ecCodes – using grib_filter

Computer User Training Course 2019

Paul Dando

User Support

advisory@ecmwf.int

ECMWF

grib_filter – introduction

- ecCodes advanced command-line tool
- Iterates over all the messages in the input
- Applies a set of user defined rules to each message
- The rules are formed using a macro language ecCodes provides
- Note that the macro language does not have the capabilities of a full-blown programming language
- Syntax of macro language is the same for both grib_filter and bufr_filter

grib_filter – introduction

- Access data inside a message through keys
- Print contents of a message
- Set values inside a message
- Use control structures (`if`, `switch`)
- Write a message to disk

grib_filter – usage

```
grib_filter [-o out_file] rules_file in_file1 in_file2 …
```

- Each field from the input files is processed and the rules contained in the rules_file are applied to it
- A GRIB message is written to an output file only if a write instruction is applied to it
- Each instruction in the rules_file must end with a semicolon "`;`"
- Syntax errors in the rules_file are reported with their line number

- Always put `-o out_file` before the other options !

- Alternatively, read the rules from standard input:

```
cat rules_file | grib_filter - in_file1 in_file2 …
echo 'print "Hello";' | grib_filter - in_file1 in_file2 …
```

Rules syntax – print statement

- `print “some text”; # this is a comment`
- `print “some text [key]”;`
  - Print to the standard output
  - Retrieve the value of the keys in squared brackets.
  - If a key is not found in the message then the value of `[key]` will be displayed as “undef”

`[key]`	-> native type
`[key:i]`	-> integer
`[key:s]`	-> string
`[key:d]`	-> double
`[key!c%F’S’]`	-> arrays: c->columns F->format (C style) S->separator

- `print (“filename”) “some text [key]”;`

Example 1 – using print

```
# A simple print
print "ed = [edition] centre is [centre:s] = [centre:i]";
```

```
> grib_filter rule.filter x.grib1
ed = 1 centre is ecmf = 98
```

Example 2 – formatted print

```
# one column 3 decimal digits
print "[distinctLatitudes!1%.3f]";
```

```
> grib_filter rule.filter x.grib1
-90.000
-88.500
-87.000
-85.500
…
```

Example 3 – print with separator

```
# three columns 5 decimal digits comma separated
print “[latLonValues!3%.5f’,’]”;
```

```
> grib_filter rule.filter x.grib1
90.00000,0.00000,1.00000,
90.00000,1.50000,1.00000,
90.00000,3.00000,1.00000,
…
```

Rules syntax – write statement

- `write;`
  - Writes the current message to the output file defined in the command line with the option `-o`

    ```
    grib_filter -o outfile rules_file grib_file
    ```

  - If the `-o` option is not specified, the default value “`filter.out`” is used

- `write “filename_[key]”;`
  - Writes the current message to the file “`filename_[key]`” where the key in square brackets is replaced with its value retrieved from the message
  - If two messages have different values for `[key]` they are also written to different files

Example 4 – write statement

```
# Creating multiple files
write "[centre]_[dataDate]_[step].grib[edition]";
```

```
> grib_filter rule.filter x.grib1
> ls
ecmf_20080213_0.grib1
ecmf_20080213_6.grib1
ecmf_20080213_12.grib1
ecmf_20080213_24.grib1
```

Rules syntax – append statement

- `append;`
  - Appends the current message to the output file defined in the command line with the option `-o`

    ```
    grib_filter -o outfile rules_file grib_file
    ```

  - If the `-o` option is not specified, the default value “`filter.out`” is used

- `append “filename_[key]”;`
  - Appends the current message to the file “`filename_[key]`” where the key in square brackets is replaced with its value retrieved from the message
  - The file is created if it does not exist
  - If two messages have different values for `[key]` they are appended to different files

Example 5 – append statement

```
append;
```

```
> grib_count out.grib
> 1
>
> grib_filter -o out.grib rule.filter in.grib
>
> grib_count out.grib
> 2
```

Rules syntax – setting keys

- `set key1 = key2 ;      # set key1 to the value of key2`
- `set key = {val1,val2,val3,val4} ; # set an array key`
- `set key = “string” ;              # set key to a string`
- `set key = expression ;     # set key to an expression`
- `set key = MISSING ;     # set value of key to missing`
- expression operators :

`==`	equal to
`!=`	not equal to
`is`	equals to for strings
`||`	or
`&&`	and
`!`	not
`* / + -`	arithmetic operators
`( )`	

Example 6 – setting a key

```
set edition = 2;

write "[file][edition]";
```

```
> grib_filter rule.filter x.grib
> ls
x.grib
x.grib2
```

Example 7 – setting an array key

```
set values = {12.2,14.8,13.7,72.3};
print “values = { [values] }”;
write “[file].[edition]”;
```

```
> grib_filter rule.filter x.grib
values = {  12.2 14.8 13.7 72.3 }
```

Rules syntax – transient keys

- `transient key1 = key2;`
  - Defines the new key1 and assigns to it the value of key2
- `transient key1 = “string”;`
- `transient key1 = expression ;`
- expression operators:

`==`	equal to
`!=`	not equal to
`is`	equals to for strings
`||`	or
`&&`	and
`!`	not
`* / + -`	arithmetic operators
`( )`	

Example 8 – transient keys

```
transient mystep = step + 24;
print “step = [step] mystep = [mystep]”;
```

```
> grib_filter rule.filter x.grib
step = 24 mystep = 48
```

Practicals

- To get the material for these practicals:

```
cd $SCRATCH/grib_tools/filter
```

Reminder: If you need to get the material for the GRIB tools practicals:

- Make a copy of the practicals directory in your $SCRATCH

```
tar -xvf /home/ectrain/trx/ecCodes/grib_tools.tar
```

- This will create a directory in your $SCRATCH containing the GRIB data files for this morning's practicals

1. Run grib_filter with the rules files 'print.filter', 'write.filter', 'transient.filter' on 'tigge.grib'.
2. Comment/uncomment the instructions one by one to see the different behaviours.

Rules syntax – if statement

- `if ( expression ) { instructions }`
- `if ( expression ) { instructions }`
  `else { instructions }`

There is no 'else if' - you have to create a new 'if' block

- Expression operators:

`==`	equal to
`!=`	not equal to
`<`	less than (less than or equal to: `<=`)
`>`	greater than (greater than or equal to: `>=`)
`is`	equals to for strings ("`! (key is value)`" for "not is")
`||`	or
`&&`	and
`!`	not
`* / + -`	arithmetic operators
`( )`	

Example 9 – if statement

```
if (localDefinitionNumber == 1) {
   set edition = 2;
   write;
}
```

```
> grib_filter -o out.grib2 rule.filter x.grib1
> ls
out.grib2
```

Rules syntax – switch statement

- Alternate version of an ‘if-else’ statement
- More convenient to use when you have code that needs to choose a path from many to follow

```
switch (var) {
       case val1:
              # set of actions
              ...
       case val2:

              # set of actions
              ...
       default:
              # default block of actions
}
```

default: case is mandatory even if empty

Example 10 – switch statement

```
print "processing [paramId] [shortName] [stepType]";
switch (shortName) {
    case "tp" :
         set stepType="accum";
    case "sp" :
         set typeOfLevel="surface";
    default:
         print "Unexpected parameter";
}
write;
```

Example 11

```
if (centre is "lfpw" &&
       (indicatorOfParameter == 6 ||
        indicatorOfParameter == 11 ||
        indicatorOfParameter == 8) )
{
    if (step!=0) {
      set typeOfGeneratingProcess=0;
      set typeOfProcessedData=0;
    } else {
    # Other steps
      set typeOfProcessedData=1;

 …
```

```
…
  switch (typeOfLevel) {
    case "hybrid":
      set changeDecimalPrecision=1;
    case "surface":
      set changeDecimalPrecision=2;
    case "isobaricInhPa":
      if (level > 300) {
        print "level > 300";
        set level = level*2 + 15;
      } # end if (level > 300)
    default:
      print "Unknown level type!";
    } # end switch (typeOfLevel)
  } # end if (step!=0)
  write;
} # end main if
```

Rules syntax – assert statement

- **`assert(condition);`**

- If the condition evaluates to false then the filter will abort

```
# This filter should be run on GRIB edition 1 only;
# abort otherwise

assert (edition == 1) ;

...
```

```
> grib_filter -o out.grib2 rule.filter x.grib2
ECCODES ERROR   :  Assertion failure:
binop(access('edition=2'),long(2))
```

Advanced grib_filter practicals

1. Change the date to 20170301 and the step to step+48 in the file ‘tigge.grib’ only for the data produced by ECMWF. Write all messages to a file called ‘question1.grib’.
2. Set the values of the first message (remember the count key !) in the file ‘tigge.grib’ to 1.2, 3.4, 5.6, 3.7 and step to 72. Write only this message to the file ‘question2.grib’
    1. Check the values coded with grib_get_data or grib_dump.
3. Append to ‘question2.grib’ all the messages containing the same parameter of the other centres that are not encoded using a reduced Gaussian grid, setting the step to 72.

You can check if your filter is correct by comparing your output GRIB file with the sample in sample_outputs/, e.g.:

```
> grib_filter -o question1.grib question1.filter tigge.grib
> grib_compare question1.grib sample_outputs/question1.grib
```

Practicals (Extra)

4. Split ‘tigge.grib’ into several files, one for each centre, containing only surface parameters and parameters that are at level 10 of height above ground.
   - For the surface parameters, set changeDecimalPrecision to 2,
   - For the height above ground parameters set changeDecimalPrecision to 3.

   Print information messages for each case, such as:

```
Centre ammc parameter v not written
Centre ammc parameter 10u written to question4-ammc.out
```

5. Merge the messages from the previously split GRIB files into a single file
   - Write only messages encoded in a regular lat-long grid, and exclude messages where the parameters are 10u or 10v.