

GRIB API

Fortran 90 - C - Python interfaces part 2

Dominique Lucas – Xavier Abellan Ecija
User Support

Content

- GRIB API indexed access
- Encoding a loaded GRIB message
- C API
- Python API

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 grib 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(n)**)

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

GRIB API indexed access

- Several subroutines:

`grib_index_create(indexid, filename, keys, status)`

to create the index of the content of a file

`grib_index_get_size(indexid, key, size, status)`

to get the dimension of a key in the index

`grib_index_get(indexid, key, values, status)`

to get the different “values” for a key in the index

`grib_index_select(indexid, key, value, status)`

to select a “value” for a key in the index

Input arguments

Output arguments

GRIB API indexed access

Input arguments

Output arguments

- Several subroutines:

`grib_new_from_index(indexid, igrib, status)`

to load the GRIB message corresponding to the selection made.

`grib_index_release(indexid, status)`

to release the index.

and ... `grib_release(igrib)`

- Indexed access is usually much faster than sequential access for “random” access.

Example – indexed access

Input arguments
Output arguments

! create an index from a grib file using two keys
call `grib_index_create(idx,'ensemble.grib','paramId')`

List of keys to be indexed, comma separated, without any spaces, between one single set of quotes.

! get the number of distinct values of parameters in the index
call `grib_index_get_size(idx,'paramId',paramIdSize)`

! allocate the array to contain the list of distinct paramId
`allocate(paramId(paramIdSize))`

File “ensemble,grib” contains all ensemble members for several parameters.

! get the list of distinct parameters from the index
call `grib_index_get(idx,'paramId',paramId)`

Note that I have to select a value for all the keys used to build the index.

```
count=1
do i=1,paramIdSize ! loop on paramId
  ! select paramId=paramId(i)
  call grib_index_select(idx,'paramId',paramId(i))
  call grib_new_from_index(idx,igrib,iret)
```


I load the first grib message I need into memory.

Example – indexed access

Input arguments
Output arguments

```
do while (iret /= GRIB_END_OF_INDEX)
  call grib_is_missing(igrib,'number', is_missing);
  if (is_missing /= 1) then
    call grib_get(igrib,'number',onumber)
  else
    onumber=-9999
  end if
  call grib_get(igrib,'level',olevel)
  print*, 'param:', paramId(i), ' level:', olevel, ' number:', onumber
  call grib_release(igrib)
  call grib_new_from_index(idx,igrib,iret)
end do
```

*Note that several grib messages
may be available for one selection
of my index, therefore this loop.*



```
end do ! loop on paramId
call grib_index_release(idx)
```

GRIB API indexed access – i/o

- An index can be saved into a file, to be re-used.

`grib_index_write(indexid, filename, status)`

to save an index to a file

Input arguments

Output arguments

`grib_index_read(indexid, filename, status)`

to load an index file previously created with `grib_index_write`

- One can also add the content of a data file to an index.

`grib_index_add_file(indexid, filename, status)`

to add the content of a data file to an index.

- One can build an index with the `grib_api` command

`grib_index_build`.

- A little more on this in the practical session.

Encoding a loaded GRIB message

- The idea is to “encode” as little as possible! You will never “encode” the **whole GRIB message**.
- one main subroutine to “encode”:

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

Input arguments
Output arguments

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

- Writing a message:

```
grib_write(igrib, output_file)
```

Note that a grib message written with grib_write will be **syntactically correct**, but it may be **semantically incorrect**.

Creation of a new message

- A new message can be created from a sample:
 - A sample is an example grib message available in the sample directory. The default sample directory can be found with the command '[grib_info](#)'. Sample file names end up with a suffix '.tmpl'. You can create your own samples and change/add the environment variable GRIB_SAMPLES_PATH to point to them.
 - Creating a new grib message from a sample:
`grib_new_from_samples(igrib, samplename, status)`
- A new message can be cloned (copied) from another message:

`grib_clone(igrib_src,igrib_dest,status)`

Input arguments

Output arguments

Example – grib_set

Input arguments
Output arguments

! STEP-1: open output file and load a GRIB message from a sample "GRIB1"

call `grib_open_file(outfile, 'out.grib1','w')`

call `grib_new_from_samples(igrib, "GRIB1")`

! GRIB1.tmpl is a GRIB-1 file located
! in the samples directory

! STEP-2: Get some information from the loaded message

call `grib_get_size(igrib, "values", nb_values)`

`allocate(values(nb_values))`

`call model(values); values(1:100) = 9999.0`

! Declared as real, dimension(:), allocatable
! Compute values and set some missing values

! STEP-3: set the new GRIB message

call `grib_set(igrib,'missingValues', 9999.0)`

call `grib_set(igrib,'bitmapPresent', 1)`

call `grib_set(igrib,"values", values)`

! Tells the GRIB-API 9999.0 is the missing value

! Set values as 1D real array of size nb_values

! STEP-4: write modified message to a file

call `grib_write(igrib,outfile)`

call `grib_release(igrib)`

call `grib_close_file(outfile)`

`deallocate(values)`

Changing grid definition and packing type

- You can apply a grid definition or change the packing type by changing the keys `gridType` and/or `packingType`, e.g:

```
call grib_set(igrib,'gridType', 'polar_stereographic')
```

will define a "Polar Stereographic Projection Grid" for your message.

```
call grib_set(igrib,'packingType', 'grid_simple')
```

will pack the data as single packing.

- The grid definitions and grib packing types are listed under:
<https://software.ecmwf.int/wiki/display/GRIB/Grib+API+keys>

Usage different packing types

- GRIB data can be packed in different ways, e.g. simple packing, second order packing, ...
- Not all packing types are available for GRIB1 and GRIB2.
- A packing type will be available either for grid-point or spectral field.
- The type of packing used will affect the size of your GRIB messages produced, e.g. second order packing may produce messages twice as small as simple packing.
- The type of packing used will affect the time it takes to pack/unpack your data, e.g. second order packing may be many times slower than simple packing.
- Packing doesn't lose information.
- More on this in the practical session ...

C API – Indexing 1/3

- There is no need for using `fopen()/fclose()` anymore!
- `grib_index * grib_index_new_from_file(grib_context *c, char *filename, const char *keys, int *err)`
 - Create a new index from a file.
 - `grib_context *c` should usually be set to 0.
- `int grib_index_get_size(grib_index *index, const char *key, size_t *size)`
 - Get the number of distinct values of the key in argument contained in the index.
- `int grib_index_get_double(grib_index *index, const char *key, double *values, size_t *size)`
 - Get the distinct values of the key in argument contained in the index. Before that you will need to allocate memory for amount given by `grib_index_get_size()`.

C API – Indexing 2/3

- int `grib_index_get_string`(grib_index *index, const char *key, char **values, size_t *size)
 - Get the distinct values of the key contained in the index.
 - An array of “char **” of size “size” has to be allocated before.
 - size will contain actual size of assigned string.
 - Example:

```
char** paramId=NULL;
```

```
GRIB_CHECK( grib_index_get_size( index, "paramId", &paramIdSize ), 0 );
```

```
paramId = ( char** ) malloc ( sizeof( char* ) * paramIdSize );
```

```
GRIB_CHECK( grib_index_get_string( index, "paramId", paramId, &paramIdSize ), 0 );
```

```
for ( i = 0; i < paramIdSize; i++ ) free( paramId[i] );
```

```
free( paramId );
```

C API – Indexing 3/3

- int `grib_index_select_TYPE`(grib_index *index, const char *key, TYPE value)
 - Select the message subset with key==value.
- grib_handle * `grib_handle_new_from_index`(grib_index *index, int *err)
 - Create a new handle from an index after having selected the key values.
 - After handle has been used you have to call `grib_handle_delete()` to free memory!
 - Another call of `grib_handle_new_from_index()` will create a grib handle pointing to the next grib message of the index.
- int `grib_index_add_file`(grib_index *index, const char *filename)
 - Add another file to an existing index

C API – Encoding

- int `grib_set_double`(grib_handle *h, const char *key, double value)
 - Set a double value from a key. Similar function for long exists.
- int `grib_set_string`(grib_handle *h, const char *key, const char *mesg, size_t *length)
 - Set a string value from a key. Similar function for `bytes` exists.
- int `grib_set_double_array`(grib_handle *h, const char *key, const double *vals, size_t length)
 - Set a double array from a key. Similar function for `long array` exists.

C API – Cloning

1. Create handle for existing grib message
2. `grib_handle * grib_handle_clone(grib_handle *h)`
 - Clone an existing handle using the context of the original handle, the message is copied and reparsed.
3. Encode (overwrite) keys in new grib message
4. `int grib_get_message(grib_handle *h_new, const void **message, size_t *message_length)`
 - getting the raw grib message attached to a handle.
5. Open output file: `out = fopen(file, "w")`
6. Write message: `fwrite(message, 1, message_length, out)`
7. Close file: `fclose(out)`

Python API – Indexing 1/3

- `iid = grib_index_new_from_file(file, keys)`
 - Returns a handle to the created index
 - Release with `grib_index_release(iid)`
- `grib_index_add_file(iid, file)`
 - Adds a file to an index.
- `grib_index_write(iid, file)`
 - Writes an index to a file for later reuse.
- `iid = grib_index_read(file)`
 - Loads an index saved with `grib_index_write()` to a file.

Python API – Indexing 2/3

- size = *grib_index_get_size*(iid, key)
 - Gets the number of distinct values for the index key.
- values = *grib_index_get*(iid, key, type=str)
 - Gets the distinct values of an index key.
- *grib_index_select*(iid, key, value)
 - Selects the message subset with key==value.
- gid = *grib_new_from_index*(iid)
 - Same as *grib_new_from_file*
 - Release with *grib_release*(gid)

Python API – Encoding

- *`grib_set`*(gid, key, value)
 - Sets the value for a scalar key in a grib message.
- *`grib_set_array`*(gid, key, value)
 - Sets the value for an array key in a grib message.
 - The input array can be a `numpy.ndarray` or a Python sequence like tuple, list, array, ...
- *`grib_set_values`*(gid, values)
 - Utility function to set the contents of the 'values' key.

Python API – Cloning

- `clone_id = grib_clone(gid_src)`
 - Creates a copy of a message.
 - You can directly write to file with *grib_write*
 - Don't forget to *grib_release*

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
- (Other possibility is npoints=4 which returns a list of the 4 nearest points)

iter_id = *grib_iterator_new*(gid,mode)

[lat,lon,value] = *grib_iterator_next*(iterid)

grib_iterator_delete(iter_id)