

GRIB Decoding

Computer User Training Course 2014

Paul Dando

User Support Section

advisory@ecmwf.int

Contents

- GRIB description and overview
 - GRIB edition 1 and GRIB edition 2
 - Major differences between GRIB edition 1 and 2
 - Status of ECMWF migration to GRIB edition 2
- Using GRIB Tools
 - Introduction to GRIB API
 - Inspecting the content of GRIB messages
 - Decoding GRIB messages
 - Manipulation of GRIB messages
- Decoding GRIB messages with Fortran 90 ... and Python

GRIB

- GRIB – “General Regularly-distributed Information in Binary form”
- Code defined by the WMO / CBS in 1985
- Designed to exchange and store large volumes of gridded data
- Machine independent
- Requires software for encoding and decoding
- Currently there are two different coding standards

GRIB edition 1

- Currently used for ECMWF operational surface and pressure level data

GRIB edition 2

- Recent format now being used by some centres and for the TIGGE archive
- Used for ECMWF operational model level data since 18 May 2011

GRIB edition 1 overview

Main feature of GRIB is that data descriptors are self-defining

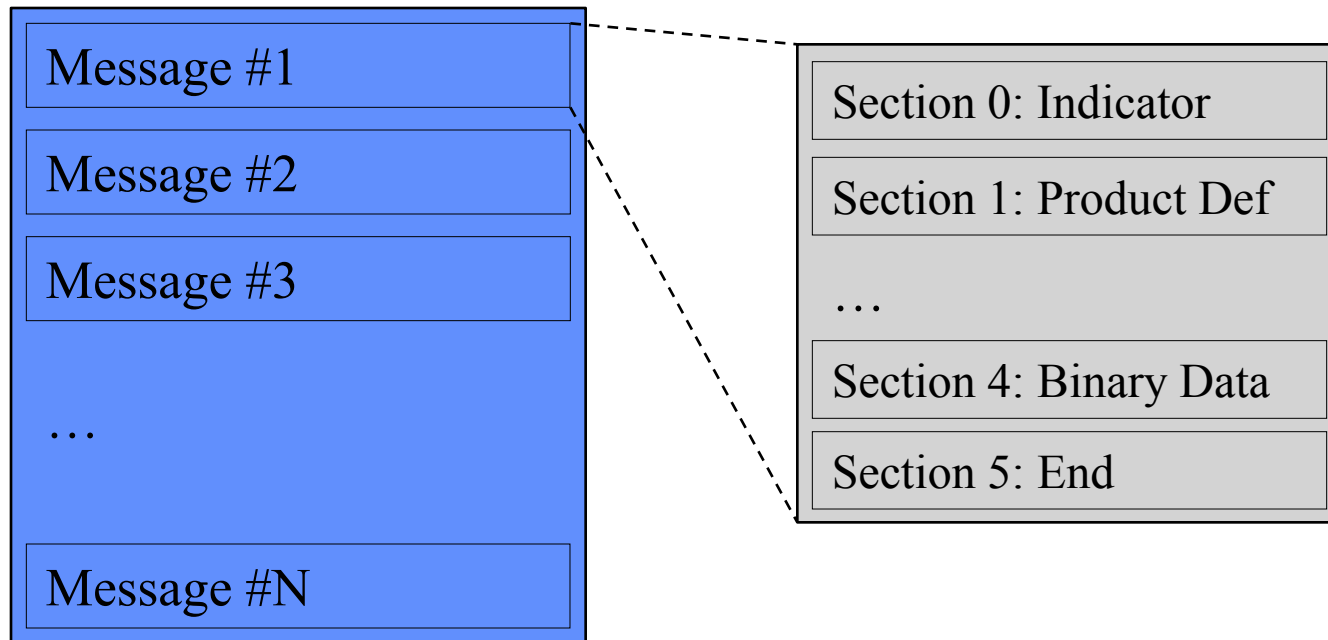
- Section 0 – Indicator section
- Section 1 – Product definition section
- Section 2 – [Grid description section]
- Section 3 – [Bit map section]
- Section 4 – Binary data section
- Section 5 – 7777 (End of GRIB message)

In the above [] indicates an optional section

GRIB Structure

- A **file** may contain one or more GRIB **messages**
- Each message contains several **sections**
- Note: A file can contain a mix of editions 1 and 2

File: eps.grib



GRIB 1 & GRIB 2 – Different Structure

GRIB 1

- SECTION 0 Indicator
- SECTION 1 Product Definition
- SECTION 2 [Grid Description]
- SECTION 3 [Bitmap]
- SECTION 4 Binary Data
- SECTION 5 End (7777)

Optional sections are indicated by []

GRIB 2

- SECTION 0 Indicator
- SECTION 1 Identification
- SECTION 2 [Local Use]
- SECTION 3 Grid Definition
- SECTION 4 Product Definition
- SECTION 5 Data Representation
- SECTION 6 Bitmap
- SECTION 7 Binary Data
- SECTION 8 End (7777)

repeat



GRIB 1 & GRIB 2 – Major differences

- The coding principles for GRIB edition 1 and 2 are similar but their implementation is **very different**
- The structure of GRIB 1 and GRIB 2 messages is different
 - Both have sections but with **different meanings**
- In GRIB 2 several variables are defined with more precision
 - In GRIB 1 latitudes and longitudes are in milli-degrees
 - In GRIB 2 latitudes and longitudes are in micro-degrees
- Encoding of the parameter is **very different**
- In GRIB 2 the description of the data (parameter, time, statistics, grid...) is template / table based
 - More flexible ... but also more complex !

Use of GRIB 2 at ECMWF

What is currently affected ?

- Since **18 May 2011** all model level fields for HRES and ENS (including the monthly extension) are encoded in GRIB 2
 - GRIB 1 model level data are no longer produced or disseminated
- Most surface and all pressure level fields are encoded in GRIB 1
 - Some recently introduced surface fields are encoded in GRIB 2
- Staged migration of remaining GRIB 1 fields to GRIB 2 will follow

And what's not ?

- The wave model
- The System-4 seasonal forecast model
- ERA-Interim

Introducing GRIB API

- GRIB API Library is an application programming interface developed by ECMWF
- GRIB API hides the binary layer of the message providing the user with a higher level of access
- It provides an easy and reliable way of encoding and decoding both GRIB 1 and GRIB 2 messages
- GRIB API decodes / encodes both GRIB editions with the **SAME** function calls
- Command line tools (the GRIB Tools) provide a quick and easy way to manipulate data
- Fortran 90, C and Python interfaces give access to the main features of the library

GRIB API approach

- GRIB API uses a key / value approach to access the information in a GRIB message
 - `numberOfPointsAlongAParallel` → Number of points along a parallel
 - `numberOfPointsAlongAMeridan` → Number of points along a meridian
 - ...
- The set of keys available changes from one message to another depending on:
 - the GRIB edition
 - the content of the message
- Changing the values of some keys can cause some other keys to disappear and new keys to become available

GRIB API – coded and computed keys

- The value of a key is not always coded in the GRIB message
- Some keys are combinations of several other keys and provided through a given algorithm or can be just temporary (transient)
- Therefore we talk about
 - **CODED** keys (coded in the message as they are)
 - **COMPUTED** keys (temporary or computed from other keys)
- Alternative names (“**aliases**”) are provided for several keys

GRIB API – coded and computed keys

- Coded keys

- Linked directly to the octets of the GRIB message
- Values obtained by decoding the octet e.g. [indicatorOfParameter](#)

- Computed keys

- Obtained by combining other keys (coded or computed)
- Provide a synthesis of the information contained in the message
- Provide a convenient way to access complex attributes
- Setting the value of a computed key sets all related keys in a cascade
 - e.g. setting [typeOfGrid=regular_ll](#) will set all the various keys in the Grid Definition Section for a regular lat-long grid
- MARS keywords are available as computed keys

GRIB API keys – parameter

- The definition of the parameter is very different in the two editions

GRIB 1 keys	GRIB 2 keys
centre	discipline
table2Version	parameterCategory
indicatorOfParameter	parameterNumber
levelType	typeOfFirstFixedSurface
level	scaleFactorOfFirstFixedSurface
...	scaledValueOfFirstFixedSurface
	typeOfSecondFixedSurface
	scaleFactorOfSecondFixedSurface
	scaledValueOfSecondFixedSurface
	productDefinitionTemplateNumber
	...

GRIB API keys – parameter

- GRIB API provides some **edition-independent** keys to identify a parameter

Key name	Example value
paramId	151
shortName	msl
centre	ecmf (or 98)
name	Mean sea level pressure
unit	Pa

- This set of keys is the parameter *namespace*

The parameter database

- The parameter database stores information about the GRIB 1, GRIB 2 and, for some parameters, netCDF encoding of all parameters recognised by GRIB API
- The database is accessible via a web interface at:
www.ecmwf.int/publications/manuals/d/gribapi/param

GRIB API – namespace

- A **namespace** is a name for a set of keys.
- There are several different namespaces
 - parameter
 - time
 - geography
 - vertical
 - statistics
 - mars

GRIB API keys – time

- Start of forecast run

Key name	Example values
dataDate	20140305 (YYYYMMDD)
dataTime	0, 600, 1200, 1800

- Forecast Step

Key name	Example values
stepType	instant, accum, avg, max, min, ...
stepUnits	s, m, h, 3h, 6h, 12h, D, M, Y, 10Y, 30Y, C
startStep	0, 3, ...
endStep (= step)	0, 3, ..
stepRange	3-6, 6 (“startStep-endStep” , “endStep”)

- Validity of the forecast

Key name	Example values
validityDate	20140305 (YYYYMMDD)
validityTime	0, 300, 1200, 1800

GRIB API keys – MARS

- There is a namespace consisting of all the MARS keywords

Key name	Example values
date	20140305 (YYYYMMDD)
time	0000, 0600, 1200, 1800
step	3, 6, 9, 12, ...
class	od, ...
stream	oper, enfo, ...
expver	0001
type	an, fc, cf, pf, ...
levtype	sfc, pl, ml
levelist	500, 850, ...
param	151.128

GRIB API keys – THE Reference

- GRIB 1 keys

www.ecmwf.int/publications/manuals/d/gribapi/fm92/grib1/

- GRIB 2 keys

www.ecmwf.int/publications/manuals/d/gribapi/fm92/grib2/

- Edition independent keys

www.ecmwf.int/publications/manuals/d/gribapi/keys/

- Use edition independent keys where possible !

GRIB API keys

- The easiest way to inspect a GRIB file and to find the keys available is to use the **GRIB Tools**
 - `grib_ls` to get a summary of the content
 - `grib_dump` to get a more detailed view

GRIB Tools – basic concepts

- The GRIB tools are part of the ECMWF GRIB API Library
- They are a set of command line programs for interactive and batch decoding and processing of GRIB data
- They provide ready and tested solutions to the most common processing of GRIB data
 - They work with both GRIB edition 1 and GRIB edition 2
- Their use will avoid the need to write new code and thus speed up your work
 - Consider using GRIB Tools instead of writing your own programs
- The tools are provided with many common options so that it is quick to apply the same options to different tools
- **Use of the tools is recommended whenever possible !**

GRIB Tools – more basics

- All of the tools use a common syntax

```
grib_<tool> [options] grib_file grib_file ... [output_grib]
```

- There are tools for getting information about the GRIB API installation and the keys available
 - grib_info, grib_keys
- There are tools to inspect the content of and compare GRIB messages
 - grib_ls, grib_dump, grib_get, grib_get_data, grib_compare
- There are tools for counting and copying some messages
 - grib_count, grib_copy
- There are tools for making changes to the content of a GRIB message and converting GRIB to netCDF
 - grib_set, grib_filter, grib_to_netcdf

GRIB Tools – getting help

- UNIX ‘man’-style pages are available for each tool by running the tool without any options or input file

```
> grib_dump
```

```
NAME      grib_dump
```

```
DESCRIPTION
```

```
Dump the content of a grib file in different formats.
```

```
USAGE
```

```
grib_dump [options] grib_file grib_file ...
```

```
OPTIONS
```

```
-O      Octet mode. WMO documentation style dump.
```

```
-D      Debug mode.
```

```
-d      Print all data values.
```

```
...
```

`grib_ls` – list the content of GRIB files

- Use `grib_ls` to get a summary of the content of GRIB files
- Without options `grib_ls` prints a default list of keys
- Options exist to specify the set of keys to print or to print other keys in addition to the default set
- Output can be ordered
 - e.g. order by ascending or descending step
- `grib_ls` does not fail if a key is not found
- `grib_ls` can also be used to find the grid point(s) nearest to a specified latitude-longitude and print the value of the field at that point
 - Modes available to obtain one or four nearest grid points

grib_ls – usage

```
grib_ls [options] grib_file grib_file ...
```

Basic options

-p key[:{s|l|d}],...

Keys to print

-P key[:{s|l|d}],...

Additional keys to print

-w key{=/{!=}value},...

Where option

-B "key asc, key desc"

Order by: "step asc, centre desc"

-n namespace

Print keys for **namespace**

-m

Print MARS keys

-i index

Print data value at given index

-l lat,lon[,MODE,FILE]

Value(s) nearest to lat-lon point

-F format

Format for floating point values

-W width

Minimum column width (default 10)

grib_ls – examples

```
> grib_ls file.grib1
```

```
file.grib1
edition  centre  typeOfLevel  level  dataDate  ...  dataType  shortName  packingType  gridType
1        ecmf    isobaricInhPa  1000  20140304  ...  an        t          spectral_complex  sh
1        ecmf    isobaricInhPa   500  20140304  ...  an        t          spectral_complex  sh
1        ecmf    isobaricInhPa   200  20140304  ...  an        t          spectral_complex  sh
1        ecmf    isobaricInhPa   100  20140304  ...  an        t          spectral_complex  sh
4 of 4 grib messages in file.grib1

4 of 4 total grib messages in 1 files
```

- Use **-p** option to specify a list of keys to be printed

```
> grib_ls -p centre:l,dataDate,shortName,paramId,typeOfLevel,level file.grib1
```

```
file.grib1
centre      dataDate      shortName      paramId      typeOfLevel      level
98          20140304      t              130          isobaricInhPa    1000
98          20140304      t              130          isobaricInhPa    500
98          20140304      t              130          isobaricInhPa    200
98          20140304      t              130          isobaricInhPa    100
4 of 4 grib messages in file.grib1

4 of 4 total grib messages in 1 files
```

grib_ls – examples

- When a key is not present in the GRIB file, it returns “not found” for this key

```
> grib_ls -p my_key file.grib1
```

```
file.grib1
```

```
my_key
```

```
not_found
```

```
> echo $?
```

```
0
```

exit code returned = 0



- Similar behaviour to `grib_get` (see later)
 - `grib_ls` is more for interactive use
 - use `grib_get` within scripts

Using the 'where' option

- The where option `-w` can be used with all the GRIB Tools
- Constraints are of the form `key=value` or `key!=value` or `key=value1/value2/value2`
 - `-w key[:{s|l|d}]=value,key[:{s|l|d}]!=value`
- Messages are processed only if they match **ALL** the key / value constraints

```
> grib_ls -w level=100 file.grib1 "IS"
...
> grib_ls -w level!=100 file.grib1 "NOT"
...
> grib_ls -w level=100,stepRange=3 file.grib1 "AND"
...
> grib_ls -w level=100/200/300/500 file.grib1 "OR"
...
```

grib_dump – dump content of GRIB files

- Use **grib_dump** to get a detailed view of the content of a file containing one or more GRIB messages
- Various output formats are supported
 - **Octet mode** provides a WMO documentation style dump
 - **Debug mode** prints all keys available in the GRIB file
 - **Octet** and **Debug modes** cannot be used together
 - Octet content can also be printed in hexadecimal format
- Options also exist to print key **aliases** and key **type** information

grib_dump – usage

```
grib_dump [options] grib_file grib_file ...
```

Basic options

- O** Octet mode (WMO Documentation style)
- D** Debug mode
- a** Print key alias information
- t** Print key type information
- H** Print octet content in hexadecimal
- w key{=/!}=value,...** Where option
- d** Print all data values
- ...**

`grib_dump` – examples

```
> grib_dump file.grib1
```

```
***** FILE: file.grib1
#===== MESSAGE 1 ( length=4284072 ) =====
GRIB {
  editionNumber = 1;
  table2Version = 128;
  # European Center for Medium-Range Weather Forecasts (grib1/0.table)
  centre = 98;
  generatingProcessIdentifier = 141;
  # Geopotential (m**2 s**-2) (grib1/2.98.128.table)
  indicatorOfParameter = 129;
  # Surface (of the Earth, which includes sea surface) (grib1/3.table)
  indicatorOfTypeOfLevel = 1;
  level = 0;
  # Forecast product valid at reference time + P1 (P1>0) (grib1/5.table)
  timeRangeIndicator = 0;
  # Unknown code table entry (grib1/0.ecmf.table)
  subCentre = 0;
  paramId = 129;
  #-READ ONLY- units = m**2 s**-2;
  #-READ ONLY- nameECMF = Geopotential;
  #-READ ONLY- name = Geopotential;
  decimalScaleFactor = 0;
  dataDate = 20140304;
  dataTime = 0; ...
```

*Some keys are
read only*

*keys are case sensitive:
dataDate, dataTime*

grib_dump - examples

```
> grib_dump -O file.grib1
```

```
***** FILE: file.grib1
===== MESSAGE 1 ( length=4284072 ) =====
1-4      identifier = GRIB
5-7      totalLength = 4284072
8        editionNumber = 1
===== SECTION_1 ( length=52, padding=0 ) =====
1-3      section1Length = 52
4        table2Version = 128
5        centre = 98 [European Center for Medium-Range Weather Forecasts
                                     (grib1/0.table) ]
6        generatingProcessIdentifier = 141
7        gridDefinition = 255
8        section1Flags = 128 [10000000]
9        indicatorOfParameter = 129 [Geopotential (m**2 s**-2)
                                     (grib1/2.98.128.table) ]
10       indicatorOfTypeOfLevel = 1 [Surface (of the Earth, which includes sea
                                     surface) (grib1/3.table) ]
11-12    level = 0
13       yearOfCentury = 14
14       month = 3
15       day = 4
16       hour = 0
17       minute = 0
18       unitOfTimeRange = 1 [Hour (grib1/4.table) ] ...
```


`grib_dump` – examples

```
> grib_dump -OtaH file.grib1
```

```
***** FILE: file.grib1
===== MESSAGE 1 ( length=4284072 ) =====
1-4  ascii identifier = GRIB ( 0x47 0x52 0x49 0x42 )
5-7  g1_message_length totalLength = 4284072 ( 0x41 0x5E 0xA8 )
8    unsigned editionNumber = 1 ( 0x01 ) [ls.edition]
===== SECTION_1 ( length=52, padding=0 ) =====
1-3  section_length section1Length = 52 ( 0x00 0x00 0x34 )
4    unsigned table2Version = 128 ( 0x80 ) [gribTablesVersionNo]
5    codetable centre = 98 ( 0x62 ) [European Center for Medium-Range Weather
Forecasts (grib1/0.table) ] [identificationOfOriginatingGeneratingCentre,
originatingCentre, ls.centre, centreForTable2]
6    unsigned generatingProcessIdentifier = 141 ( 0x88 )
[generatingProcessIdentificationNumber, process]
7    unsigned gridDefinition = 255 ( 0xFF )
8    codeflag section1Flags = 128 [10000000] ( 0x80 )
9    codetable indicatorOfParameter = 129 ( 0x81 ) [Geopotential (m**2 s**-2)
(grib1/2.98.128.table) ]
10   codetable indicatorOfTypeOfLevel = 1 ( 0x01 ) [Surface (of the Earth,
which includes sea surface) (grib1/3.table) ] [levelType, mars.levtype]
11-12 unsigned level = 0 ( 0x00 0x00 ) [vertical.topLevel,
vertical.bottomLevel, ls.level, lev]
13   unsigned yearOfCentury = 14 ( 0x0E )
14   unsigned month = 3 ( 0x03 )
15   unsigned day = 4 ( 0x04 ) ...
```

Practicals

- Work in your \$SCRATCH

```
cd $SCRATCH
```

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

```
tar -xvf /scratch/ectrain/trx/grib_practicals.tar
```

- This will create a directory in your \$SCRATCH containing the GRIB data files for all today's practicals
- There are sub-directories for each practical:

```
ls $SCRATCH/grib_practicals
```

```
practical1 practical2 practical3
```

```
practical4 practical5 practical6
```

Practical 1: using `grib_ls` and `grib_dump`

1. Use `grib_ls` to inspect the content of the files `msl.grib1` and `msl.grib2`
 - Which keys does `grib_ls` show by default ?
 - What fields do the GRIB messages contain ?
 - Print the MARS keys. Add the `shortName` to the output
 - Order the output in descending step order
2. Use `grib_ls` to print the `centre`, `dataDate`, `stepRange`, `typeOfLevel` and `shortName` for forecast step 6 only
 - Output the `centre` as both a string and an integer
3. Use `grib_dump` to inspect the fourth (`count=4`) GRIB message in both files
 - Experiment with the different `grib_dump` options: `-O`, `-a` and `-t`
 - Identify the parameter, date, time, forecast step and the grid geometry

Finding nearest grid points with grib_ls

- The value of a GRIB field close to a specified Latitude/Longitude point can be found with `grib_ls`

```
grib_ls -l Latitude,Longitude[,MODE,file] grib_file
```

MODE Can take the values

- 4 Print values at the 4 nearest grid points (default)
- 1 Print value at the closest grid point

file Specifies a GRIB file to use as a mask
The closest *land* point (with mask ≥ 0.5) is printed

- GRIB files specified **must** contain grid point data

Practical 2: using grib_ls -l

1. The file msl.grib1 contains the mean sea-level pressure from the EPS control forecast at 6-hourly time steps for the first 24 hours on a N320 reduced Gaussian grid
2. Find the value of the MSLP at the grid point nearest to ECMWF (Lat 51.42°N, Lon 0.95° W) at each forecast step
 - What is the lat-lon value of the grid point nearest to ECMWF ?
 - How far is the chosen grid point from ECMWF ?
3. Change the command used to output only the forecast step and the MSLP value at the nearest grid point
4. Change the command to output the MSLP values at the four grid points nearest to ECMWF
5. Use the file lsm.grib1 to provide a land-sea mask
 - Are all four nearest grid points land points (mask ≥ 0.5) ?

`grib_get` – get key / value pairs

- Use `grib_get` to get the values of one or more keys from one or more GRIB files – very similar to `grib_ls`
- By default `grib_get` **fails** if an error occurs (e.g. key not found) returning a non-zero exit code
 - Suitable for use in scripts to obtain key values from GRIB messages
 - Can force `grib_get` not to fail on error
- Options available to get all MARS keys or all keys for a particular namespace
 - Can get other keys in addition to the default set
- Format of floating point values can be controlled with a C-style format statement

grib_get – usage

```
grib_get [options] grib_file grib_file ...
```

● Options

-p `key[:{s|l|d}] ,...`

Keys to get

-P `key[:{s|l|d}] ,...`

Additional keys to get with **-m**, **-n**

-w `key{=/!}=value ,...`

Where option

-s `key[:{s/l/d}]=value ,...`

Keys to set

-n `namespace`

Get all keys for **namespace**

-m

Get all MARS keys

-l `lat,lon[,MODE,FILE]`

Value(s) nearest to lat-lon point

-F `format`

Format for floating point values

-f

Do *not* fail on error

...

`grib_get` – examples

- To get the centre of the first (`count=1`) GRIB message in a file (both as a 'string' and a 'long')

```
> grib_get -w count=1 -p centre f1.grib1
ecmf

> grib_get -w count=1 -p centre:l f1.grib1
98
```

- `grib_get` fails if there is an error

```
> grib_get -p mykey f1.grib1
GRIB_API ERROR      : Key/value not found

> echo $?
246
```

← returns the exit code from the previous command

grib_get – examples

- To get all the MARS keys, optionally printing the **shortName**

```
> grib_get -m f1.grib1
g sfc 20140304 1200 0 167.128 od an oper 0001

> grib_get -m -P shortName f1.grib1
2t g sfc 20140304 1200 0 167.128 od an oper 0001
```

- To get all keys belonging to the statistics namespace

```
> grib_get -n statistics f1.grib1
314.24 214.613 277.111 21.0494 41379.8 2.48314e-05 0
```

grib_get – controlling output format

- The format of floating point values can be controlled by using a C-style format statement with the **-F** option

-F "%.4f" - Decimal format with 4 decimal places (1.2345)

-F "%.4e" - Exponent format with 4 decimal places (1.2345E-03)

```
> grib_get -F "%.6f" -p maximum f1.grib1
```

```
314.240280
```

```
> grib_get -F "%.4e" -p maximum f1.grib1
```

```
3.1424e+02
```

- Default format is **-F "%.10e"**

`grib_get` – `stepRange` and `stepUnits`

- The step is always printed as an **integer** value
- By default the units of the step are printed in hours
- To obtain the step in other units set the `stepUnits` appropriately with the `-s` option

```
> grib_get -p stepRange f1.grib1
```

```
6
```

```
12
```

```
> grib_get -s stepUnits=m -p stepRange f1.grib1
```

```
360
```

```
720
```

```
stepUnits can be s, m, h, 3h, 6h, 12h, D, M, Y, 10Y, 30Y, C
```

Finding nearest grid points with grib_get

- The value of a GRIB field close to a specified Latitude/Longitude point can be found with `grib_get`
 - Works in the same way as `grib_ls`

```
> grib_get -l 52.0,-1.43 f1.grib1
```

```
273.58 272.375 273.17 273.531
```

```
> grib_get -F "%.5f" -P stepRange -l 52.0,-1.43,1 f1.grib1
```

```
0 272.37505
```

- GRIB files specified **must** contain grid point data

Getting data values at an index

- The value at a particular index can be printed using `grib_get` with the `-i` option
- For example, find the index of a nearest grid point with `grib_ls` and then use this with `grib_get` to build a list of values at that point:

```
> grib_get -F "%.2f" -i 2159 -p stepRange f1.grib1
```

```
6 99429.31
```

```
12 99360.25
```

```
18 99232.31
```

```
24 99325.56
```

- Also returns a value for non-grid point fields !

grib_get_data – print data values

- Use `grib_get_data` to print a list of latitude, longitude (for grid point data) and data values from one or more GRIB files
 - The format of the output can be controlled by using a C-style format statement with the `-F` option
 - `-F "% .4f"` – Decimal format with 4 decimal places (1.2345)
 - `-F "% .4e"` – Exponent format with 4 decimal places (1.2345E-03)
- The default format is `-F "% .10e"`
- By default missing values are not printed
 - A user-provided string can be printed in place of any missing values
 - By default `grib_get_data` fails if there is an error
 - Use the `-f` option to force `grib_get_data` not to fail on error

`grib_get_data` – usage

```
grib_get_data [options] grib_file grib_file ...
```

● Options

`-p key[:{s|l|d}],...`

Keys to print

`-w key{=/!}=value,...`

Where clause

`-m missingValue`

Specify missing value string

`-F format`

C-style format for output values

`-f`

Do *not* fail on error

`-V`

Print GRIB API Version

...

`grib_get_data` – example

```
> grib_get_data -F "%.4f" f1.grib1
```

Latitude,	Longitude,	Value
81.000	0.000	22.5957
81.000	1.500	22.9009
81.000	3.000	22.8359
81.000	4.500	22.3379
81.000	6.000	21.5547
81.000	7.500	20.7344
81.000	9.000	19.8916
81.000	10.500	18.5747
81.000	12.000	17.2578
81.000	13.500	16.1343
81.000	15.000	14.9785
81.000	16.500	13.8296

...

*Format option
applies to values
only - not to the
Latitudes and
Longitudes*

`grib_get_data` – missing values example

```
> grib_get_data -m XXXXX -F "%.4f" f1.grib1
```

```
Latitude, Longitude, Value
```

```
...
```

81.000	90.000	9.4189
81.000	91.500	8.6782
81.000	93.000	XXXXX
81.000	94.500	XXXXX
81.000	96.000	XXXXX
81.000	97.500	XXXXX
81.000	99.000	6.7627
81.000	100.500	7.4097
81.000	102.000	7.9307

```
...
```

*Missing values are
printed with
XXXXX*

Practical 3: using `grib_get` & `grib_get_data`

1. Use `grib_get` to obtain a list of all the pressure levels available for parameter T in the file `tz_an_pl.grib1`
2. Use `grib_get` to print the `stepRange` for the field in the file `surface.grib1` in (a) hours (b) minutes and (c) seconds
3. Repeat 2. for `surface2.grib1`
4. Use `grib_get_data` to print the latitude, longitude and values for the field in `surface.grib1`
 - Output results in decimal format with 5 decimal places
 - Output results in exponential format with 10 decimal places
 - Are there any missing values ?
5. Use `grib_get_data` to print the data values for the temperature at 500 hPa **only** from the file `tz_an_pl.grib1` ?
 - Make sure you print only the data for T500 ! What is printed ?

grib_copy – copy contents of GRIB files

- Use **grib_copy** to copy selected contents of GRIB files optionally printing some key values
- Without options **grib_copy** prints **no** key information
- Options exist to specify the set of keys to print
 - Use verbose option (**-v**) to print keys
- Output can be ordered
 - E.g. order by ascending or descending step
- Key values can be used to specify the output file names
- **grib_copy fails** if a key is not found
 - Use the **-f** option to force **grib_copy** not to fail on error

grib_copy – usage

```
grib_copy [options] grib_file grib_file ... out_grib_file
```

● Options

`-p key[:{s|l|d}],...`

Keys to print (only with `-v`)

`-w key{=/!}=value,...`

Where option

`-B "key asc, key desc"`

Order by: "step asc, centre desc"

`-v`

Verbose

`-f`

Do *not* fail on error

...

grib_copy – examples

- To copy only fields at 100 hPa from a file

```
> grib_copy -w level=100 in.grib1 out.grib1
```

- To copy only those fields that are not at 100 hPa

```
> grib_copy -w level!=100 in.grib1 out.grib1
```

- Information can be output using the **-v** and **-p** options

```
> grib_copy -v -p shortName in.grib1 out.grib1
in.grib1
shortName
t
1 of 1 grib messages in in.grib1
1 of 1 total grib messages in 1 files
```

grib_copy – using key values in output file


- Key values can be used to specify the output file name

```
> grib_copy in.grib "out_[shortName].grib"
```

```
> ls out_*
```

```
out_2t.grib  out_ms1.grib  ...
```

*Use quotes to
protect the []s*



- This provides a convenient way to filter GRIB messages into separate files

grib_set – set key / value pairs

- Use `grib_set` to
 - Set key / value pairs in the input GRIB file
 - Make simple changes to key / value pairs in the input GRIB file
- Each GRIB message is written to the output file
 - By default this includes messages for which no keys are changed
 - With `-s` (strict) option **only** messages matching **all constraints** in the where clause are copied
- An option exists to repack data
 - Sometimes after setting some keys involving properties of the packing algorithm the data needs to be repacked
- `grib_set` **fails** when an error occurs
 - e.g. when a key is not found

`grib_set` – usage

```
grib_set [options] grib_file grib_file ... out_grib_file
```

● Options

`-s key[:{s|l|d}]=value,...`

List of key / values to set

`-p key[:{s|l|d}],...`

Keys to print (only with `-v`)

`-w key{=/{!=}value,...`

Where option

`-d value`

Set all data values to `value`

`-f`

Do *not* fail on error

`-v`

Verbose

`-S`

Strict

`-r`

Repack data

...

grib_set – examples

- To set the parameter value of a field to 10m wind speed (10si)

```
> grib_set -s shortName=10si in.grib1 out.grib1
```

- This changes e.g.
 - `shortName` to 10si
 - `paramId` to 207
 - `name` / `parameterName` to '10 metre wind speed'
 - `units` / `parameterUnits` to 'm s ^{**} -1'

 - `indicatorOfParameter` to 207
 - `marsParam` to 207.128

grib_set – examples

- Some keys are read-only and cannot be changed directly

```
> grib_set -s name="10m wind speed" in.grib1 out.grib1
```

```
GRIB_API ERROR      : grib_set_values[0] name (3) failed:  
Value is read only
```

- The read-only keys can only be set by setting one of the other keys, e.g.
 - `shortName=10si`
 - `paramId=207`
 - `indicatorOfParameter=207` **GRIB edition dependent !**

grib_set – modify data values

- An offset can be added to all data values in a GRIB message by setting the key `offsetValuesBy`

```
> grib_get -F "%.5f" -p max,min,average TK.grib  
315.44727 216.96680 286.34257
```

```
> grib_set -s offsetValuesBy=-273.15 TK.grib TC.grib
```

```
> grib_get -F "%.5f" -p max,min,average TC.grib  
42.29726 -56.18321 13.19257
```

grib_set – modify data values

- The data values in a GRIB message can be multiplied by a factor by setting the key `scaleValuesBy`

```
> grib_get -F "%.2f" -p max,min,average Z.grib  
65035.92 -3626.08 2286.30
```

```
> grib_set -s scaleValuesBy=0.102 Z.grib1 orog.grib1
```

```
> grib_get -F "%.2f" -p max,min,average orog.grib1  
6633.64 -369.86 233.20
```

grib_set – using key values in output file

- Key values can be used to specify the output file

```
> grib_set -s time=0000 in.grib "out_[shortName].grib"
```

```
> ls out_*
```

```
out_2t.grib  out_msl.grib ...
```

- Remember: Use quotes to protect the []s !

What **cannot** be done with `grib_set`

- `grib_set` cannot be used for making transformations to the data representation
 - It cannot be used to transform data from spectral to grid-point representation (and vice-versa)
- `grib_set` cannot be used transform data from one grid representation to another
 - It cannot be used to transform data from regular or reduced Gaussian grids to regular latitude-longitude grids
- `grib_set` cannot be used to select sub-areas of data
 - It will change the value of, e.g. `latitudeOfFirstGridPointInDegrees` etc, but the data will still be defined on the original grid
- The GRIB tools cannot be used to interpolate the data

`grib_to_netcdf` – convert to netCDF

- Use `grib_to_netcdf` to convert GRIB messages to netCDF
- Input GRIB fields must be on a regular grid
 - `typeOfGrid=regular_ll` or `regular_gg`
- Options allow user to specify the netCDF data type:
 - `NC_BYTE`, `NC_SHORT`, `NC_INT`, `NC_FLOAT` or `NC_DOUBLE`
 - `NC_SHORT` is the default
- Options allow the user to specify the reference date
 - Default is 19000101
- Used in the MARS web interface and the public Data Servers to provide data in netCDF

`grib_to_netcdf` – usage

```
grib_to_netcdf [options] grib_file grib_file ...
```

● Options

`-o output_file`

Output netCDF file (required)

`-R YYYYMMDD`

Use YYYYMMDD as reference date

`-D NC_DATATYPE`

netCDF data type

`-f`

Do *not* fail on error

...

grib_to_netcdf – examples

- To convert the fields in file.grib1 to netCDF

```
> grib_to_netcdf -o out.nc file.grib1
grib_to_netcdf: Processing input file 'file.grib1'.
grib_to_netcdf: Found 1 GRIB fields in 1 file.
grib_to_netcdf: Creating netcdf file 'out.nc'
grib_to_netcdf: NetCDF library version: "3.6.3" of
  Aug 31 2010 17:29:46 $
grib_to_netcdf: Defining variable 't'.
grib_to_netcdf: Done.

> ls -s out.nc
132 out.nc
```

grib_to_netcdf – examples

- To convert the fields in file.grib1 to netCDF with data type set to **NC_FLOAT**

```
> grib_to_netcdf -D NC_FLOAT -o out.nc file.grib1
grib_to_netcdf: Processing input file 'file.grib1'.
grib_to_netcdf: Found 1 GRIB field in 1 file.
grib_to_netcdf: Creating netcdf file 'out.nc'
grib_to_netcdf: NetCDF library version: "3.6.3" of
  Aug 31 2010 17:29:46 $
grib_to_netcdf: Defining variable 't'.
grib_to_netcdf: Done.
```

```
> ls -s out.nc
```

```
260 out.nc
```

Output netCDF file is about twice the size

Practical 4: modifying GRIB messages

1. The file `tz_an_pl.grib1` contains parameters T and Z on five pressure levels. Use `grib_copy` to create two files, one containing all the pressure levels for parameter T, the other for Z. Check the content of the new files with `grib_ls`
2. Use `grib_ls` to inspect the contents of `tp.grib`. What is the parameter set to? Use `grib_set` to change the parameter for the message in the file `tp.grib` to total precipitation ('tp' – parameter 228). Check the new message with `grib_ls`.
3. Use `grib_to_netcdf` to convert the GRIB messages in `file1.grib` to netCDF. Try with both the default data type (`NC_SHORT`) and `NC_FLOAT`. Check the data values in each case with `ncdump`.
4. Use `grib_to_netcdf` to convert the GRIB messages in `file2.grib` to netCDF. What happens ... and why?

GRIB API user interfaces

- For some processing it is more convenient – or even necessary – to write a program
- The GRIB API library supports three user interfaces:
 - C: `#include <grib_api.h>`
 - Fortran 90 interface: `use grib_api`
 - Python interface: `import gribapi`
- At ECMWF two environment variables `GRIB_API_INCLUDE` and `GRIB_API_LIB` are defined to aid compilation and linking of Fortran 90 and C programs
- On ecgate:

```
gcc myprog.c $GRIB_API_INCLUDE $GRIB_API_LIB -lm
```

```
gfortran myprog.f90 $GRIB_API_INCLUDE $GRIB_API_LIB
```

General framework

- A (Fortran) code will generally include the following steps:
 - Open one or more GRIB files (for read or write)
 - Standard Fortran calls **cannot** be used to open or close a GRIB file. You **have to** call **`grib_open_file`** / **`grib_close_file`**
 - Calls to load one or more GRIB messages into memory
 - These subroutines will return a unique **`grib identifier`** which can be used to manipulate the loaded GRIB messages
 - Calls to encode / decode the loaded GRIB messages
 - Only **loaded** GRIB messages can be encoded / decoded
 - You should encode / decode only 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

Specifics of the GRIB API F90 interface

- Only subroutine names starting with `grib_`
 - `grib_get`, `grib_set`, `grib_new_from_file`, etc ...
- **All** routines have an optional argument for error handling:

```
subroutine grib_new_from_file(ifile, igrib, status)
  integer, intent (in)           :: ifile
  integer, intent (out)          :: igrib
  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
- Use **status** to handle errors yourself (e.g. necessary for MPI parallel codes)

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

Input arguments

Output arguments

Loading / Releasing a GRIB message (1/2)

- GRIB API can decode only loaded GRIB messages
- Two main subroutines to load a GRIB message for decoding

- `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

Input arguments

Output arguments

- `grib_new_from_index` (`indexid`, `igrib`)

Loads a GRIB message from an index

This index will first have been built

Loading / Releasing a GRIB message (2/2)

- These subroutines return a **unique grib identifier** (*igrib*)
 - Loaded messages are manipulated through this identifier
- **You do not have access to the buffer containing the loaded GRIB message**
 - This buffer is **internal** to the GRIB API 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 buffer message.

Input arguments


Output arguments

Example – Load from file

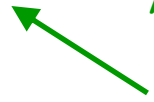
Input arguments
Output arguments

```
1 PROGRAM load_message
2 USE grib_api
3 IMPLICIT NONE
4
5 INTEGER :: rfile, igrib
6 CHARACTER(LEN=256), PARAMETER :: input_file='input.grb'
7 CHARACTER(LEN=10), PARAMETER :: open_mode='r'
8
9 !
10 ! Open GRIB data file for reading.
11 !
12 CALL grib_open_file(rfile, input_file, open_mode)
13
14 CALL grib_new_from_file(rfile, igrib)
15
16
17 CALL grib_release (igrib)
18 CALL grib_close_file (rfile)
19 END PROGRAM load_message
```

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



*Unique link to the buffer
loaded in memory. Calls to
grib_get subroutine are
needed to access and
decode this message*



GRIB
Message



Decoding a loaded GRIB message

- The idea is to decode as little as possible !
- You will **never** decode all the loaded GRIB message
 - use `grib_dump` for this !
- One subroutine for decoding:

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

Input arguments
Output arguments

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

Fortran example – grib_get

Input arguments
Output arguments

! Load all the GRIB messages contained in file.grib1

call `grib_open_file`(ifile, 'file.grib1','r')

call `grib_new_from_file`(ifile,igrib, iret)

LOOP: do while (iret /= GRIB_END_OF_FILE)

! Decode/encode data from the loaded message

call `grib_get`(igrib , “dataDate”, date)

call `grib_get`(igrib, “typeOfLevel”, levtype)

call `grib_get`(igrib, “level”, level)

call `grib_get_size`(igrib, “values”, nb_values)

allocate(values(nb_values))

call `grib_get`(igrib, “values”, values)

Values is declared as

real, dimension(:), allocatable:: values

print*, date, levtype, level, values(1), values(nb_values)

! Release

deallocate(values)

call `grib_release`(igrib)

Release the memory!

! Next message

call `grib_new_from_file`(ifile,igrib, iret)

end do LOOP

call `grib_close_file`(ifile)

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

Python example – grib_get

Input arguments
Output arguments

```
#!/usr/bin/env python
import sys
from gribapi import *
```

```
# Load all the GRIB messages contained in file.grib1
```

```
ifile = open('file.grib1')
```

```
while 1:
```

```
    igrb = grib_new_from_file(ifile)
```

```
    if igrb is None: break
```

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

```
# Decode/encode data from the loaded message
```

```
date = grib_get( igrb , "dataDate")
```

```
levtype = grib_get(igrb, "typeOfLevel")
```

```
level = grib_get(igrb, "level")
```

```
values = grib_get_values(igrb)
```

```
print date, levtype, level, values[0], values[len(values)-1]
```

Values returned as an array

```
# Release
```

```
grib_release(igrb)
```

```
ifile.close()
```

Release the memory!

Practical 5: GRIB decoding with Fortran 90

- Work on `ecgate`
- The `practical5` directory contains the program `grib_api_demo.f90`, a `Makefile` and some data in `grib_file.grib`
- Build an executable and run with
 - > `make`
 - > `./grib_api_demo > output`
- Look at the GRIB contents in the output file. Use `grib_ls` and `grib_dump` to examine the file `grib_file.grib`
- Change the program, replacing the call to `grib_dump` with several calls to `grib_get` to decode the values for the edition, date, time, paramId (or shortName) and level
- Add your own 'WRITE' or 'PRINT' statements to output this information

GRIB API 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, etc ...
 - Subroutines to compute the latitude, longitude and values
 - `grib_get_data` *For lat/lon, Gaussian, reduced Gaussian grids. It is similar to the grib_get_data GRIB tool*
 - Subroutines to extract values
 - `grib_find_nearest`: extract values closest to given geographical points
 - `grib_get_element`: extract values from a list of indexes
 - Subroutines for indexed access
 - Usually much faster than sequential access for “random” access

GRIB decoding – summary

- Use GRIB Tools where possible
 - It is not always necessary to write a program !
- Use edition-independent keys
 - This will make the migration to GRIB 2 easier
- ECMWF introduced GRIB 2 encoding for all its model level fields in May 2011
- If you do need to write a program think carefully about how the fields are accessed
 - Indexed access can be much faster than sequential access
- If you want to learn more about GRIB API then we hold a course each year – GRIB API: library and tools

Documentation

- The WMO FM 92 GRIB Manuals can be obtained from www.wmo.int/pages/prog/www/WMOCodes.html
- The ECMWF GRIB API manual is available at <https://software.ecmwf.int/wiki/display/GRIB/Home/>
- The GRIB Tools are documented at <https://software.ecmwf.int/wiki/display/GRIB/GRIB+tools>
- GRIB API Fortran 90 interface: https://software.ecmwf.int/wiki/display/GRIB/Fortran+package+grib_api
- GRIB API examples <https://software.ecmwf.int/wiki/display/GRIB/GRIB+API+examples>