ecCodes

GRIB Fortran 90 - Python APIs

Part 1

Dominique Lucas and Xavi Abellan

Dominique.Lucas@ecmwf.int Xavier.Abellan@ecmwf.int

For GRIB data, the only difference between the GRIB API and ecCodes packages is …

Content

Introduction

- Do you really need to write a program to decode GRIB or BUFR data?
  - Before converting/writing a (Fortran, C or Python) code, you should evaluate whether you could use some tools included with ecCodes instead
  - Writing programs will remain necessary:
    - Encoding/decoding GRIB messages within a meteorological model or within some post-processing (e.g. MAGICS code).
  - Writing a program will usually be more efficient than trying to achieve the same with tools/scripts.

Usage at ECMWF

- The ecCodes library is available on all ECMWF platforms.
- One single library for single and double precision. Within the library, everything is done in double precision. In Fortran 90, the ecCodes will return/use the precision of the data variables defined in your program.
- Three user interfaces supported: Fortran (90), C and Python
  - Fortran 90 interface: 'use eccodes'
  - Python interface: 'from eccodes import *'
- (At ECMWF) Two environment variables ECCODES_INCLUDE and ECCODES_LIB are defined to ease the usage of ecCodes.
- The GRIB API module should be unloaded before loading ecCodes:

```
$ module swap grib_api eccodes
```

Usage at ECMWF

- Compilation of ecCodes at ECMWF:

```
ecgate$  gfortran myprogram.f90 $ECCODES_INCLUDE $ECCODES_LIB
cca/ccb$ ftn myprogram.f90
```

- Changing version of ecCodes at ECMWF:

```
$ module switch eccodes/<version>
```

- See change history…

https://software.ecmwf.int/wiki/display/ECC/Latest+news

Names of ecCodes routines to handle GRIB data

- All routines in ecCodes start with ‘codes_’, e.g.

```
call codes_open_file(infile,’data.grib’)
```

- Most routines apply both fto GRIB and BUFR data, e.g.

```
call codes_get(igrib,’date’)
call codes_get(ibufr,’typicalDate’)
```

  - Some routines are different for GRB and BUFR, e.g.

```
call codes_grib_new_from_file(ifile, igrib)
call codes_bufr_new_from_file(ifile, ibufr)
call codes_any_new_from_file(ifile, iany)                      NEW!
call codes_new_from_file(ifile, product_kind, diany)     NEW!
```

- The GRIB API routine names starting with ‘grib_’ also exist in ecCodes, e.g.

```
call grib_open_file(infile,’data.grib’)
```

General framework

- A (Fortran/C/Python) code will include the following steps:
  - – Open one or more GRIB files (to read or write)
    - You cannot use the standard Fortran calls to open or close a GRIB file.
  - – Calls to load one or more GRIB messages into memory
    - These subroutines will return a unique grib identifier which you can use to manipulate the loaded GRIB messages
  - – Calls to encode/decode the loaded GRIB messages
    - You can only encode/decode loaded GRIB messages
    - You should only encode/decode what you need (not the full message)
  - – Calls to write one or more GRIB messages into a file (encoding only)
  - – Release the loaded GRIB messages
  - – Close the opened GRIB files
- This framework is also valid for BUFR data.

Particulars of the F90 ecCodes interface

- All routines have an optional argument for error handling:

```
subroutine grib_new_from_samples(igrib, samplename, status)
    integer, intent(out)            :: igrib
    character(len=*), intent(in)    :: samplename
    integer,optional, intent(out)   :: status
```

- If status is not present and an error occurs, the program stops and returns the error code to the shell.

Particulars of the F90 ecCodes interface

- Use status to handle error yourself, e.g. necessary for MPI parallel codes)

```
call grib_new_from_samples(igrib, samplename, status)
  if (status /= 0) then
     call grib_get_error_string(status,err_msg)
    print*,'ECCODES Error: ',trim(err_msg),' (err=',status,')'
    call mpi_finalize(ierr)
    stop
 end if
```

Input arguments
Output arguments

- The exit codes and their meanings are available under:

https://software.ecmwf.int/wiki/display/GRIB/Error+codes

Loading/Releasing a GRIB message (1/2)

- It is absolutely necessary to load a message because the ecCodes can only encode/decode loaded GRIB messages.
- 3 main subroutines to load a GRIB message

Input arguments

Output arguments

```
call grib_new_from_file(ifile, igrib)
```

Loads a GRIB message from a file already opened with grib_open_file (use grib_close_file to close this file)

```
call grib_new_from_samples(igrib, “GRIB1” )
```

Name of an existing GRIB sample

Loads a GRIB message from a sample. Used for encoding. See further …

```
call grib_new_from_index(indexid, igrib)
```

Loads a GRIB message from an index. This index will first have to be built. See further ...

Loading/releasing a GRIB message (2/2)

- These 3 ‘loading’ subroutines will return a unique grib identifier (*igrib*). You will manipulate the loaded GRIB message through this identifier.
- You do not have access to the buffer containing the loaded GRIB message. This buffer is internal to the ecCodes library.
- The buffer occupied by any GRIB message is kept in memory.
- Therefore, the routine grib_release(*igrib*) should always be used to free the buffer containing a loaded GRIB message.

Input arguments

Output arguments

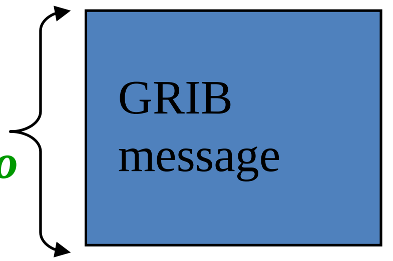
Input arguments
Output arguments

Example – Load from file

```
PROGRAM load_message
 USE eccodes
 implicit none

 INTEGER                          :: rfile, igrib
 CHARACTER(LEN=256), PARAMETER    :: input_file='input.grb'
 CHARACTER(LEN=10), PARAMETER     :: open_mode='r'

 !
 ! Open GRIB data file for reading.
 !
 call grib_open_file(rfile, input_file, open_mode)

 call grib_new_from_file(rfile, igrib)


 call grib_release(igrib)
 call grib_close_file (rfile)
 END PROGRAM load_message
```

'r' to read, 'w' to write, 'a' to append (C naming convention)

Unique link to the buffer loaded in memory. Calls to grib_get/grib_set subroutines are necessary to access and decode/encode this message



Decoding a loaded GRIB message

- The idea is to decode as little as possible! You will never decode the whole loaded GRIB message.

(use ‘grib_dump –D <grib_file>’ to see how many ecCodes keys there are!)

- One main subroutine to decode:

Input arguments

Output arguments

```
grib_get(igrib, keyname, keyvalues, status)

     integer, intent(in)                         ::  igrib
     character(len=*), intent(in)                :: keyname
     <type>,[dimension(:),] intent(out)  ::  keyvalues
     integer, optional, intent(out)              :: status
```

Where <type> is integer or single/double real precision or string

Decoding a GRIB message – helper routines

Input arguments

Output arguments

- Get the size of an [array] key:

```
grib_get_size(igrib, keyname, size, status)
```

- Dump the content of a grib_message:

```
grib_dump(igrib, status)
```

- Check if a key is defined (exists) or not:

```
grib_is_defined(igrib, keyname, exists, status)
```

- Check if the value for a key is missing or not:

```
grib_is_missing(igrib, keyname, missing, status)
```

- Count messages in file:

```
grib_count_in_file(ifile, count, status)
```

Example – grib_get

Input arguments
Output arguments

```
! Load all the GRIB messages contained in file.grib1
call grib_open_file(ifile, 'file.grib1','r')
n=1
call  grib_new_from_file(ifile,igrib(n), iret)
LOOP: do while (iret /= GRIB_END_OF_FILE)
     n=n+1; call grib_new_from_file(ifile,igrib(n), iret)
end do LOOP
! Decode/encode data from the loaded message
read*, indx                              ! Choose one loaded GRIB message to decode
call grib_get( igrib(indx) , “dataDate”, date)
call grib_get(igrib(indx), “typeOfLevel”, typeOfLevel)
call grib_get(igrib(indx), “level”, level)
call grib_get_size(igrib(indx), “values”, nb_values); allocate(values(nb_values))
call grib_get(igrib(indx), “values”, values)
print*, date, levelType, level, values(1), values(nb_values)
! Release
do i=1,n
     call grib_release(igrib(i))
end do
deallocate(values)
call grib_close_file(ifile)
```

Loop on all the messages in a file. A new grib message is loaded from file. igrib(n) is the grib id to be used in subsequent calls

Values is declared as real, dimension(:), allocatable:: values

ecCodes can do more …

- The idea is to provide a set of high-level keys or subroutines to derive/compute extra information from a loaded GRIB message
- For instance:
  - keys (READ-ONLY) to return average, min, max of values, distinct latitudes or longitudes …
  - Subroutines to compute the latitude, longitude and values:

    ```
    call grib_get_data(igrib,lats,lons,values,status)
    ```

  - Subroutines to extract values closest to given geographical points:

    ```
    call grib_find_nearest(igrib, is_lsm, inlat, inlon, outlat, outlon, value, distance, index, status)
    ```

  - Subroutine to extract values from a list of indexes:

    ```
    call grib_get_element(igrib, key, index, value, status)
    ```

Input arguments
Output arguments

How to migrate from GRIBEX to ecCodes?

- GRIBEX is no longer available in the default environment at ECMWF. It is no longer supported.
- If you still have GRIBEX codes, the essential task is to find the corresponding ecCodes key names to the elements in the GRIBEX ksec0(*), …, ksec4(*), psec2(*),…psec4(*) header sections. See conversion tables under:

  https://software.ecmwf.int/wiki/display/GRIB/GRIBEX+keys

- Try to use the “recommended” keys, i.e. keys that are valid for both GRIB-1 and GRIB-2 (for instance “dataDate” instead of “YearOfCentury”, “month”, “day”). See edition independent keys under:

  http://apps.ecmwf.int/codes/grib/format/edition-independent/

How to migrate from GRIB API to ecCodes?

- There should be nothing to do to your GRIB API codes.
- You could replace the calls to grib_ routines by calls to codes_ routines.
- Compile your GRIB API codes with the ecCodes libraries, e.g.:

```
ecgate $ module swap grib_api  eccodes
ecgate $  gfortran myprogram.f90 $ECCODES_INCLUDE $ECCODES_LIB
ecgate $ ./a.out
```

- Compare the results with the ones you obtain with the GRIB API.
- Note that ecCodes has not yet been fully released.

Python API - Specifics

- To import, use:

```
import gribapi          from gribapi import *
import eccodes          from eccodes import *
```

- module load eccodes.
  - If grib_api is loaded, then you should module swap grib_api eccodes
- Low level, procedural
- Provides almost 1 to 1 mappings to the C API functions
- Uses the NumPy module to efficiently handle data values

Python API – Loading/Releasing a GRIB message

gid = ***grib_new_from_file*** (file, headers_only=False) Returns a handle to a GRIB message in a file. Requires the input file to be a Python file object. The use of the headers_only option is not recommended at the moment.	***codes_grib_new_from_file*** ***codes_any_new_from_file*** ***codes_new_from_file*** (file, **product_kind**, headers_only) - CODES_PRODUCT_GRIB - CODES_PRODUCT_BUFR - CODES_PRODUCT_ANY
gid = ***grib_new_from_samples*** (samplename) Returns a handle to a message contained in the samples directory	***codes_new_from_samples***
gid = ***grib_new_from_message*** (message) Returns a handle to a message in memory	***codes_new_from_message***
grib_release (gid) Releases the handle	***codes_release***

Python API – Decoding

value = ***grib_get*** (gid, key, type=None) Returns the value of the requested key in the message gid is pointing to in its native format. Alternatively, one could choose what format to return the value in (int, str or float) by using the type keyword.	***codes_get***
values = ***grib_get_array*** (gid, key, type=None) Returns the contents of an array key as a NumPy ndarray or Python array. type can only be int or float.	***codes_get_array***
values = ***grib_get_values*** (gid) Gets data values as 1D array	***codes_get_values***
On error, a ***GribInternalError*** exception (which wraps errors coming from the C API) is thrown	***CodesInternalError***

Python API – Utilities

[outlat, outlon, value, distance, index] = ***grib_find_nearest*** (gid, inlat, inlon, is_lsm=False, npoints=1) Find the nearest point for a given lat/lon With npoints=4 it returns a list of the 4 nearest points	***codes_grib_find_nearest***
iter_id = ***grib_iterator_new*** (gid,mode)	***codes_grib_iterator_new***
[lat,lon,value] = ***grib_iterator_next*** (iterid)	***codes_grib_iterator_next***
grib_iterator_delete (iter_id)	***codes_grib_iterator_delete***

References

- GRIB-1, GRIB-2:

  http://www.wmo.int/pages/prog/www/WMOCodes.html

- ecCodes:

  https://software.ecmwf.int/wiki/display/ECC/ecCodes+Home

- ecCodes Fortran, C or Python interfaces to GRIB data (currently on the GRIB API wiki):

  f90: https://software.ecmwf.int/wiki/display/GRIB/Fortran+package+grib_api

  C: https://software.ecmwf.int/wiki/display/GRIB/Module+Index

  Python: https://software.ecmwf.int/wiki/display/GRIB/Python+package+gribapi

- GRIB API examples:

  https://software.ecmwf.int/wiki/display/GRIB/Grib+API+examples

  https://software.ecmwf.int/wiki/display/ECC/GRIB+examples

- GRIBEX – GRIB API conversion:

  https://software.ecmwf.int/wiki/display/GRIB/GRIBEX+keys