST, m0=[value=0.455732593, unit=d]],
       utc=[type=epoch, refer=UTC, m0=[value=52086.2779, unit=d]]]]
- dm.riseset(dm.direction('sun'), dq.unit('80deg'))
[solved=F,
 rise=[last=below, utc=below],
 set=[last=below, utc=below]]
- dq.form.long(dm.riseset(dm.direction('sun')).rise.utc.m0)
21:30:47.439
```

---

measures.posangle - Function

Package	**general**
Module	**measures**
Tool	**measures**

get position angle of two directions

Synopsis
posangle(m1, m2)

Arguments

m1	in	direction of source Allowed:direction measure
m2	in	direction of other source Allowed:direction measure

Returns
position angle as angle quantity

Description
posangle will give the position angle from a direction to another. I.e. the angle in a direction between the direction to the North pose and the other direction.

Example

```
- a:=dm.direction('j2000','0deg','70deg');
- b:=dm.direction('j2000','0deg','80deg');
- dm.posangle(a,b)
[value=-0, unit=deg]
- dm.separation(a,b)
[value=10, unit=deg]
- tim := dm.epoch('utc','today') # set the time
- dm.doframe(tim)
T
- pos := dm.observatory('ATCA') # set where
- dm.doframe(pos)
T
- dm.posangle(a,b)
[value=7.99647705, unit=deg]
```

---

measures.separation - Function

Package	**general**
Module	**measures**
Tool	**measures**

get separation angle between two directions

Synopsis
separation(m1, m2)

Arguments

m1	in	direction of source Allowed:direction measure
m2	in	direction of other source Allowed:direction measure

Returns
position angle as angle quantity

Description
separation will give the separation of a direction from another as an angle.

Example

```
- a:=dm.direction('j2000','0deg','70deg');
- b:=dm.direction('j2000','0deg','80deg');
- dm.separation(a,b)
[value=10, unit=deg]
- tim := dm.epoch('utc','today')             # set the time
- dm.doframe(tim)
T
- pos := dm.observatory('ATCA')              # set where
- dm.doframe(pos)
T
- c:=dm.measure(b,'azel')                    # try with different type
- dm.separation(a,c)
[value=10, unit=deg]
```

---

measures.addxvalue - Function

Package	**general**
Module	**measures**
Tool	**measures**

get some additional measure information

Synopsis
addxvalue()

Arguments

Returns
string

Description
addxvalue will give some additional information about some measures as a vector of quantities. It is used internally to get the rectangular coordinates of measures that are normally given in angles. The casual user will probably in general not interested in this function.

Example

```
- print a := dm.observatory('atca')
[type=position, refer=ITRF, m2=[value=6372960.26, unit=m],
 m1=[unit=rad, value=-0.52613792],
  m0=[unit=rad, value=2.61014232]]
- dm.addxvalue(a)
[__*0=[value=-4750915.84, unit=m], __*1=[value=2792906.18, unit=m],
   __*2=[value=-3200483.75, unit=m]]
- dm.addxvalue(a)::
[id=quant, shape=3]
- dm.addxvalue(dm.epoch('utc','today'))
[=]
- dm.addxvalue(dm.epoch('utc','today'))::
[id=quant, shape=0]
```

---

measures.id - Function

Package	**general**
Module	**measures**
Tool	**measures**

identification of tool

Synopsis
id()

Arguments

Returns
object id

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

Example

```
- dm.id()
[sequence=1, pid=4484, time=1020733850, host=cetus, agentid=4]
```

---

measures.type - Function

Package	**general**
Module	**measures**
Tool	**measures**

type of tool

Synopsis
type()

Arguments

Returns

string

Description
type will return the tool name.

Example

```
- dm.type()
measures
```

---

measures.done - Function

Package	**general**
Module	**measures**
Tool	**measures**

free resources used by tool

Synopsis
done(kill)

Arguments

kill	in	force kill of the default tool Allowed:bool 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:=measures()
- a.done()
T
- dm.done()
F
- dm.done(kill=T)
NORMAL: Successfully closed empty server: measures
T
```

1.4.3 measuresgui - Tool

Package general
Module measures

gui code for measures.g
include "measuresgui.g"

Description

measuresgui provides the code for the GUI interface to the units, quantities and measures 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 measures closure (using dm.gui()).
