

Geopoints

Geopoints is the format used by Metview to handle spatially irregular data (e.g. observations) in a non-BUFR format. For a full list and details of functions and operators on geopoints, see [Geopoints Functions](#).

Geopoints is an ASCII format which can be generated 'by hand', or could be the result of a Metview computation. For example, the [Observation Filter](#) icon can return a geopoints variable as the result of filtering BUFR data (see [Observation filtering and plotting Example](#)). Several geopoints variables can be contained within a [Geopointset](#) variable (currently supported by Macro and Python only, not by the user interface or the plotting).

The Metview Macro language has a number of functions available for the input, output, handling and operating on geopoints. Geopoints can be visualised with Metview and can be customised using symbol visual definitions - see [Symbol Plotting](#). Metview can also convert between Geopoints and GRIB - see [Geopoints To Grib](#) and [Grib To Geopoints](#).

The Geopoints file formats

A geopoints file is an ASCII file containing a header section and a data section consisting of several columns. There are some different 'flavours' of the format, described below.

The data elements that can be present in a geopoints file are the point coordinates (latitude and longitude), the level, date, time and value of either one or two parameters. A two-parameter geopoints file is considered by the plotting engine to contain the components of a vector quantity such as wind.

The format does not care about the alignment of columns, it just requires that there is at least one whitespace character between entries. The elements that must be present are an initial line tagged with the keyword **#GEO**, a line tagged with the keyword **#DATA** and the data points themselves. There can also be an optional section for meta-data, which must start with **#METADATA** - see section below for details. The lines in-between the header and the data sections are for human-readable information only and are not used in the interpretation of the file. All formats apart from the Standard format require an additional line in the header section to specify the format; this line must start with **#FORMAT** followed by the name of the format being used.

Note that a time should be expressed as HHMM; a time of 12 will be interpreted as 0012, ie 00:12.

Standard (6-column) geopoints

This is the default format that Metview uses. This example shows a geopoints file containing dry bulb temperature at 2m (PARAMETER = 12004).

```
#GEO
PARAMETER = 12004
lat      long    level  date      time     value
#DATA
36.15    -5.35    0    19970810    1200     300.9
34.58    32.98    0    19970810    1200     301.6
41.97    21.65    0    19970810    1200     299.4
45.03     7.73    0    19970810    1200      294
45.67     9.7    0    19970810    1200     302.2
44.43     9.93    0    19970810    1200     293.4
```

#FORMAT XYV (Compact format)

This format allows data to be specified with just three columns: X (longitude), Y (latitude) and V (value). The start of an example file would look like the following:

```
#GEO
#FORMAT XYV
PARAMETER = 12004
x/long y/lat value
#DATA
-5.35  36.15  300.9
32.98  34.58  301.6
21.65  41.97  299.4
```

#FORMAT XY_VECTOR (XY Vector format)

This format allows two parameters to be stored as the components of a two-dimensional vector (for example uv wind components). The start of an example file would look like the following:

```
#GEO
#FORMAT XY_VECTOR
# lat lon height date time u v
#DATA
80 0 0 20030617 1200 -4.9001 -8.3126
80 5.5 0 20030617 1200 -5.6628 -7.7252
80 11 0 20030617 1200 -6.4254 -7.13829
```

#FORMAT POLAR_VECTOR (Polar Vector format)

This format allows two parameters to be stored as the speed and direction of a two-dimensional vector, the direction being specified in degrees where zero is due South and angles increase clockwise. The start of an example file would look like the following:

```
#GEO
#FORMAT POLAR_VECTOR
# lat lon height date time speed direction
#DATA
50.97 6.05 0 20030614 1200 4 90
41.97 21.65 0 20030614 1200 5 330
35.85 14.48 0 20030614 1200 11 170
```

#FORMAT NCOLS (Multi-column format)

This format allows any number of parameters to be stored in a geopoints file. The #COLUMNS section is used to understand the columns, as they can be put in any order. The following column names are reserved and are treated specially: **longitude**, **level**, **date**, **time**, **stnid**. A column with a different name will be treated as a value column. The data should all be numeric, apart from **stnid**, which is stored as a string.

The start of an example file would look like the following:

```
#GEO
#FORMAT NCOLS
#COLUMNS
latitude longitude time date t2 o3 td rh
#DATA
32.55 35.85 0600 20120218 273.9 35 280.3 75
31.72 35.98 1800 20120218 274.9 24 290.4 68
51.93 8.32 1200 20140218 278.9 28 300.5 34
41.1 20.82 1200 20150218 279.9 83 310.6 42
```

For Polar Vector geopoints, only the first value (speed) is considered during operations. For XY geopoints, both values are considered during most operations where it makes sense to do so. For the NCOLS format, all value columns are manipulated during operations.

Currently the level, date and time can only be used for filtering (or can be extracted into [Vector](#) variables for other uses). They must be present in the file but you can specify any dummy value if you do not intend to use them.

Storing and retrieving meta-data

A geopoints file can have a section of meta-data key-value pairs in its header before the **#DATA** section, as illustrated here:

```
#GEO
PARAMETER = 12030
#METADATA
param=temperature
date=20130804
time=1200
level=0.2
#lat      long      level      date      time      value
#DATA
55.01     8.41     0.2       20130804  1200     294.4
54.33     8.60     0.2       20130804  1200     296.9
```

Here, four pieces of meta-data are stored. They can be set and queried in the Macro (or Python) language, like this:

```
data = read('geopoints_with_metadata.gpt')
md = metadata(data)
print(md)
print(md['level'])
```

Output:

```
(date:20130804,level:0.2,param:temperature,time:1200)
0.2
```

Meta-data can also be set by passing a definition like this:

```
gpt_new = set_metadata(gpt, (mykey1:'vall', mykey2: 5))
```

If geopoints variables contain meta-data and they are part of a geointset, they can be filtered on their meta-data - see [Geointset](#) for details.

Extracting and setting columns

There are two ways to extract columns of data from a geopoints variable.

1. Use the functions provided, e.g.

```
lats = latitudes(gpt)
vals = values(gpt)
rh = values(gpt, 'rh') # assuming NCOLS format with a value column of name 'rh'
```

2. Use column indexing, e.g.

```
lats = gpt['latitude']
vals = gpt['value']
rh = gpt['rh'] # assuming NCOLS format with a value column of name 'rh'
```

To assign values to a column, again there are 2 methods, but they have different behaviours:

1. Use the `set_` functions provided - these create new geopoints variables and do not modify the originals, e.g.

```
gpt_new = set_latitudes(gpt, lats) # lats is a vector
```

2. Use column indexing - this modifies the original geopoints variable and is therefore more efficient, e.g.

```
gpt['latitude'] = lats # lats is a vector
```

Operations between geopoints and fieldsets

When you carry out an operation between geopoints and fieldset (or images) variables the result is another geopoints variable :

- When operating with fieldsets, the values of the field(s) at the geopoints locations are calculated by interpolation and the resulting field values undergo the operation with the geopoints values
- When combining with an image no interpolation takes place; the pixel values where the geopoints are located are extracted and these undergo the operation with the geopoints values
- Unless otherwise stated in the operator or function description, only the first value of a two-valued geopoint is considered during a calculation

Combinations include algebraic operations, boolean operations and a number of functions. See [Geopoints Functions](#) for details.

Missing values in geopoints

When you combine fieldset data with geopoints, you may end up with some missing values in your geopoints variable. These will have the value contained in the built-in global variable `geo_missing_value`. Any operation on a geopoints variable will bypass missing values (e.g. `mean()`) or retain them unaltered (e.g. `max()`); see individual function descriptions for more details.

In order to remove missing values from a geopoints variable, use the function `remove_missing_values()` as illustrated below:

```
geo_clean = remove_missing_values (geo_source)
```

Missing coordinates in geopoints

It is possible (since Metview 5.7.0) to include missing values in the latitude or longitude columns (or both). A point with either coordinate missing will be excluded from any operation that requires location information.