Metview Macro – A powerful meteorological batch language

STEPHAN SIEMEN, FERNANDO II, IAIN RUSSELL

PARALLEL TO its graphical user interface (GUI), Metview offers a high-level meteorological scripting language to describe the retrieval, processing and visualisation of meteorological data. A scripting language was part of the first design specification of Metview and has always been an integral part. A language is the best 'user interface' to describe complex sequences of actions, particularly if the flow of action is conditional; this includes the expression of mathematical formulae and other forms of data manipulation.

The philosophy behind Metview Macro is that it is easy to get started, but powerful and flexible enough for advanced needs. The user does not need to declare variables – their types are assigned automatically according to the data they store. The language also supports flow control, user-defined functions, I/O and error control. Functions can be called from other macros; this feature enables users to build their own libraries of Macro functions that can then be used by a larger user group.

This article outlines the key features and versatility of Metview Macro.

Overview of Metview Macro

The Metview Macro language provides an easy, powerful and comprehensive way for an analyst or researcher to manipulate and display meteorological data. It extends the use of Metview into an operational environment as it enables a user to write complex scripts that may be run with any desired periodicity. Metview has few runtime dependencies, thus making it ideal for running in an operational environment. This and the integrated access to MARS (Meteorological Archival and Retrieval System) archives makes Metview the ideal tool to run operational tasks for internal and external users of ECMWF's operational data. As described later, users can create and customise their visualisation interactively and then save it as a macro that can be executed in batch.

There are various ways to run a macro. It can be executed through the right-click menu of its icon on the GUI or executed within the Macro editor. Outside the GUI a script can be executed through the command *metview -b* followed by the name of the macro file and the macro's parameters. The latter is the way to run scripts in batch operationally, for example within SMS or as cron jobs.

To demonstrate the power of Metview Macro, consider the following example in Listing 1. To calculate and plot the difference between some point observations and an analysis field needs just four lines of code. The user does not need to worry about the conversion between formats and the interpolation of data values. If required, Metview directs such calculations to its support packages, such as Emoslib and Grib_API. The resulting plot is shown in Figure 1.

```
t2m_points = read("temperature_towns.gpt")
t2m_analysis = read("t2m_an.grib")
t2m_diff = t2m_points - t2m_analysis
plot(t2m_diff)
```

Listing 1 Example Macro visualising the difference between point observations and analysis field. Here 'grib' and 'gpt' are used as file extensions for the GRIB and Metview's Geopoints data formats respectively.

Key Macro features

Macro syntax and functions

Metview Macro offers all the facilities of a modern scripting language, for instance loops, if and case statements, and functions. As well as simple variable types (e.g. numbers, strings and lists), Macro has native variable types for commonly used meteorological data formats such as GRIB, BUFR, netCDF and ODB.

When used with the GUI, most interaction with Metview is performed via a rich set of icons that enable data retrieval, manipulation and plotting. For every icon, a corresponding Macro function exists; parameters are supplied via a set of parameter-value pairs whose syntax is based on the MARS language. Executing a *MARS Retrieval* icon and running the function *retrieve()* result in the same outcome.

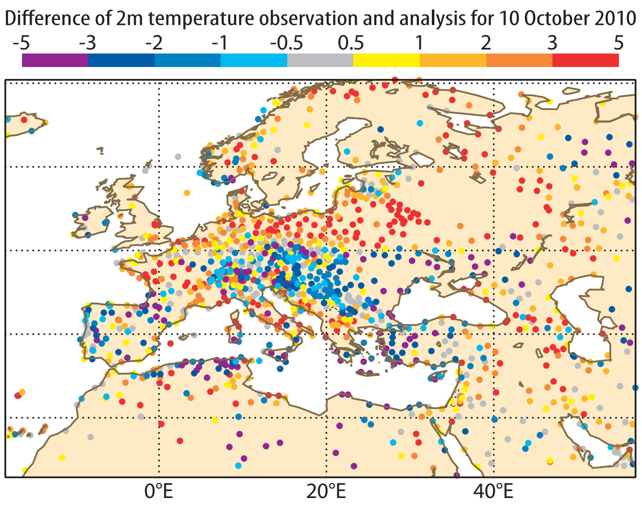

Difference of 2m temperature observation and analysis for 10 October 2010
-5
-3
-2
-1
-0.5
0.5
1
2
3
5
0°E
20°E
40°E

Figure 1 Plot resulting from Macro code in Listing 1. Additional to the listing, the differences between analysis and observation are coloured according to whether they are positive or negative.

In addition Metview Macro contains a large set of built-in functions that can be used for meteorological data processing. Macro also contains functionality to cache results, reducing the need to recalculate parts of the computation if the input data does not change. This can improve performance on repeated execution over the same data set. The time period of how long data is cached can be set in the Metview preferences.

Writing Macro code

The Metview user interface provides a Macro editor which not only offers syntax highlighting, but also the running and debugging of scripts interactively. While the editor in Metview 3 required the help of an additional external editor to offer extended functionality, the new Macro editor in Metview 4 is more powerful and supports all features expected from a modern code editor.

To allow users to quickly write Macro code, the Metview GUI offers features to generate code from the interactive user interface in two ways:

- Users who have generated their desired plot in the interactive plot window can use the 'Generate Macro' button (red punch card) to generate the corresponding macro code. In some cases of complex displays the resulting script may need some adjustments.
- Thanks to the direct mapping between desktop icons and macro icon-functions the user can drop an icon inside a macro editor. The result is the textual translation of the icon contents into a macro icon-function.

Figure 2 gives a snapshot of the Metview 4 macro editor.

Inlining Fortran and C/C++

The Macro language can be extended by the user with Fortran and C/C++ code. This ability extends immensely the scope of the macro language and enables the programmer to make efficient use of existing Fortran and C codes.

These programs are used in tasks that cannot be easily achieved by means of a Macro language function or combination of functions. Also, suitable code might already exist and the writing and testing of the same task in Macro language would simply consume precious time.

Building a Macro User Interface

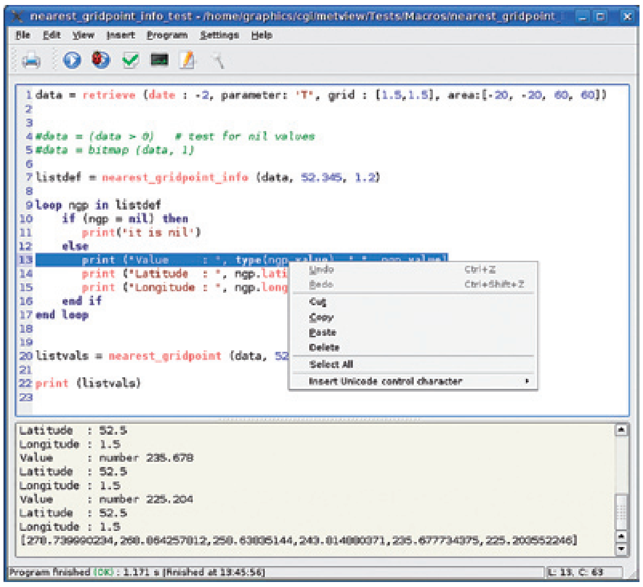
Metview allows developers to build their own user interfaces to their scripts. These are useful to provide generality, meaning that the same Macro program used for a given task will be able to accept a variety of input parameters which will be provided via an icon editor window-like interface (in batch, parameters may be supplied on the command line). Listing 2 shows some sample code, and Figure 3 the resulting user interface.

```
c = slider (name    : “days”,
            min     : 1,
            max     : 10,
            default : 5)
```

Listing 2 Macro code to generate a slider in a customised user interface.

Defining outputs

One major highlight of Metview 4 is the introduction of many more graphical output formats, due to the use of Magics++. Scripts run on the interactive GUI can make use of Metview 4's new interactive plot window or plot in new formats such as PDF, EPS, SVG and KML. These new options required a change in the way graphical output formats are defined in comparison to previous Metview versions. However, Metview 4 largely retains backwards compatibility with Metview 3's macro functions to export plots to files (`output()` and `setoutput()` functions). For more details see Box A.

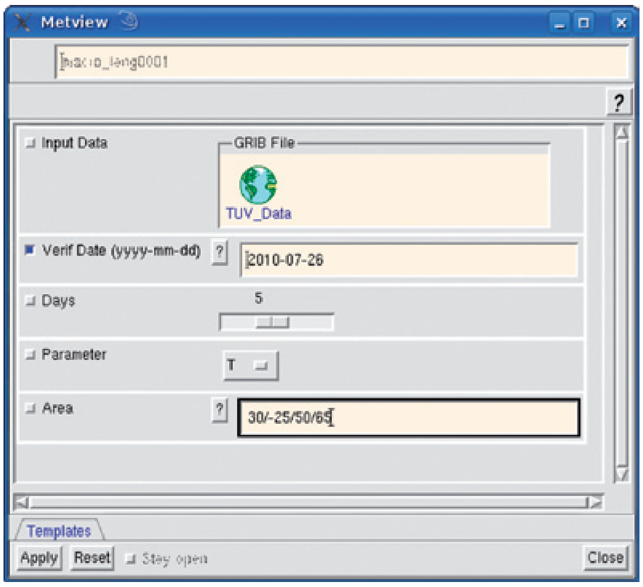

nearest_gridpoint_info_test - /home/graphics/cgi/metview/Tests/Macros/nearest_gridpoint
File Edit View Insert Program Settings Help
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut
Copy
Paste
Delete
Select All
Insert Unicode control character

```
data = retrieve (date : -2, parameter: 'T', grid : [1.5,1.5], area:[-20, -20, 60, 60])


#data = (data > 0)    # test for nil values
#data = bitmap (data, 1)

listdef = nearest_gridpoint_info (data, 52.345, 1.2)

loop ngp in listdef
    if (ngp = nil) then
        print('it is nil')
    else
        print ('Value      : ', type(ngp.value), ' ', ngp.value)
        print ('Latitude   : ', ngp.lati
        print ('Longitude : ', ngp.long
    end if
end loop


listvals = nearest_gridpoint (data, 52

print (listvals)
```

```
Latitude  : 52.5
Longitude : 1.5
Value     : number 235.678
Latitude  : 52.5
Longitude : 1.5
Value     : number 225.204
Latitude  : 52.5
Longitude : 1.5
[278.739990234,268.064257812,258.63806144,243.014080371,235.677734375,225.203552246]
```

```
Program finished (OK) : 1.171 s [finished at 13:45:56]     L: 13, C: 63
```

Figure 2 Snapshot of the Metview 4 Macro editor, showing syntax highlighting and debug output (bottom).



Input Data — GRIB File: TUV_Data

Verif Date (yyyy-mm-dd): 2010-07-26

Days: 5

Parameter: T

Area: 30/-25/50/65

Figure 3 Snapshot of a user-defined user interface generated by a macro (see Listing 2 for a partial code listing).

Extensions to defining output formats in Metview 4 — A

The following syntax is used to define a PostScript format in Metview 3:

```
ps = output(format    : “postscript”,
            file_name : “plotfile.ps” )
```

Metview 4 has a new set of functions, for example:

```
ps  = ps_output ( output_name : “plotfile” )
kml = kml_output( output_name : “plotfile” )
svg = svg_output( output_name : “plotfile” )
```

Both syntaxes are valid in Metview 4. An extension is that setoutput() can take a list of several output definitions – in this case, Metview will generate its plot for all outputs. This can be significantly faster than running a macro multiple times (once for each output format). For instance:

```
setoutput([ps, kml, svg])
```

An alternative way to select which output formats will be generated is to give them directly to the `plot()` command:

```
plot([ps, kml, svg],…)
```

Any output formats given to the `plot()` command will override those given to `setoutput()`.

Optionally, within a macro the user can define specific output formats depending on the environment in which it is run:

```
# sets output destination according to
run-mode
if mode = “batch” then
     setoutput(ps_file)
else if mode = “visualise” then
     setoutput(screen)
end if
```

Changes introduced in version 4.0

The new options to define outputs in Metview 4 are only the beginning of improvements to come for the Macro language (see Figure 4). Parameters are being cleaned up to remove those specific to MAGICS 6 and new ones added to access features in Magics++. Many issues in version 3 regarding performance and temporary files are being improved.

From version 4.0 onwards Metview's macro language handles missing values in its data in a more consistent and useful way. Previously, functions such as `integrate()` returned a 'missing value indicator' if all its input values were missing. This was not easy to test, and computations could use the result incorrectly without realising it. From now on all such functions return a `nil` variable when their inputs are invalid. Macros that do not test for this condition will fail if they try to use a `nil` variable in a computation. The following is an example code extract:

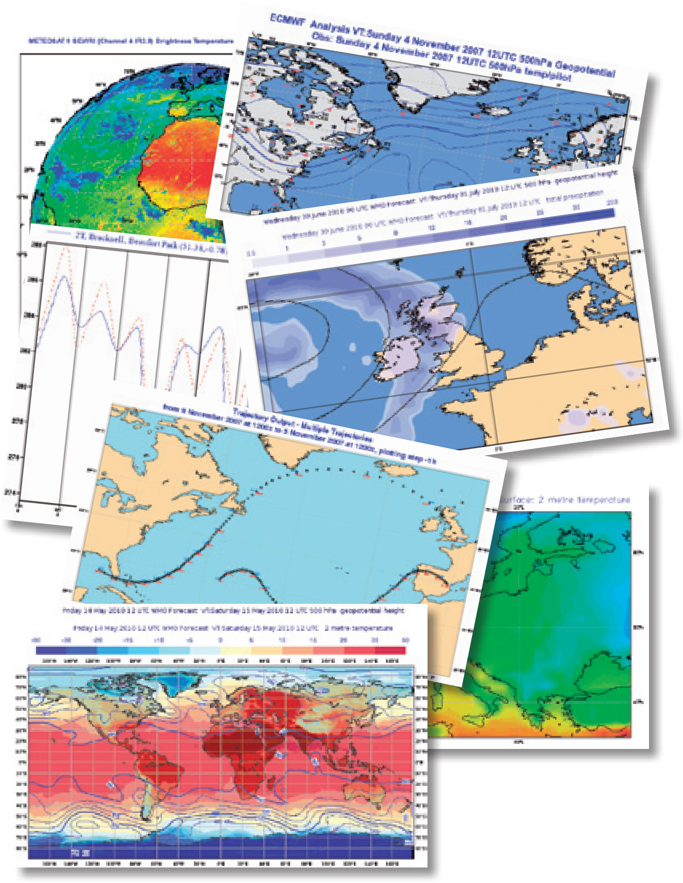


Figure 4 The user has all the functionality of the interactive graphical user interface plus a lot more.

```
a = integrate(data)
if (a = nil) then
    fail (‘Integration failed’)
end if
```

The much improved installation scripts of Metview 4 will also allow for a batch only installation to reduce dependencies resulting from the GUI. This version is aimed for usage on operational servers.

Documentation, training and future developments

For users interested to find out more about Metview Macro there is full documentation of the language and available functions at:

http://www.ecmwf.int/publications/manuals/metview/documentation.html

The page also contains the training material from the annual Metview training course held at ECMWF, including tutorials and presentations.

The evolution of Metview's macro language will continue beyond the release of Metview 4.0. More built-in functions will be provided to support ECMWF tasks of observation handling and model developments. Metview's GUI and Macro editor will be extended to provide facilities to search for available macro functions and libraries.

User requests will of course continue to play an important role in improving the performance and reliability of Metview.