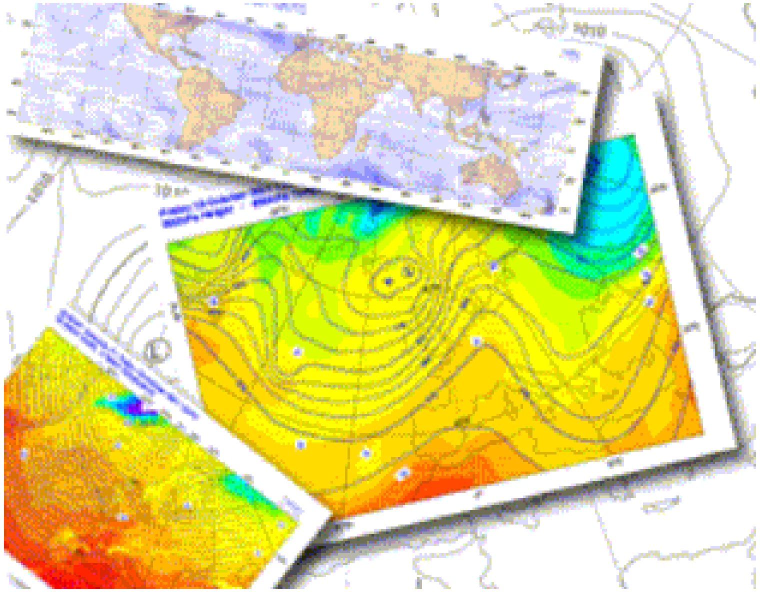
Metview – Macro Language

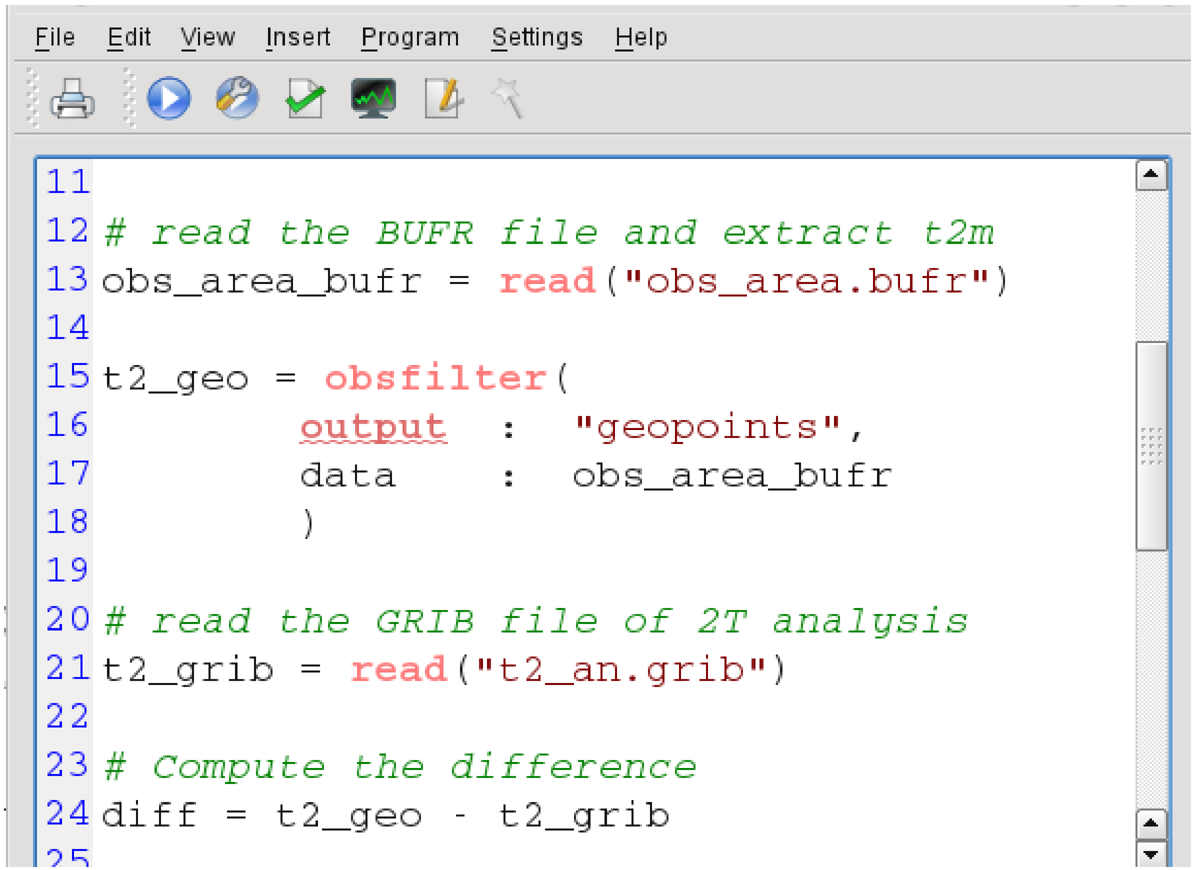


Iain Russell, Sándor Kertész, Fernando Ii

Development Section, ECMWF

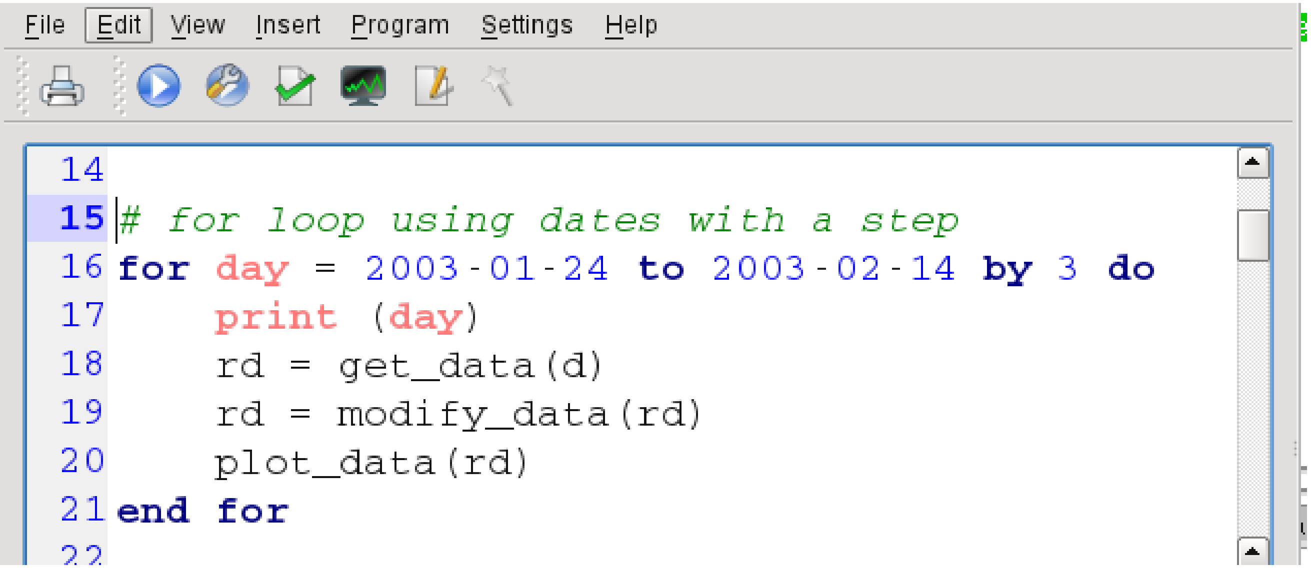
Macro Introduction

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



Macro Introduction

- **Able to describe complex sequences of actions**



Macro Introduction

- **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"
```

Macro Introduction

- **Complex as a programming language - support for variables, flow control, functions, I/O and error control**

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
```

Macro Introduction

- 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   "GRADIENTB" COMPUTES
C
C   THIS PROGRAM IS A MO
C   "GRADIENT" TO TAKE I
C

      PROGRAM GRADIENTB
      PARAMETER (ISIZE=
      DIMENSION ISEC0(2
      DIMENSION ISEC1(1

***   GET FIRST ARGUMEN

      CALL MGETG(IGRIB1
```

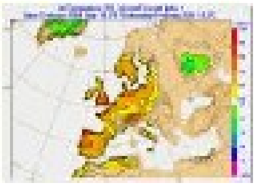
hics/cgi/metview/macro_tutorial_prep/for_overheads/basic 1181 bytes L: 55 C: 27

```
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)
```

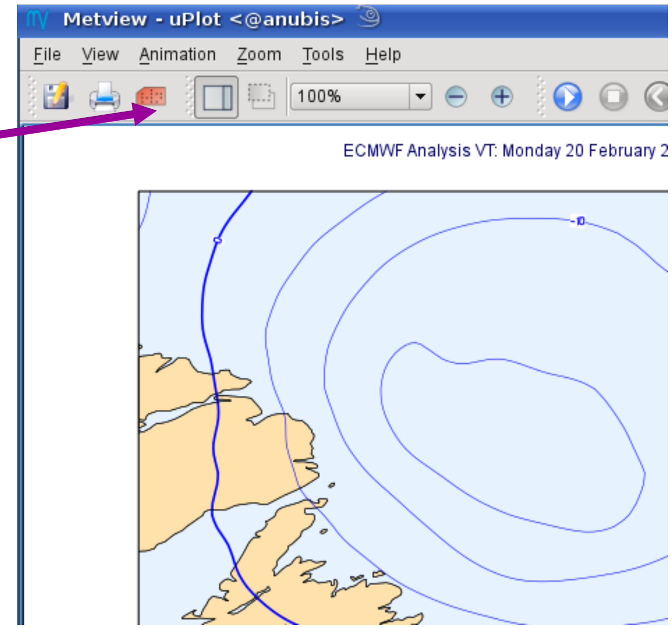
Uses of Macro Language



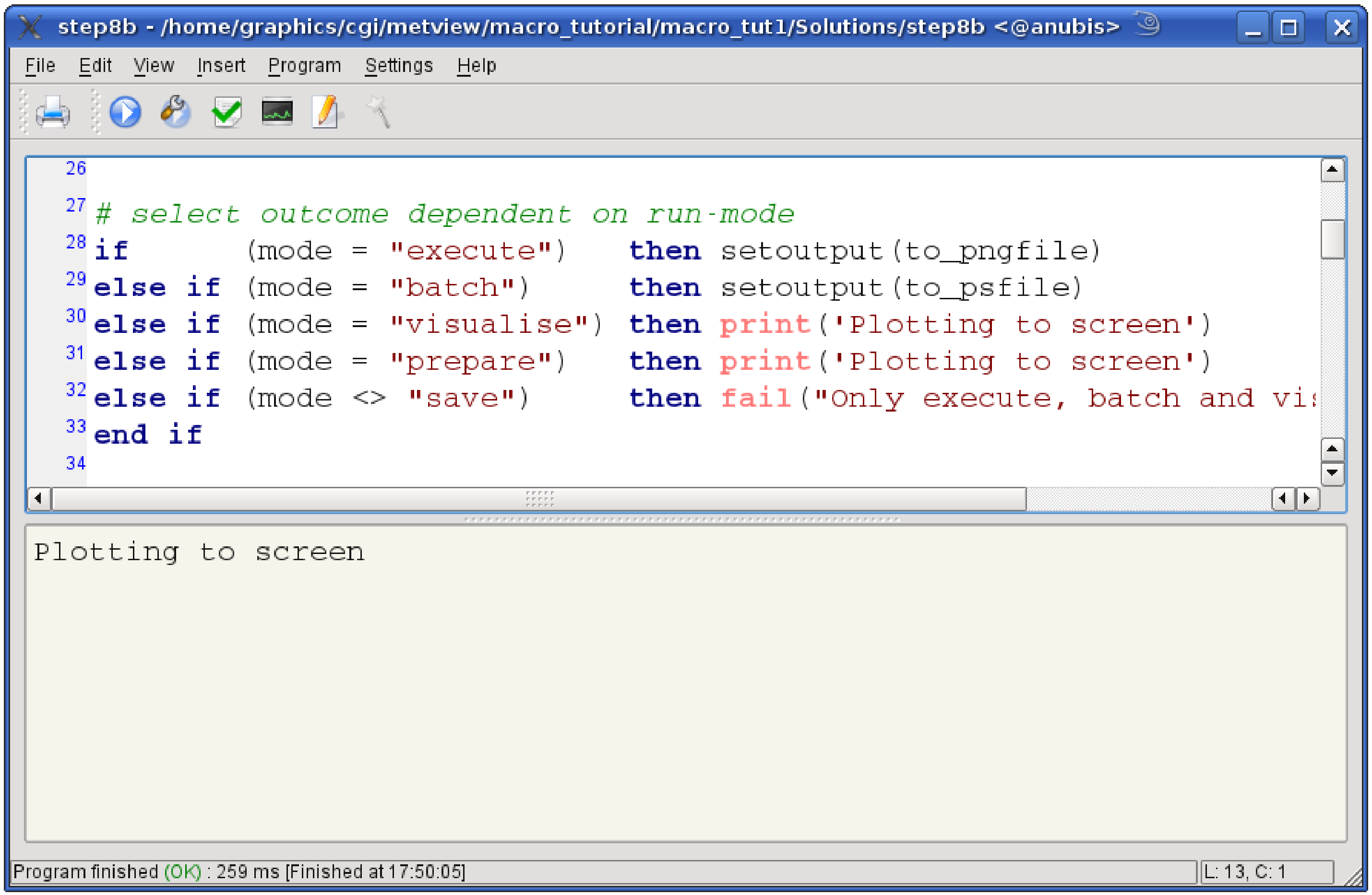
- Generate visualisation plots directly
- Generate a derived data set to save to disk, to drop in plot windows or to provide input to other Metview modules
- Provide a user interface for complex tasks
- Incorporate macros in scheduled tasks - thus use Metview in an operational environment, run in batch mode

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)



The Macro Editor

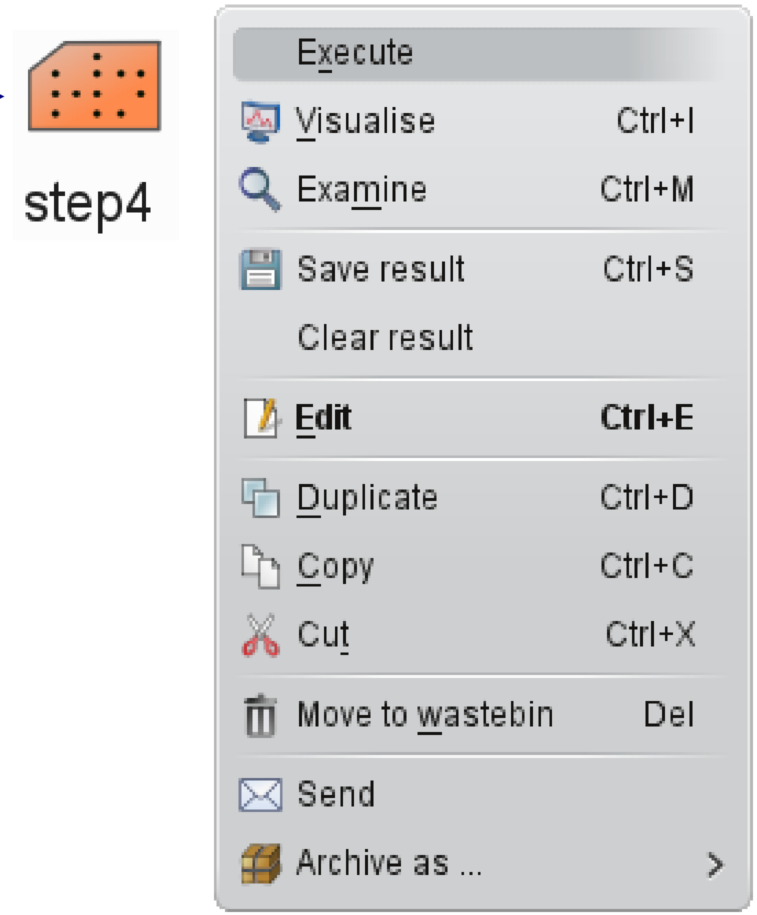


The Macro Editor

- Drop icons directly into the editor
- Run (automatically saves the macro first)
- Tab settings (Settings | Tabs…)
- Insert function name (F2)
- Insert code template (F4)
- Advanced run options

Executing Macros Another Way

Right-click



Execute

Visualise

Examine

Save result

Macro Documentation

- All Macro functions are documented in the new Metview 4 Confluence pages:
  - https://software.ecmwf.int/metview/The+Macro+Language
- Some more Metview 4 documentation there, plus tutorials
- But some things are still only in the Metview 3 documentation:
  - http://www.ecmwf.int/publications/manuals/metview
- ‘Full’ Metview 4 documentation is in progress

Data For Tutorial

- `cd ~/metview`
- `~trx/mv_data/get_macro_data`
- **Data is unzipped into**
  - `metview/macro_tutorial`

Tutorial Steps 1-4

- **Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions (start on page 5)**
- Steps 5-7 : Outputs other than on-screen
- Step 8 : Macro run mode control
- Steps 9-10 : User Interfaces in Macro
- Step 11 : Macro in Batch
- Steps 12a,b,c : Using functions in Macro (libraries)
- Embedding FORTRAN and C in Macro

Macro Essentials - Variables

- 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

- `'Hello'` is the same as `"Hello"`

- Concatenate strings with strings, numbers and dates using the `'&'` operator

  eg. `"part1_" & "part2_" & 3`

  produces `"part1_part2_3"`

- Obtain substrings with `substring()`

  e.g. `substring ("Metview", 2, 4)`

  produces `"etv"`

  first last

Macro Essentials - Strings

- Split a string into parts using `parse()`
- Creates a list of substrings

```
n = parse("z500.grib", ".")
print ("name = ", n[1], " extension = ", n[2])
```

- prints the following string :

  name = z500 extension = grib

Macro Essentials - Dates

- Dates defined as a built-in type - year, month, day, hour, minute and second.
- Dates can be created as literals using :
  - yyyy-mm-dd
  - yyyy-DDD
  - where : yr, yyyy - 4 digit yr, mm - 2 digit month, dd - 2 digit day, DDD - 3 digit Julian day.
- The time can be added using :
  - HH:MM or HH:MM:SS
  - Eg `start_date = 2003-03-20 12:01`

Macro Essentials - Dates

- Function `date()` creates dates from numbers:

```
d1        = date(20080129)
today     = date(0)
yesterday = date(-1)
```

- Hour, minute and second components are zero.
- To create a full date, use decimal dates:

```
d = date(20080129.5)
```

or

```
d = 2008-01-29 + 0.5
```

or

```
d = 2008-01-29 + hour(12)
```


ECMWF

Macro Essentials - Dates

- **Note that numbers passed to Metview modules are automatically converted to dates:**

```
r = retrieve(date : -1, ...)

r = retrieve(date : 20070101, ...)
```

Macro Essentials - Dates

- Loops on dates using a for loop:

```
for d = 2007-01-01 to 2007-03-01 do
      ... # each step is 1 day
end for
```

```
for d = 2007-01-01 to 2007-03-01 by 2 do
      ... # each step is 2 days
end for
```

```
for d = 2007-01-01 to 2007-03-01 by hour(6) do
      print(d)
      ... # each step is 6 hours
end for
```

Macro Essentials - Lists

- **Ordered, heterogeneous collection of values. Not limited in length. List elements can be of any type, including lists. List are built using square brackets, and can be initialised with `nil`:**

```
l = [3,4,"foo","bar"]

l = nil
l = l & [2,3,[3,4]]
l = l & ["str1"] & ["str2"]

europe = [35,-12.5,75,42.5]    # S, W, N, E
```

Macro Essentials - Lists

- Accessing List Elements
- Indexes start at 1

```
mylist = [10,20,30,40]
a = mylist[1]       # a = 10
b = mylist[2,4]     # b = [20,30,40] (m to n)
c = mylist[1,4,2] # c = [10,30] (step 2)
```

Macro Essentials - Lists

- **Useful List Functions**

```
num_elements = count (mylist)
sorted = sort (mylist)
  # can provide custom sorting function

 if (2 in mylist) then
    …
 end if
```

Macro Essentials - Lists

- **Useful List Functions**

```
mylist = ['b', 'a', 'a', 'c']

# find occurrences of 'a' in list
index   = find(mylist, 'a') # 2
indexes = find(mylist, 'a', 'all') # [2,3]

# return list of unique members
reduced = unique(mylist) # ['b', 'a', 'c']
```


ECMWF

Macro Essentials - Lists

- List Operations
- Operators acting on lists will act on each list element, returning a list of results
-
```
a = [3, 4]
b = a + 5   # b is now [8, 9]
c = a * b   # c is now [24, 36]
```

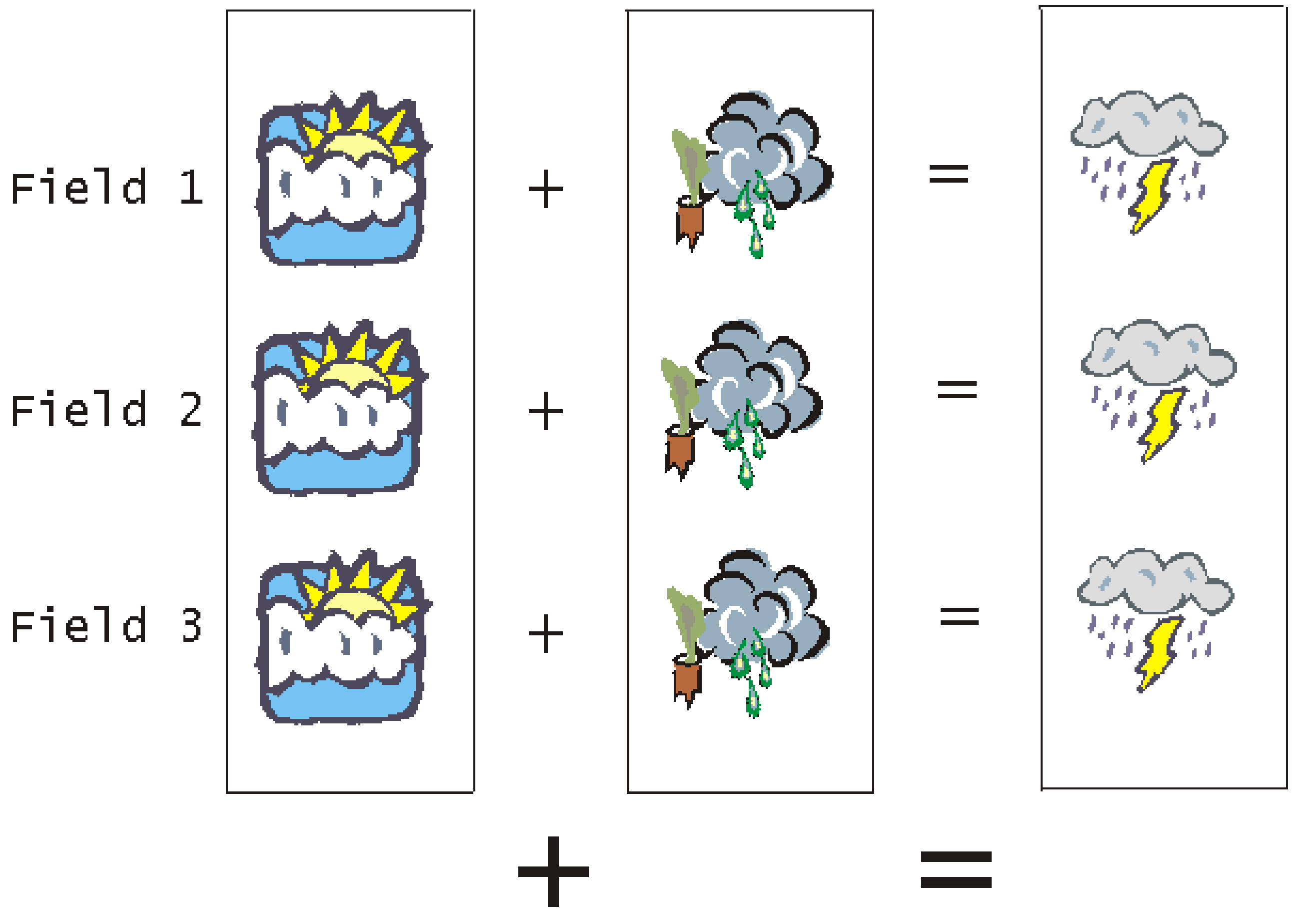
- Lists are general-purpose, and are not recommended for handling large amounts (thousands) of numbers – for that, use *vectors* (see later)

Macro Essentials - Fieldsets

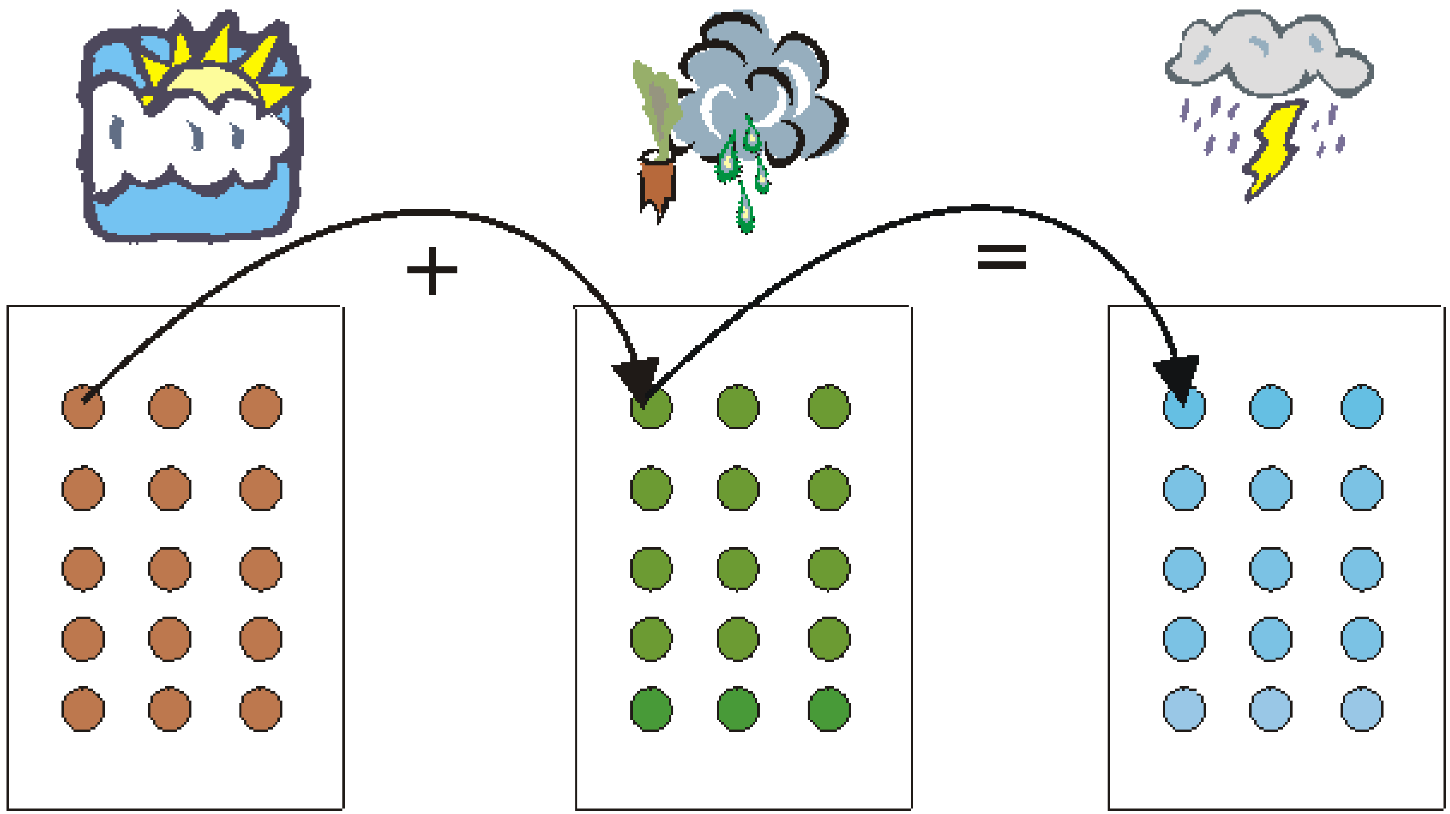
- Definition
  - Entity composed of several meteorological fields, (e.g. output of a MARS retrieval).
- Operations and functions on fieldsets
  - Operations on two fieldsets are carried out between each pair of corresponding values within each pair of corresponding fields. The result is a new fieldset.

```
result = fieldset_1 + fieldset_2
```

Macro Essentials - Fieldsets



Macro Essentials - Fieldsets



Macro Essentials - Fieldsets

- Operations and functions on fieldsets
  - Can also combine fieldsets with scalars:

    Z = X – 273.15

    Gives a fieldset where all values are 273.15 less than the original (Kelvin to Celcius)
  - Functions such as log:

    Z = log(X)

Macro Essentials - Fieldsets

- Operations and functions on fieldsets
  - Boolean operators such as > or <= produce, *for each point*, 0 when the comparison fails, or 1 if it succeeds:

    Z = X>0

  Gives a fieldset where all values are either 1 or 0

  – can be used as a mask to multiply by

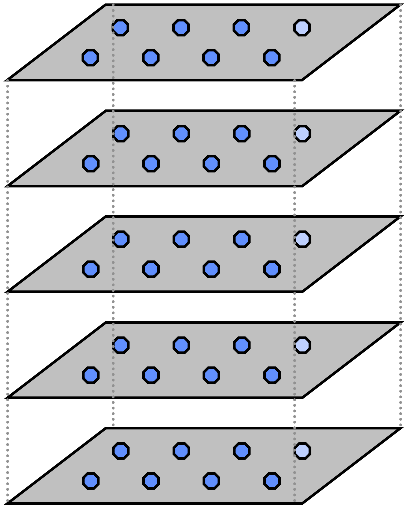
  – bitmap() can be used to invalidate values

  e.g.

```
t2m_masked = t2m * landseamask
t2m_masked = bitmap (t2m_masked, 0)
```

Macro Essentials - Fieldsets

- **suppose that fieldset ‘fs’ contains 5 fields:**
  - **`accumulate(fs)`**
    - ➔ returns a list of 5 numbers, each is the sum of all the values in that field
  - **`sum(fs)`**
    - ➔ returns a single field where each value is the sum of the 5 corresponding values in the input fields
  - **Many, many more – see the user guide**
    - ➔ e.g. **`mean()`**, **`maxvalue()`**, **`stdev()`**, **`coslat()`**



Macro Essentials - Fieldsets

- **Building up fieldsets**
  - `fieldset & fieldset , fieldset & nil`
  - merge several fieldsets. The output is a fieldset with as many fields as the sum of all fieldsets.

```
fs = nil
for d = 2006-01-01 to 2006-12-31 do
  x = retrieve(date : d, ...)
  fs = fs & x
end for
```

  - This is useful to build a fieldset inside a loop.

Macro Essentials - Fieldsets

- **Extracting fields from fieldsets**
  - ➔ fieldset [number]
  - ➔ fieldset [number,number]
  - ➔ fieldset [number,number,number]
- **Examples :**

```
y = x[2]        # copies field 2 of x into y
y = x[3,8]      # copies fields 3,4,5,6,7 and 8
y = x[1,20,4]   # copies fields 1, 5, 9, 13 and 17
```

Macro Essentials - Fieldsets

- **Writing Fieldsets as Text**
  - Easy to save in Geopoints format (see next slide)

```
for i = 1 to count (fields) do
    gpt = grib_to_geo (data : fields[i])
    write ('field_' & i & '.gpt', gpt)
end for
```

Tutorial Steps 5-7

- Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions
- **Steps 5-7 : Outputs other than on-screen**
- Step 8 : Macro run mode control
- Steps 9-10 : User Interfaces in Macro
- Step 11 : Macro in Batch
- Steps 12a,b,c : Using functions in Macro (libraries)
- Embedding FORTRAN and C in Macro

Macro Essentials – Loops, Tests & Functions

- The for, while, repeat, loop statements
  - See 'Metview Macro Syntax' handout
- The if/else, when, case statements
  - See 'Metview Macro Syntax' handout
- Function declarations
  - See 'Metview Macro Syntax' handout

Macro Essentials – Functions

- **Multiple versions**
  - Can declare multiple functions with the same name, but with different parameter number/types.

```
function fn_test ()
function fn_test (param1: string)
function fn_test (param1: number)
```

  - Correct one will be chosen according to the supplied parameters

Tutorial Step 8

- Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions
- Steps 5-7 : Outputs other than on-screen
- **Step 8 : Macro run mode control**
- Steps 9-10 : User Interfaces in Macro
- Step 11 : Macro in Batch
- Steps 12a,b,c : Using functions in Macro (libraries)
- Embedding FORTRAN and C in Macro

Tutorial Steps 9-10

- Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions
- Steps 5-7 : Outputs other than on-screen
- Step 8 : Macro run mode control
- **Steps 9-10 : User Interfaces in Macro**
- Step 11 : Macro in Batch
- Steps 12a,b,c : Using functions in Macro (libraries)
- Embedding FORTRAN and C in Macro

Tutorial Step 11

- Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions
- Steps 5-7 : Outputs other than on-screen
- Step 8 : Macro run mode control
- Steps 9-10 : User Interfaces in Macro
- **Step 11 : Macro in Batch**
- Steps 12a,b,c : Using functions in Macro (libraries)
- Embedding FORTRAN and C in Macro

Tutorial Step 12

- Steps 1-4 : Basic intro - input, basic contours, plot window, variables and functions
- Steps 5-7 : Outputs other than on-screen
- Step 8 : Macro run mode control
- Steps 9-10 : User Interfaces in Macro
- Step 11 : Macro in Batch
- **Steps 12a,b,c : Using functions in Macro (libraries)**
- Embedding FORTRAN and C in Macro

Fortran and C in Macro - Introduction

- Users can write their own Macro functions in Fortran or C/C++, extending the Macro language

- Used in tasks which cannot be achieved by macro functions. Or use existing FORTRAN/C code to save time.

- FORTRAN/C-Metview macro interfaces support input data of types GRIB, number, string and vector. BUFR, images and matrices are waiting implementation.

Fortran and C in Macro - Introduction

- **3 interfaces available:**
  - **Macro/Fortran Interface (MFI)**
    - Uses GRIB_API for fieldsets (GRIB 1 and 2)
  - **Macro/C Interface (MCI)**
    - Uses GRIB_API for fieldsets (GRIB 1 and 2)
  - **Legacy Macro/Fortran interface**
    - Uses GRIBEX for fieldsets (GRIB 1 only)
    - Deprecated: will disappear in the future – do not use!

Fortran/C in Macro – General Approach

- **Embed FORTRAN/C source code in the macro source file**
  - *Metview will automatically compile it at run-time*
- **OR**
- **Compile FORTRAN/C program separately or take an existing executable**

- **FORTRAN/C program is treated as another macro function**
- **E.g. specify some MARS retrievals to provide input fieldsets, use FORTRAN/C function to provide derived field(s);**

Fortran/C in Macro – Inline Code

- Embed the FORTRAN/C code in the macro program using the `inline` keyword

```
extern gradientb(f:fieldset) "fortran90" inline

   PROGRAM GRADIENTB
   ...

   INTEGER  grib_id, isize, istatus, i
   ...
   CALL mfi_get_fieldset( fieldset_in, icnt ) !-- GET FIRST ARGUMENT
   ...
    ...
end inline


# Retrieve the specific humidity
q = retrieve(
        date     :   -1,
        param    :   "q",
        ...)
```

Fortran/C in Macro – External Binary

- **<u>OR</u> specify location of the FORTRAN/C executable to the macro program**

basic (modified) - /home/graphics/cgi/metview/macro_tutorial_prep/

File Edit Search Preferences Shell Macro Windows Help

hics/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)
```

Fortran/C in Macro – General Approach

- Use suite of FORTRAN/C routines to get the input arguments, obtain GRIB_API handles for interrogation of GRIB data, save and set results, - these are the "interface routines" (`mfi_*`, `mci_*`).

- Schematically, the FORTRAN/C program dealing with a GRIB file is composed of
  - a section where input is read and output prepared
  - a loop where fields are loaded, expanded, validated, processed and saved
  - a section where output is set

Fortran in Macro – A Simple Example

- Advection of scalar field requires FORTRAN/C program to obtain the gradient of the field.
- Assume you will have a FORTRAN program called `gradientb` returning the gradient of a fieldset in two components (then advection is trivial). First concentrate on the writing of the macro program itself.
- Examine macro provided, which computes advection of specific humidity q at 700 hPa
- Examine FORTRAN source code provided, which computes gradient of a field

Fortran in Macro – A Simple Example

- Note interface routines, prefixed by "MFI" (e.g. `mfi_get_fieldset`, `mfi_load_one_grib`, `mfi_save_grib`). Most of the FORTRAN code is standard to process a GRIB fieldset.
- User routine `GRAD()` calculates gradient of input fieldset in two components:
  - saved separately and coded as wind components -
  - each can be accessed separately in the macro for the calculation of the advection.
- Two methods for making the program visible to macros:

Fortran in Macro – Embedding the FORTRAN Program

- **Method 1**: write the FORTRAN code inline – i.e., inside the macro code itself:

```
extern gradientb(f:fieldset) "fortran90" inline
PROGRAM GRADIENTB
CALL mfi_get_fieldset(fieldset_in, icount)
. . .
end inline
```

Fortran in Macro – Embedding the FORTRAN Program

- This can be written directly into the macro that will use it or else in a separate file.
- If written to a separate file, it can be accessed with the `include` macro command.
- If named correctly, it can be placed in the Macro folder of the System folder (`~uid/metview/System/Macros`) . In this case, the calling macro does not need any extra lines in order to use this function.

Fortran in Macro – Embedding the FORTRAN Program

- Method 2: compile and link the FORTRAN program separately. Then:
- a) inform the macro program where to find the FORTRAN executable:

```
extern gradientb(f:fieldset)
  "/home/xy/xyz/metview/fortran/gradientb"
```

- or b) place the executable in the Macro folder of the System folder (`~uid/metview/System/Macros`)
  - No need to specify this location to the macro

Fortran in Macro – Embedding the FORTRAN Program

- Finally, save the macro and execute to obtain the desired result.
- The procedure above is fairly general and with minor changes, can be adapted to other tasks just by replacing the processing routine.
- **NOTE:** in some cases, it may be a good idea to perform the GRIB handling within Macro, extract the values and coordinates as *vectors*, and pass these to the inline FORTRAN/C code instead – simpler inline code.

Macro Essentials - Variables

- Scope and Visibility
  - Variables inside functions are local
- Functions cannot see 'outside' variables

```
x = 9                 # cannot see y here

function func

      y = 10          # cannot see x here

end func

                      # cannot see y here
```


ECMWF

Macro Essentials - Variables

- **Scope and Visibility**
  - … unless a variable is defined to be 'global'

```
global g1 = 9              # cannot see y1 here
function func
     y1 = 10 + g1          # can see g1 here
end func
                           # cannot see y1 here
```

Macro Essentials - Variables

- **Scope and Visibility**
  - … a better solution is to pass a parameter
  - … that way, the function can be reused in other macros

```
x = 9

func(x)     # x is passed as a parameter

function func (t : number) #t adopts value of x

    y1 = 10 + t             # y1 = 10 + 9

end func
```

Macro Essentials - Variables

- **Destroying variables automatically**
  - When they go out of scope

```
function plot_a
   a = retrieve(...)
   plot(a)
end plot_a


# Main routine
plot_a()  # a is created and destroyed
```

Macro Essentials - Variables

- Destroying variables manually
  - Set to zero
  - (Variables can ‘hold’ lots of data, either in memory or in temporary files)

```
a = retrieve(...)
plot(a) # we have finished with 'a' now
a = 0
b = retrieve(...)
plot(b)
```

Macro Essentials - Geopoints

- Hold spatially irregular data
- ASCII format file

```
#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
```

Macro Essentials - Geopoints

- **Alternative format: XYV**

```
#GEO
#FORMAT XYV
PARAMETER = 2m Temperature
long      lat     value
#DATA
-5.35   36.15 300.9
32.98   34.58 301.6
21.65   41.97 299.4
```

Macro Essentials - Geopoints

- **Alternative format: XY_VECTOR**

```
#GEO
#FORMAT XY_VECTOR
lat  lon height date     time    u       v
#DATA
80   10   0    20030617 1200 -4.9001   -8.3126
80   5.5  0    20030617 1200 -5.6628   -7.7252
70   11   0    20030617 1200 -6.42549 -7.13829
```

Macro Essentials - Geopoints

- **Alternative format: POLAR_VECTOR**

```
#GEO
#FORMAT POLAR_VECTOR
lat     lon height  date   time speed direction
#DATA
50.97  6.05   0    20030614 1200  23    90
41.97 21.65   0    20030614 1200  4     330
35.85 14.48   0    20030614 1200  12    170
```

Macro Essentials - Geopoints

- 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)
```


ECMWF

Macro Essentials - Geopoints

- **Operations on geopoints**

  - `geo_gt_5 = geo_pts > 5`

    - → Means "create a new set of geopoints of 1 where input value is greater than 5, and 0 where it is not".

```
(3, 4, 5, 6, 7, 8)
 ↓  ↓  ↓  ↓  ↓  ↓
(0, 0, 0, 1, 1, 1)
```

Macro Essentials - Geopoints

- **Filtering geopoints**
  - ♦ `result = filter (geo_pts, geo_pts > 5)`
  - ♦ `result = filter (geo_pts, geo_gt_5)`

Equivalent

→ Means “extract from the first set of geopoints the points where the corresponding point in the second parameter is non-zero".

→ Means "create a new set of geopoints consisting only of those points whose value is greater than 5".

```
geo_pts   : (3, 4, 5, 6, 7, 8)
geo_gt_5  : (0, 0, 0, 1, 1, 1)
result    : (6, 7, 8)
```

Macro Essentials - Geopoints

- **Example of functions on geopoints**
  - `count (geopoints)`
    - ➔ Returns the number of points
  - `distance (geopoints, number, number)`
    - ➔ Returns the set of distances from the given location
  - `mean (geopoints)`
    - ➔ Returns the mean value of all the points

Macro Essentials - Geopoints

- **Combining Fieldsets And Point Data**
  - **Point data is stored in *geopoints* variables**
  - **Combination of geopoints and fieldsets 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

Macro Essentials – ASCII Tables

- ASCII Tables – columns of data in text files
  - E.g. CSV (Comma Separated Value)
  - Various parsing options for different formats
- 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
```

Macro Essentials – Vectors

- 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 initialised with `nil`:

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


v = nil # start from nil and append
v = v & |4.4, 5.5, 3.14| & |8, 9|


v = vector(10000) # pre-allocate space
v[1] = 4 # assign values to indexes
```

Macro Essentials - Vectors

- Assigning/replacing a range of values at once:

```
v = |10,20,30,40|

v[2] = |99,99| # v is now |10,99,99,40|
```

Macro Essentials - Vectors

- 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))
```

Macro Essentials - Vectors

- Accessing vector elements
- Indexes start at 1

```
v = |10,20,30,40|
a = v[1]           # a = 10
b = v[2,4]         # b = |20,30,40| (m to n)
c = v[1,4,2]       # c = |10,30| (step 2)
d = v[1,4,2,2]     # d = |10,20,30,40|
                   # (take 2 at each step)
```

Macro Essentials - Vectors

- The raw data in most file formats supported by Metview can be extracted into a vector:

```
vals = values(fieldset)

vals = values(netcdf)

vals = values(geopoints)

vals = values(table, 'column_A')

vals = values(odb, 'column_A')
```

Macro Essentials - Vectors

- Vectors honour missing values and will not include them in calculations
- For computations with many steps, vectors can be the most efficient way to do it
- Stored in memory, no intermediate files on disk (but greater memory usage!)
- Operations on lists of vectors:

```
a = [v1,v2] * [v3,v4]
# a is now [v1*v3, v2*v4]
```

Macro Essentials - Definitions

- 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"]
```

- Like a struct in 'C' or a dictionary in Python

Macro Essentials - Definitions

- 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"
     )
```

Macro Essentials - Definitions

```
param_def = ( param : "Z",
              type  : "FC",
              date  : -1,
              step  : 24 )

# retrieve as LL grid or not according to user
# choice
if (use_LL = "yes") then
  param_def.grid = [1.5,1.5]
end if

Z_ret = retrieve (param_def)
```

Macro Essentials - Definitions

```
common_input = ( levtype : "PL",
                 levelist : 850,
                 time : 12,
                 grid : [2.5,2.5],
                 type : "AN" )

Uan = retrieve ( common_input,
                 date  : -1,
                 param : "U" )

Van = retrieve ( common_input,
                 date  : -2,
                 param : "V" )
```

Macro Essentials – Data Input

- For GRIB files, `read()` reads the data into a **fieldset**
- For BUFR files, `read()` reads the data into an **observations** variable (usually convert to geopoints before using)
- For geopoints, `read()` reads the data into a **geopoints** variable
- For netCDF, `read()` reads the data into a **netcdf** variable
- For ODB, `read()` reads the data into an **odb** variable (Observational DataBase – see separate tutorial on the web)

Macro Essentials – Data Input

- For ASCII tables, `read_table()` reads the data into a **table** variable
- For other ASCII data, `read()` reads the data into a **list**, where each element is a string containing a line of the text file. Use string functions `parse()` and `substring()` to separate elements further.

Macro Essentials – Data Output

- **Use the `write()` function**
  - ➔ using filename, subsequent calls overwrite
  - ➔ using file handler, subsequent calls append
- **Can also use `append()`**
- **Automatic file format**

fieldset	**->**	**GRIB file**
observations	**->**	**BUFR file**
geopoints	**->**	**geopoints file**
netcdf	**->**	**netcdf file**
string	**->**	**ASCII file (custom formats)**

ECMWF

Macro Documentation

- **Metview 4 documentation here:**
  - https://software.ecmwf.int/metview/
  - Documentation / User Guide , FAQ
  - Material from this course will soon appear there!
- **Some information still only for Metview 3**
  - PDF available
  - Will be migrated to the Metview 4 pages
- **Ask!**
  - **metview@ecmwf.int**