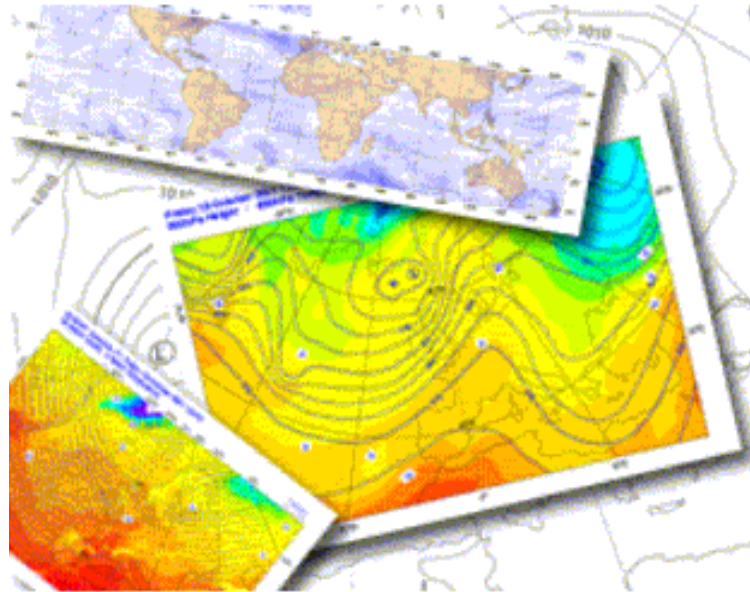


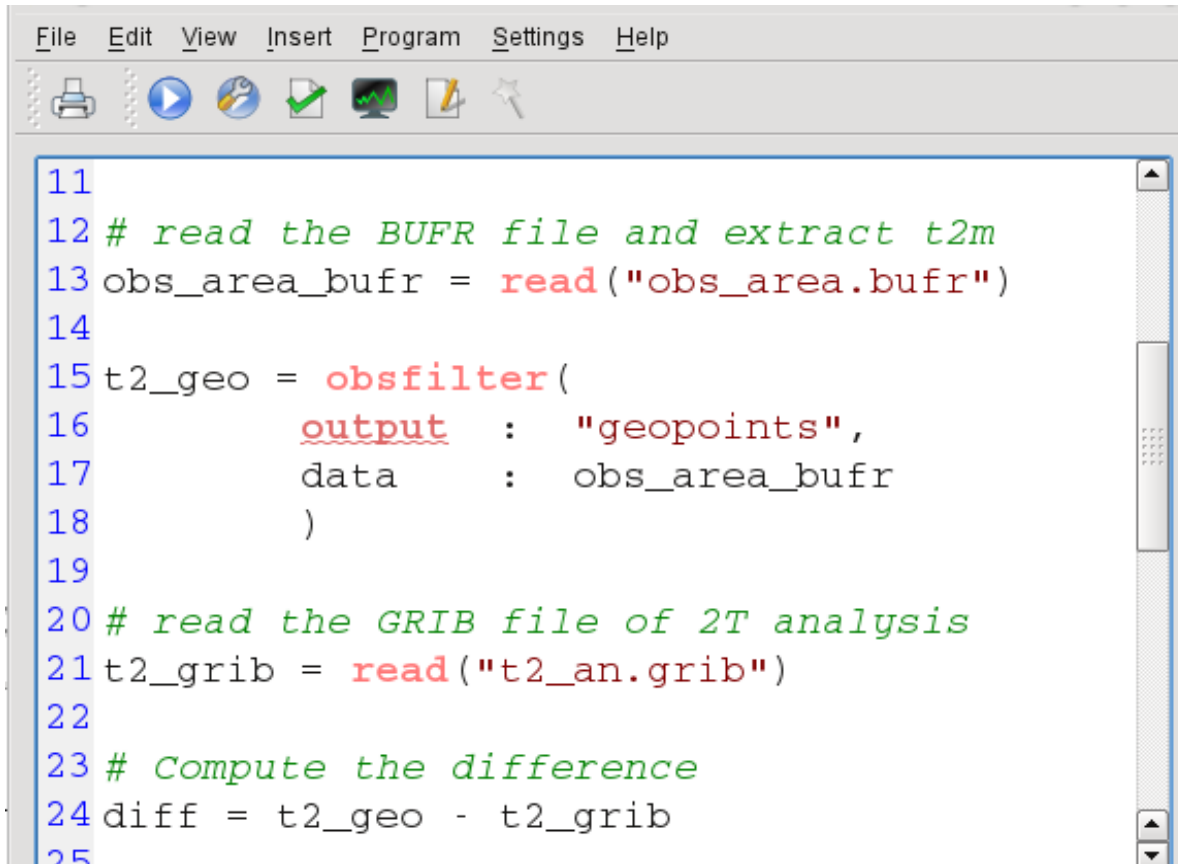
Metview – Macro Language



Iain Russell

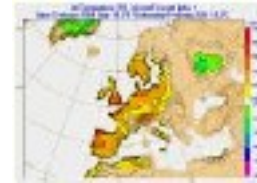
Meteorological Visualisation Section, ECMWF

- Designed to perform data manipulation and plotting from within the Metview environment



```
File Edit View Insert Program Settings Help
[Printer] [Play] [Wrench] [Checkmark] [Monitor] [Pencil] [Star]
11
12 # read the BUFR file and extract t2m
13 obs_area_bufr = read("obs_area.bufr")
14
15 t2_geo = obsfilter(
16     output : "geopoints",
17     data   : obs_area_bufr
18 )
19
20 # read the GRIB file of 2T analysis
21 t2_grib = read("t2_an.grib")
22
23 # Compute the difference
24 diff = t2_geo - t2_grib
25
```

Usage of Macro Language



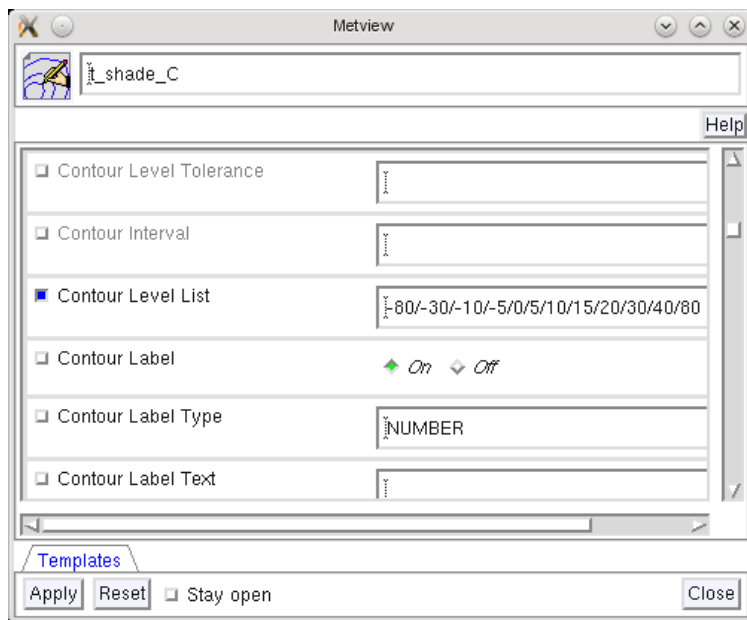
- Can run from within the Metview interface, or from the command line (`metview -b <macro>`) - therefore can be run as part of a scheduled task
- Used to generate plots or derived data sets (can then be used as input to other icons)
- Macros can provide their own (simple) user interfaces if desired

Macro Introduction

- Icons can be automatically translated into Macro code

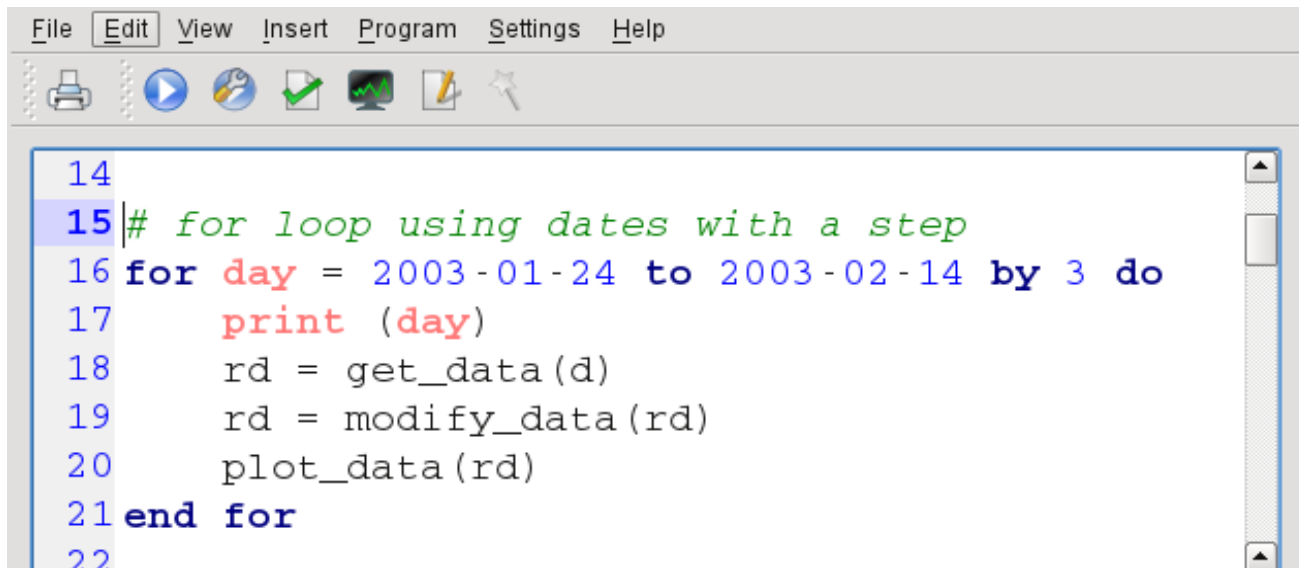


t_shade_C



```
Macro* - /home/graphics/cgi/metview/tutorials/webinars_2013/Macro
File Edit View Insert Program Settings Help
1 # Importing : /tutorials/webinars_2013/t_shade_C
2
3 t_shade_c = mcont(
4   legend                               : "on",
5   contour_highlight                     : "off",
6   contour_level_selection_type         : "level_list",
7   contour_level_list                    : [-80,-30,-10,-5,0,5,10,15,20,30,40,80],
8   contour_shade                         : "on",
9   contour_shade_method                  : "area_fill",
10  contour_shade_max_level_colour        : "red",
11  contour_shade_min_level_colour        : "blue",
12  contour_shade_colour_direction        : "clockwise"
13 )
14
15
File saved L: 12, C: 49
```

- Able to describe complex sequences of actions
 - ◆ Various loop types and conditionals



```
File Edit View Insert Program Settings Help
[Icons: Print, Play, Refresh, Checkmark, Stop, Erase, Pointer]
14
15 # for loop using dates with a step
16 for day = 2003-01-24 to 2003-02-14 by 3 do
17     print (day)
18     rd = get_data(d)
19     rd = modify_data(rd)
20     plot_data(rd)
21 end for
22
```

- Easy as a script language - no variable declarations or program units; typeless variables ; built-in types for meteorological data formats

```
/home/graphics/cgi/metview/macro_tutorial_prep/for_overheads/basic 853 bytes L: 36 C: 0

# Load various data files

fs_rain    = read ("rain.grib")           # loads as a fieldset
geo_rain   = read ("rain_points.txt")    # loads as geopoints
ncdf_rain  = read ("rain.netcdf")        # loads as netcdf

print(type(fs_rain))                     # output: "fieldset"
print(type(geo_rain))                    # output: "geopoints"
print(type(ncdf_rain))                   # output: "netcdf"
```

- **Complex as a programming language - support for variables, flow control, loops, functions, I/O and error control**
- **Can store functions in a common area (creating a macro library)**

```
home/graphics/cgi/metview/macro_tutorial_prep/for_overheads/basic 979 bytes L: 45 C: 0

home = getenv("HOME")
path = home & "/metview/test_data.grib"

if (not(exist(path))) then
    fail("file does not exist")
end if
```

- Interfaces with user's FORTRAN and C programs

```
/home/graphics/cgi/metview/macro_tutorial_prep/macro_tut1/gradientb.f 3154 bytes L: 12 C: 0
C
C  "GRADIEN" COMPUTES
C
C  THIS PROGRAM IS A MO
C  "GRADIEN" TO TAKE 1
C
PROGRAM GRADIEN
PARAMETER (ISIZE=
DIMENSION ISEC0(2
DIMENSION ISEC1(1

***  GET FIRST ARGUMEN

CALL MGETG(IGRIB1

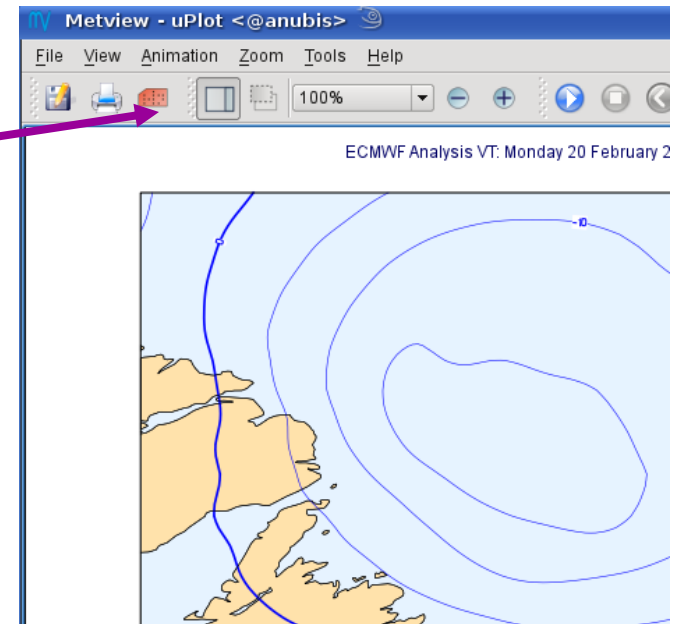
graphics/cgi/metview/macro_tutorial_prep/for_overheads/basic 1181 bytes L: 55 C: 27
extern gradientb(f:fieldset) "gradientb"

# Retrieve the specific humidity
q = retrieve (
    date      : -1,
    param     : "q",
    level     : 700,
    grid      : [1.5,1.5]
)

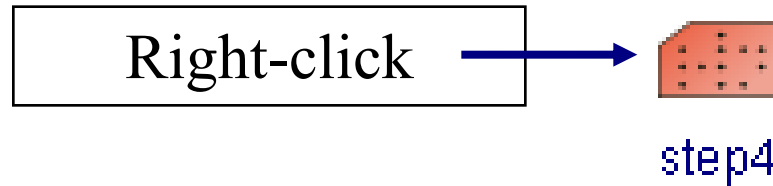
# Compute the gradient of Q
q = gradientb(q)
```


Creating a Macro Program

- Save visualisation as Macro - limited in scope
- Drop icons inside Macro Editor, add extra bits
- Write from scratch (the more macros you write, the more you recycle those you have done, lessening the effort)

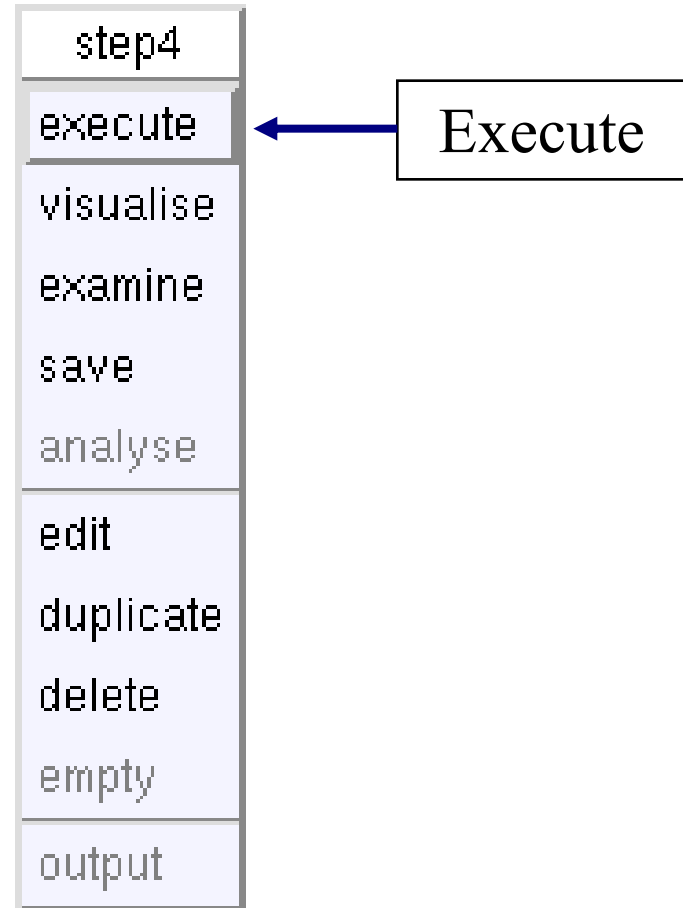


Demo!



```
1 # Metview Macro
2
3 mode = runmode ()
4 if      mode = 'visualise' then ...
5 else if mode = 'save'      then ...
6 else ...
7
```

e.g. if mode = 'batch' then take arguments from command line; otherwise bring up a user interface



- No need for declaration
- Dynamic typing

```
a = 1           # type(a) = 'number'  
a = 'hello'     # type(a) = 'string'  
a = [4, 5]      # type(a) = 'list'  
a = |7, 8|      # type(a) = 'vector'
```

Macro Essentials - Strings

- Character strings, can be of any length. Use double or single quotes. Example code:

```
3 a = 'Hello world'  
4 print(a)  
5 print(parse(a, ' '))  
6 print(uppercase(a))  
7 print('wo occurs at char ', search(a, 'wo'))  
8 print(substring(a, 2, 4))  
9 a = a & ', nice to see you'  
10 print(a)
```

```
Hello world  
[Hello,world]  
HELLO WORLD  
wo occurs at char 7  
ell  
Hello world, nice to see you
```

Program finished (OK) : 16 ms [Finished at 11:47:23]

L: 9, C: 28

- Dates defined as a built-in type - year, month, day, hour, minute and second. Example:

```
1 # Metview Macro
2
3 d = 2013-05-23 # construct a date: YYYY-MM-DD [HH:MM[:SS]]
4 print('d: ', d)
5 dm100 = d - 100 # 100 days before 'd'
6 print('dm100: ', dm100)
7 print('dm100+0.5: ', dm100 + 0.5) # add half a day
8 print('yesterday: ', date(-1)) # yesterday
9
```

```
d: 2013-05-23 00:00:00
dm100: 2013-02-12 00:00:00
dm100+0.5: 2013-02-12 12:00:00
yesterday: 2013-05-22 00:00:00
```

Program finished (OK) : 9 ms [Finished at 11:01:20]

L: 9, C: 1

- Ordered list of values of mixed types

```
2 a = [10, 20, 'a', 'b', ['z', 'y']]
3 print(type(a), ': ', a)
4 print(count(a))
5 print('element 1 is ', a[1])
6 print('elements2-4 are ', a[2,4])
7 types = nil # empty string
8 loop thing in a # loop through all list elements
9     types = types & type(thing) & ', '
10 end loop
11 print(types)
```

```
list: [10,20,a,b,[z,y]]
5
element 1 is 10
elements2-4 are [20,a,b]
number, number, string, string, list,
```

Program finished (OK) : 14 ms [Finished at 12:09:40]

L: 8, C: 50

- Ordered, array of numbers. Much more efficient than lists for high volumes of numeric data. Vectors are built using the vertical bar symbol, and can be appended to, or pre-allocated for efficiency

```
v = |7, 8, 9|
```

```
v = v & |5,6| # v is now |7,8,9,5,6|
```

```
v = vector(10000) # pre-allocate space
```

```
v[1] = 4 # assign values to indexes
```

- Operations and functions are applied to each element:

```
x = | 3, 4, 5 |
```

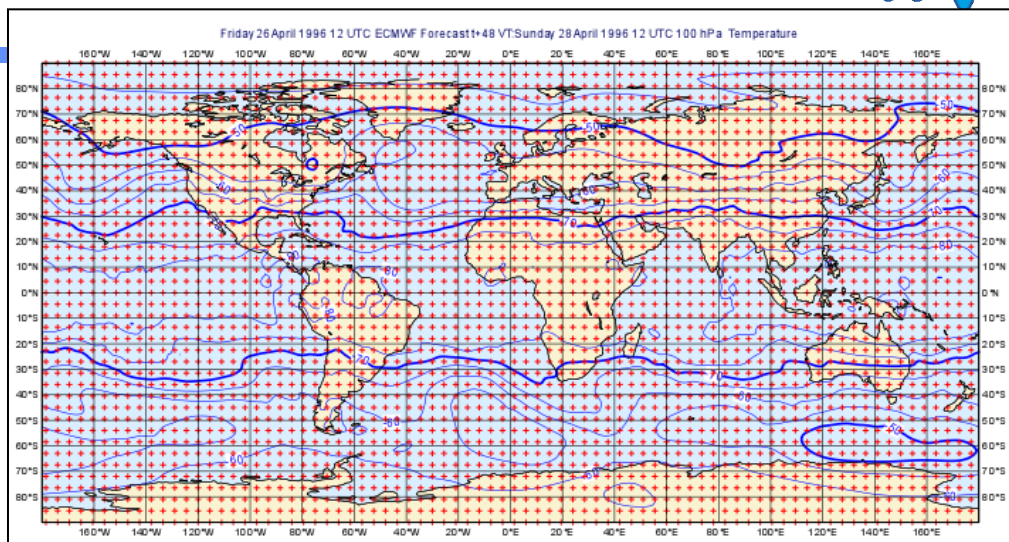
```
y = x + 10 # y is now |13, 14, 15|
```

```
c = cos(x)
```

```
u = | 7.3, 4.2, 3.6 |
```

```
v = | -4.4, 1.1, -2.1 |
```

```
spd = sqrt((u*u) + (v*v))
```

- **Definition**

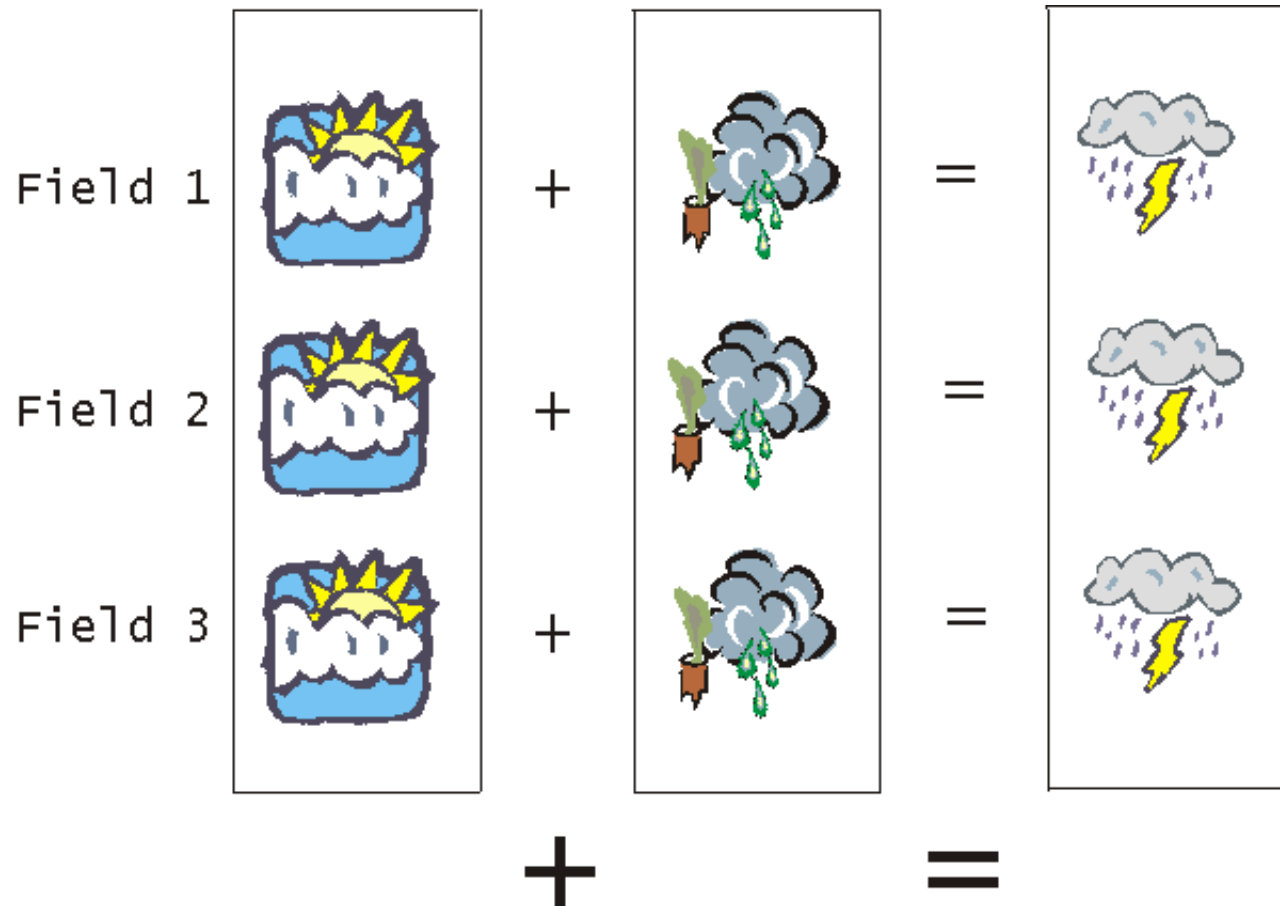
- ◆ Entity composed of several meteorological fields, (e.g. output of a MARS retrieval). Associated with GRIB files.

- **Operations and functions on fieldsets**

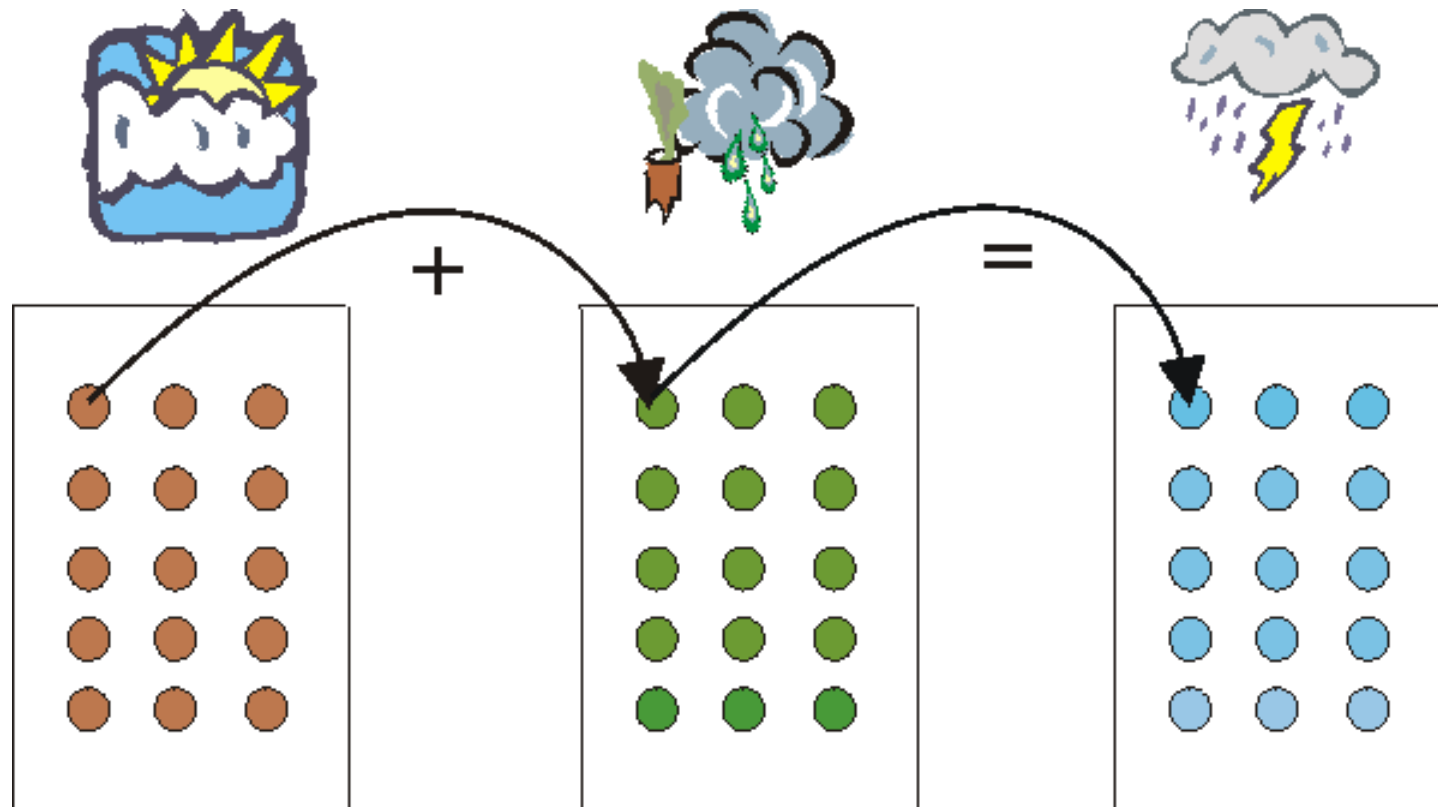
- ◆ Operations on two fieldsets are carried out between each pair of corresponding values within each pair of corresponding fields. The result is a new fieldset.

`result = fieldset_1 + fieldset_2`

Macro Essentials - Fieldsets



Macro Essentials - Fieldsets



- Operations and functions on fieldsets

- ◆ Can also combine fieldsets with scalars:

$$Z = X - 273.15$$

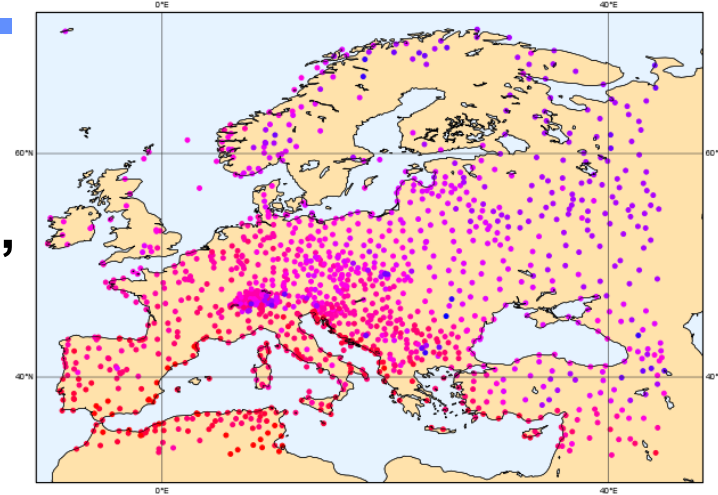
Gives a fieldset where all values in each field are 273.15 less than the original (Kelvin to Celcius)

- **Many many functions available for fieldsets!**
- **See Macro Tutorial 3 for some examples, such as masking one field based on the values of another (e.g. apply a land-sea mask to remove all sea-based points from a field)**

Macro Essentials - Geopoints



- Hold spatially irregular data
- ASCII format file (with 4 sub-formats, default shown here)



#GEO

PARAMETER = 2m Temperature

lat	long	level	date	time	value
-----	------	-------	------	------	-------

#DATA

36.15	-5.35	850	19970810	1200	300.9
-------	-------	-----	----------	------	-------

34.58	32.98	850	19970810	1200	301.6
-------	-------	-----	----------	------	-------

41.97	21.65	850	19970810	1200	299.4
-------	-------	-----	----------	------	-------

- Operations on geopoints

- ◆ Generally create a new set of geopoints, where each value is the result of the operation on the corresponding input value

- ◆ `geo_new = geo_pts + 1`

- Means "add 1 to each geopoint value, creating a new set of geopoints".

(3,	4,	5,	6,	7,	8)
↓	↓	↓	↓	↓	↓
(4,	5,	6,	7,	8,	9)

- **Combining Fieldsets And Geopoints**

- ◆ **Combination is done automatically by Metview Macro :**

- - for each geopoint, find the corresponding value in the fieldset by interpolation
- - now combine corresponding values (add, subtract etc.)
- - the result is a new geopoints variable
- - only considers the first field in a fieldset

- ◆ **Can compute difference between model field and observations in a single line of code ($a = b - c$)**

- **See Macro Tutorial 2 for examples!**

- **ASCII Tables – columns of data in text files**
 - ◆ **E.g. CSV (Comma Separated Value)**
 - ◆ **Various parsing options for different formats (use the *Table Reader* icon to help)**
- **Metview can directly visualise these, or read columns of data into vectors (numeric) or lists of strings (text)**
- **Metview can currently only *read* ASCII Tables, not *write***

```
Station,Lat,Lon,T2m
1,71.1,28.23,271.3
2,70.93,-8.67,274.7
```

```
t2_csv = read_table(
    table_filename : 't2m.csv')
vals = values(t2_csv, 'T2m')
# vals is now a vector
```

- NetCDF support in Metview – example code

```
1 # Metview Macro
2
3 z500 = read("z500.nc")
4 print(global_attributes(z500))
5 setcurrent(z500, 3)      # select the 3rd variable
6 print(attributes(z500)) # print this var's attributes
7 vals = values(z500)     # extract the values for this var
8 print(maxvalue(vals))   # compute max value
9
```

```
.....
ATTRIBUTES (Conventions:MARS,stream:da,date:20030529,time:12
00,step:0,expver:1,class:od,type:an,levtype:pl,levelist:500
,repres:11,domain:g)
ATTRIBUTES(scale_factor:0.161463,add_offset:52807.557294,un
```

- A collection of named items (members)
- Eg

```
a = (x : 1, y : 2) # create definition  
  
c = a.x           # get value of 'x'  
or  
c = a["x"]
```

- Can dynamically add/change members, e.g.

```
a.x = 7  
a.z = 'something'
```

- **Icon-functions take definitions:**

```
acoast = mcoast(  
    map_coastline_resolution      :    "high",  
    map_coastline_colour         :    "red",  
    map_grid_colour              :    "grey",  
    map_grid_longitude_increment :    10,  
    map_label_colour             :    "grey",  
    map_coastline_land_shade     :    "on",  
    map_coastline_land_shade_colour:  "cream"  
)
```

- **Can dynamically construct / edit these definitions**

For more information ...



email us:

 **Metview:** metview@ecmwf.int

visit our web pages:

 <https://software.ecmwf.int/metview>

- *Training / Webinars*
- **Links to optional tutorial material**
- **Download the virtual machine**

Wednesday, 26th June, 9.30am UK (8.30am UTC) : Q&A

<http://www.hipchat.com/gtOzdpBoZ>

