GRIB API Version 1.8.0
Reference Manual

Author: Enrico Fucile

Generated by Doxygen 1.5.4

Contents

Chapter 1

GRIB API

The grib_api is the application program interface developed at ECMWF to provide an easy and realiable way for encoding and decoding WMO FM-92 GRIB edition 1 and edition 2 messages.

With the grib_api library, that is written entirely in C, some command line tools are provided to give a quick way to manipulate grib data. Moreover a Fortran interface 90 is available giving access to the main features of the C library.

The library is designed to access and modify messages in both editions with the same function calls using a set of Grib API keys to access the coded information (examples: get.f90 set.f90, get.c, set.c, grib_get, grib_set).

The keys available for a message are different depending not only on the edition but also and mainly on the type of each message and the information it contains. A list of all the available keys in a message can be obtained dynamically using the library as shown in keys_iterator.c or using the Grib tools as shown in grib_dump or grib_keys.

GRIB API will replace the GRIBEX function and a table of conversion between the numeric encoding of GRIBEX and the alphanumeric keys of GRIB API is provided to help the migration.

To learn how to use the grib_api we recommend the user works through the Grib API examples.

Reference manuals are also provided for the C library (organized in C interface) and for the Fortran 90 interface.

Installation instructions are also provided.

1.1 Compiling and linking on ECMWF platforms

The grib API is installed on all systems at ECMWF with both its components: the library and the tools.

The latest version of the tools is always available in the system PATH so that users can begin using the tools immediately by typing directly the tool name (see tools reference).

The latest version of the library is also installed on any platform and it is available for linking through the following two environment variables: $GRIB_API_INCLUDE $GRIB_API_LIB.

Here is a short summary on how to compile and link on ECMWF systems:

- ecgate, hpce, hpcf

```
> xlc -o foo foo.c $GRIB_API_INCLUDE $GRIB_API_LIB -lm
```

```
> xlf90 -o foo foo.f90 $GRIB_API_INCLUDE $GRIB_API_LIB
```

- linux cluster (C programs)

```
> gcc -m32 -o foo foo.c $GRIB_API_INCLUDE $GRIB_API_LIB
```

- workstation (C programs)

```
> gcc -o foo foo.c $GRIB_API_INCLUDE $GRIB_API_LIB
```

- linux cluster,workstation (Fortran programs)

```
> use pgf90
> pgf90 -o foo foo.f90 $GRIB_API_INCLUDE $GRIB_API_LIB
```

1.2 grib_api installation

The grib_api installation is based on the standard configure utility. It is tested on several platforms and with several compilers. However for some platforms modifications to the installation engine may be required. If you encounter any problem during the installation procedure please send an e-mail with your problem to Software.Services@ecmwf.int.

The only required package for a standard installation is jasper which enables the jpeg2000 packing/unpacking algorithm. It is possible to build grib_api without jasper, by using the –disable-jpeg configure option, but to install a fully functional library, its download is recommended.

1.2.1 Standard installation

1. Download grib_api from here.
2. Unpack distribution:

```
> gunzip grib_api-X.X.X.tar.gz
> tar xf grib_api-X.X.X.tar
```

3. Create the directory where to install grib_api say *grib_api_dir*

```
> mkdir grib_api_dir
```

4. Run the configure in the grib_api-X.X.X

```
> cd grib_api-X.X.X
> ./configure --prefix=grib_api_dir
```

5. make, check and install

```
> make
...
> make check
...
> make install
...
```

1.3 Grib API keys

The GRIBEX routine used at ECMWF to encode and decode GRIB messages works on a number based table to retrive all the information from the message. This approach forces the user either to learn a code table or to use the documentation intensively. With grib_api a key name based access is provided so that all the information contained in the GRIB message is retrieved through alphanumeric names.

All the key names are built from the official WMO documentation on the GRIB edition 1 and 2 coding standard removing the spaces in the key description and capitalizing the initials so that the caption:

```
identification of originating generating centre
```

is transformed into the key name

```
identificationOfOriginatingGeneratingCentre
```

Some short names (aliases) are also provided, e.g. "centre" is an alias for identificationOfOriginatingGeneratingCentre. The names are always easily releated to the meaning of their value.

A different set of keys is available for each message because the content is different. It is easy to find the keys available in a message by using the GRIB tools (grib_dump) or the library (keys_iterator.c).

1.3.1 Coded and Computed keys

There are two different types of keys: coded and computed.

The coded keys are directly linked to octets of the GRIB message and their value is obtained by only decoding the octets. A list of all the coded keys in a message can be obtained using grib_dump without any option (use the -a option to obtain also their aliases).

The computed keys are obtained by combining other keys (coded or computed) and when their value is set all the related keys are set in a cascade process.

These keys provide a synthesis of the information contained in the GRIB message and are a safe way to set complex attributes such as the type of grid or the type of packing. They are also helpful in the interpretation of some octets such as the scanning mode whose bits are related to the way of scanning the grid. In this case the computed keys:

```
iScansNegatively
jScansPositively
jPointsAreConsecutive
alternativeRowScanning (available only for edition 2)
```

will provide access to single bits of the scanning mode octect hiding its structure from the user.

The keys can also have some attributes as *read only*, which means that the key cannot be set (e.g. 7777 at the end of the message), or *edition specific* that is the attribute of all the keys having different values in the two editions (e.g. longitudeOfFirstGridPoint) or being present in one edition only (e.g. alternativeRowScanning).

Moreover there are some computed keys that cannot be "get" and can be considered as functions acting on the grib in some way. These keys are always characterised by a predicate in their name (e.g. setDecimalPrecision).

For the computed keys we provide the following preliminary documentation that will be extended soon.

- MARS keywords.

  All MARS keywords are available. Some examples are:

  - date
  - param
  - levtype
  - levelist
  - step
  - stream

- angles in degrees.

  All the angle variables are provided in two versions, a native one with the units coded into the GRIB file and an edition independent one in degrees. It is always better to work with the "in degrees" version that is always provided through the key which has the same name of the native version with the suffix InDegrees

```
longitudeOfFirstGridPoint -> longitudeOfFirstGridPointInDegrees
latitudeOfFirstGridPoint -> latitudeOfFirstGridPointInDegrees
longitudeOfLastGridPoint -> longitudeOfLastGridPointInDegrees
latitudeOfFirstGridPoint -> latitudeOfLastGridPointInDegrees
latitudeOfFirstGridPoint -> latitudeOfFirstGridPointInDegrees
iDirectionIncrement -> iDirectionIncrementInDegrees
jDirectionIncrement -> jDirectionIncrementInDegrees
```

- gridType

  The type of grid computed from the grid description section.

  - For both editions:
    - regular_ll
    - reduced_ll
    - mercator
    - lambert
    - polar_stereographic
    - UTM
    - simple_polyconic
    - albers
    - miller
    - rotated_ll
    - stretched_ll
    - stretched_rotated_ll
    - regular_gg
    - rotated_gg
    - stretched_gg
    - stretched_rotated_gg
    - reduced_gg
    - sh
    - rotated_sh
    - stretched_sh
    - stretched_rotated_sh
    - space_view

  - For edition 2 only:
    * triangular_grid
    * equatorial_azimuthal_equidistant
    * azimuth_range
    * cross_section
    * Hovmoller
    * time_section

- packingType

  The alghorithm used to pack data into the GRIB message.

  - For GRIB edition 1:
    * grid_simple
    * grid_simple_matrix
    * grid_simple_matrix_bitmap
    * grid_second_order
    * grid_second_order_different_width
    * spectral_complex
    * spectral_simple
    * grid_unknown
    * spectral_unknown
  - For GRIB edition 2:
    * grid_simple
    * grid_simple_matrix
    * grid_simple_matrix_bitmap
    * grid_complex
    * grid_complex_spatial_differencing
    * grid_jpeg
    * grid_png
    * grid_ieee
    * spectral_simple
    * spectral_complex
    * grid_simple_log_preprocessing

- setDecimalPrecision

  is a function key used to set the decimal precision see the grib_set page for usage.

- getNumberOfValues

  The number of values coded into the data section of the GRIB message

Chapter 2

Tools

2.1 Grib tools

The following command line tools are provided to help users in all interactive and batch processing of grib data.

Use of the tools is recommended whenever possible. They provide a ready and tested solution for many situations and their use will avoid the need to write new cod and thus speeding up your work.

To make easier their use the tools are provided with a common set of options so that it's quick to apply the same options to different tools. We suggest to begin with grib_dump, grib_ls and grib_get to inspect the content of some files and then to learn about the other tools to change the content of the grib message (grib_set, grib_convert, grib_filter) or to copy some messages from a file (grib_copy) or to get a latitude/longitude/values list of data. A smart compare tool (grib_compare) is also provided to compare grib messages focusing on some keys or comparing data with a given precision.

- grib_dump
- grib_ls
- grib_get
- grib_copy
- grib_set
- grib_convert
- grib_filter
- grib_compare
- grib_get_data
- grib_keys

2.2 grib_ls

2.2.1 DESCRIPTION

List content of grib files printing values of some keys. It does not fail when a key is not found.

2.2.2 USAGE

grib_ls [options] grib_file grib_file ...

2.2.3 OPTIONS

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

Declaration of keys to print. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be requested. Default type is string.

-F format

C style format for floating point values.

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

As -p adding the declared keys to the default list.

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

Where clause. Grib messages are processed only if they match all the key/value constraints. A valid constraint is of type key=value or key!=value. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be specified. Default type is string.

-B order by directive

Order by. The output will be ordered according the order by directive. Order by example: "step asc, centre desc" (step ascending and centre discending)

-l Latitude,Longitude[,MODE,file]

Value close to the point of a Latitude/Longitude. Allowed values for MODE are: 4 (4 values in the nearest points are printed) Default 1 (the value at the nearest point is printed) file (file is used as mask. The closer point with mask value>=0.5 is printed)

-i index

Data value corresponding to the given index is printed.

-n namespace

All the keys belonging to namespace are printed.

-m

Mars keys are printed.

-V

Version.

-W width

Minimum width of each column in output. Default is 10.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-7

Does not fail when the message has wrong length

2.2.4 grib_ls examples

1. Without options a default list of keys is printed.
   The default list is different depending on the type of grib message.

   ```
   > grib_ls ../data/reduced*.grib1 ../data/regular*.grib1 ../data/reduced*.grib2 \n
   ```

2. To print offset and count number in file use the keys offset and count
   Also the total count in a set of files is available as countTotal

   ```
   > grib_ls -p offset,count,countTotal ../data/reduced*.grib1
   ```

3. To list only a subset of messages use the -w (where option).
   Only the pressure levels are listed with the following line.

   ```
   > grib_ls -w levType=pl ../tigge_pf_ecmwf.grib2
   ```

4. All the grib messages not on pressure levels are listed as follows:

   ```
   > grib_ls -w levType!=pl ../tigge_pf_ecmwf.grib2
   ```

2.3 grib_get

2.3.1 DESCRIPTION

Get values of some keys from a grib file. It is similar to grib_ls, but fails returning an error code when an error occurs (e.g. key not found).

2.3.2 USAGE

grib_get [options] grib_file grib_file ...

2.3.3 OPTIONS

-f

Force. Force the execution not to fail on error.

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

Declaration of keys to print. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be requested. Default type is string.

-F format

C style format for floating point values.

-l Latitude,Longitude[,MODE,file]

Value close to the point of a Latitude/Longitude. Allowed values for MODE are: 4 (4 values in the nearest points are printed) Default 1 (the value at the nearest point is printed) file (file is used as mask. The closer point with mask value>=0.5 is printed)

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

As -p adding the declared keys to the default list.

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

Where clause. Grib messages are processed only if they match all the key/value constraints. A valid constraint is of type key=value or key!=value. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be specified. Default type is string.

-n namespace

All the keys belonging to namespace are printed.

-V

Version.

-W width

Minimum width of each column in output. Default is 10.

-m

Mars keys are printed.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-7

Does not fail when the message has wrong length

2.3.4 grib_get examples

1. grib_get fails if a key is not found.

```
>grib_get -p gribname ../data/tigge_pf_ecmwf.grib2
```

2. To get the step of the first GRIB message in a file:

```
>grib_get -w count=1 -p step ../data/tigge_pf_ecmwf.grib2
```

2.4 grib_get_data

2.4.1 DESCRIPTION

Print a latitude, longitude, data values list

2.4.2 USAGE

grib_get_data [options] grib_file grib_file ...

2.4.3 OPTIONS

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-m missingValue

The missing value is given through this option. Any string is allowed and it is printed in place of the missing values. Default is to skip the missing values.

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

Declaration of keys to print. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be requested. Default type is string.

-R key1=relative_error1,key2=relative_error2,...

Compare floating point values using the relative error as tolerance. key1=relative_error will compare key1 using relative_error1. all=relative_error will compare all the floating point keys using relative_error. Default all=0.

-F format

C style format for values. Default is "%.10e"

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

Where clause. Grib messages are processed only if they match all the key/value constraints. A valid constraint is of type key=value or key!=value. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be specified. Default type is string.

-f

Force. Force the execution not to fail on error.

-G

GRIBEX compatibility mode.

-7

Does not fail when the message has wrong length

-V

Version.

2.4.4 grib_get_data examples

1. To get a latitude, longitude, value list, skipping the missing values(=9999)

   ```
   >grib_get_data ../data/reduced_gaussian_model_level.grib2
   ```

2. If you want to define your missing value=1111 and to print the string missing in place of it

   ```
   >grib_get_data -m 1111:missing ../data/reduced_gaussian_model_level.grib2
   ```

3. If you want to print the value of other keys with the data value list

   ```
   >grib_get_data -p centre,level,step ../data/reduced_gaussian_model_level.grib2
   ```

2.5 grib_set

2.5.1 DESCRIPTION

Sets key/value pairs in the input grib file and writes each message to the output_grib_file. It fails when an error occurs (e.g. key not found).

2.5.2 USAGE

grib_set [options] grib_file grib_file ... output_grib_file

2.5.3 OPTIONS

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

Key/values to set. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be defined. By default the native type is set.

-r

Repack data. Sometimes after setting some keys involving properties of the packing algorithm a repacking of data is needed. This repacking is performed setting this -r option.

-d value

Set all the data values to "value".

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

Declaration of keys to print. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be requested. Default type is string.

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

As -p adding the declared keys to the default list.

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

Where clause. Set is only executed for grib messages matching all the key/value constraints. If a grib message does not match the constraints it is copied unchanged to the output_grib_file. This behaviour can be changed setting the option -S. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be defined. Default type is string.

-7

Does not fail when the message has wrong length

-S

Strict. Only grib messages matching all the constraints are copied to the output file

-V

Version.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-f

Force. Force the execution not to fail on error.

-v

Verbose.

2.5.4 grib_set examples

1. To set productDefinitionTemplateNumber=2 only for the fields with productDefinitionTemplateNumber=11

```
>grib_set -s productDefinitionTemplateNumber=2 -w productDefinitionTemplateNumber=11 ../data/tigge_pf
```

2. To set productDefinitionTemplateNumber=2 only for the fields for which productDefinitionTemplateNumber is not equal to 11

```
>grib_set -s productDefinitionTemplateNumber=2 -w productDefinitionTemplateNumber!=11 tigge_pf_ecmwf.
```

3. When a key is not used all the bits of its value should be set to 1 to indicate that it is missing. Since the length (number of octet) is different from a key to another, the value that we have to code for missing keys is not unique. To give an easy way to set a key to missing a string "missing" or "MISSING" is accepted by grib_set as follows:

```
>grib_set -s scaleFactorOfFirstFixedSurface=missing,scaledValueOfFirstFixedSurface=MISSING ../data/re
```

   Since some values can not be set to missing you can get an error for those keys.

4. To set scaleFactorOfSecondFixedSurface to missing only for the fields for which scaleFactorOfSecondFixedSurface is not missing:

```
>grib_set -s scaleFactorOfSecondFixedSurface=missing -w scaleFactorOfSecondFixedSurface!=missing tigg
```

5. It's possible to produce a grib edition 2 file from a grib edition 1 just changing the edition number with grib_set. At this stage of development all the geography parameters, level and time information is correctly translated, for the product definition extra set calls must be done. To do this properly grib_convert is suggested.

```
grib_set -s editionNumber=2 ../data/reduced_gaussian_pressure_level.grib1
```

6. With grib edition 2 is possible to compress data using the jpeg algorithm. To change packing algorithm from grid_simple (simple packing) to grid_jpeg (jpeg2000 packing):

```
>grib_set -s packingType=grid_jpeg ../data/regular_gaussian_model_level.grib2 out.grib2
```

7. It's possible to ask grib_api to calculate the number of bits per value needed to pack a given field with a fixed number of decimal digits of precision. For example if we want to pack a temperature expressed in Kelvin with 1 digits of precision after the decimal point we can set changeDecimalPrecision=1

```
>grib_set -s changeDecimalPrecision=1 ../data/regular_latlon_surface.grib2 ../data/out.grib2
rm -f ../data/out.grib2 | true
./grib_set -s changeDecimalPrecision=1 ../data/regular_latlon_surface.grib2 ../data/out.grib2
```

2.6 grib_dump

2.6.1 DESCRIPTION

Dump the content of a grib file in different formats.

2.6.2 USAGE

grib_dump [options] grib_file grib_file ...

2.6.3 OPTIONS

-O

Octet mode. WMO documentation style dump.

-D

Debug mode.

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

As -p adding the declared keys to the default list.

-d

Print all data values. Available only in C mode

-C

C code mode. A C code program generating the grib message is dumped.

-t

Print type information.

-H

Print octet content in hexadecimal format.

-a

Dump aliases.

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

Where clause. Grib messages are processed only if they match all the key/value constraints. A valid constraint is of type key=value or key!=value. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be specified. Default type is string.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-7

Does not fail when the message has wrong length

-V

Version.

-G

GRIBEX compatibility mode.

2.6.4 grib_dump examples

1. To dump in a WMO documentation style with hexadecimal octet values (-H)

```
>grib_dump -H ../data/reduced_gaussian_model_level.grib1
```

2. To obtain all the key names available in a grib file.

```
> grib_dump -D ../data/regular_latlon_surface.grib1
```

3. To obtain a C code example from a grib file.

```
>grib_dump -C ../data/regular_latlon_surface.grib1
#include <grib_api.h>

/* This code was generated automatically */


int main(int argc,const char** argv)
{
    grib_handle *h     = NULL;
    size_t size        = 0;
    double* vdouble    = NULL;
    long* vlong        = NULL;
    FILE* f            = NULL;
    const char* p      = NULL;
    const void* buffer = NULL;

    if(argc != 2) {
       fprintf(stderr,"usage: %s out\n",argv[0]);
        exit(1);
    }

    h = grib_handle_new_from_samples(NULL,"GRIB1");
    if(!h) {
        fprintf(stderr,"Cannot create grib handle\n");
        exit(1);
    }

    GRIB_CHECK(grib_set_long(h,"editionNumber",1),0);
    GRIB_CHECK(grib_set_long(h,"table2Version",128),0);

    /* 98 = European Center for Medium-Range Weather Forecasts (grib1/0.table)  */
    GRIB_CHECK(grib_set_long(h,"centre",98),0);

    GRIB_CHECK(grib_set_long(h,"generatingProcessIdentifier",130),0);
    GRIB_CHECK(grib_set_long(h,"gridDefinition",255),0);

    /* 128 = 10000000
    (1=1)  Section 2 included
    (2=0)  Section 3 omited
    See grib1/1.table */
    GRIB_CHECK(grib_set_long(h,"section1Flags",128),0);


    /* 167 = 2T 2 metre temperature K (grib1/2.98.128.table)  */
    GRIB_CHECK(grib_set_long(h,"indicatorOfParameter",167),0);


    /* 1 = Surface  (of the Earth, which includes sea surface)  (grib1/3.table)  */
    GRIB_CHECK(grib_set_long(h,"indicatorOfTypeOfLevel",1),0);
```

```
GRIB_CHECK(grib_set_long(h,"level",0),0);
GRIB_CHECK(grib_set_long(h,"yearOfCentury",8),0);
GRIB_CHECK(grib_set_long(h,"month",2),0);
GRIB_CHECK(grib_set_long(h,"day",6),0);
GRIB_CHECK(grib_set_long(h,"hour",12),0);
GRIB_CHECK(grib_set_long(h,"minute",0),0);

/* 1 = Hour (grib1/4.table)  */
GRIB_CHECK(grib_set_long(h,"unitOfTimeRange",1),0);

GRIB_CHECK(grib_set_long(h,"P1",0),0);
GRIB_CHECK(grib_set_long(h,"P2",0),0);

/* 0 = Forecast product valid at reference time + P1  (P1>0)  (grib1/5.table)  */
GRIB_CHECK(grib_set_long(h,"timeRangeIndicator",0),0);

GRIB_CHECK(grib_set_long(h,"numberIncludedInAverage",0),0);
GRIB_CHECK(grib_set_long(h,"numberMissingFromAveragesOrAccumulations",0),0);
GRIB_CHECK(grib_set_long(h,"centuryOfReferenceTimeOfData",21),0);

/* 0 = Unknown code table entry (grib1/0.ecmf.table)  */
GRIB_CHECK(grib_set_long(h,"subCentre",0),0);

GRIB_CHECK(grib_set_long(h,"decimalScaleFactor",0),0);

/* 1 = MARS labelling or ensemble forecast data (grib1/localDefinitionNumber.98.table)  */
GRIB_CHECK(grib_set_long(h,"localDefinitionNumber",1),0);


/* 1 = Operational archive (mars/class.table)  */
GRIB_CHECK(grib_set_long(h,"marsClass",1),0);


/* 2 = Analysis (mars/type.table)  */
GRIB_CHECK(grib_set_long(h,"marsType",2),0);


/* 1025 = Atmospheric model (mars/stream.table)  */
GRIB_CHECK(grib_set_long(h,"marsStream",1025),0);

p    = "0001";
size = strlen(p)+1;
GRIB_CHECK(grib_set_string(h,"experimentVersionNumber",p,&size),0);
GRIB_CHECK(grib_set_long(h,"perturbationNumber",0),0);
GRIB_CHECK(grib_set_long(h,"numberOfForecastsInEnsemble",0),0);
GRIB_CHECK(grib_set_long(h,"numberOfVerticalCoordinateValues",0),0);
GRIB_CHECK(grib_set_long(h,"pvlLocation",255),0);

/* 0 = Latitude/Longitude Grid (grib1/6.table)  */
GRIB_CHECK(grib_set_long(h,"dataRepresentationType",0),0);

GRIB_CHECK(grib_set_long(h,"Ni",16),0);
GRIB_CHECK(grib_set_long(h,"Nj",31),0);
GRIB_CHECK(grib_set_long(h,"latitudeOfFirstGridPoint",60000),0);
GRIB_CHECK(grib_set_long(h,"longitudeOfFirstGridPoint",0),0);

/* 128 = 10000000
(1=1)  Direction increments given
(2=0)  Earth assumed spherical with radius = 6367.47 km
(5=0)  u and v components resolved relative to easterly and northerly directions
See grib1/7.table */
GRIB_CHECK(grib_set_long(h,"resolutionAndComponentFlags",128),0);

GRIB_CHECK(grib_set_long(h,"latitudeOfLastGridPoint",0),0);
GRIB_CHECK(grib_set_long(h,"longitudeOfLastGridPoint",30000),0);
GRIB_CHECK(grib_set_long(h,"iDirectionIncrement",2000),0);
```

```
    GRIB_CHECK(grib_set_long(h,"jDirectionIncrement",2000),0);

    /* 0 = 00000000
    (1=0)  Points scan in +i direction
    (2=0)  Points scan in -j direction
    (3=0)  Adjacent points in i direction are consecutive
    See grib1/8.table */
    GRIB_CHECK(grib_set_long(h,"scanningMode",0),0);


    /* ITERATOR */


    /* NEAREST */

    GRIB_CHECK(grib_set_long(h,"bitsPerValue",16),0);
    GRIB_CHECK(grib_set_long(h,"sphericalHarmonics",0),0);
    GRIB_CHECK(grib_set_long(h,"complexPacking",0),0);
    GRIB_CHECK(grib_set_long(h,"integerPointValues",0),0);
    GRIB_CHECK(grib_set_long(h,"additionalFlagPresent",0),0);

    /* gribSection5 */

/* Save the message */

    f = fopen(argv[1],"w");
    if(!f) {
        perror(argv[1]);
        exit(1);
    }

    GRIB_CHECK(grib_get_message(h,&buffer,&size),0);

    if(fwrite(buffer,1,size,f) != size) {
        perror(argv[1]);
        exit(1);
    }

    if(fclose(f)) {
        perror(argv[1]);
        exit(1);
    }

    grib_handle_delete(h);
    return 0;
}
```

2.7 grib_dump examples

With the -O option you can get only the keys actually coded into the message, with the -a option the aliases of each key are printed. grib_dump -Oa "grib_file"

2.8 grib_debug

2.8.1 DESCRIPTION

Dump the content of a grib file in debug mode.

2.8.2 USAGE

grib_debug [options] grib_file grib_file ...

2.8.3 OPTIONS

-V

Version.

2.8.4 grib_debug examples

Dumping in a WMO documentation style with hexadecimal octet values (-H)

and with the aliases of each key listed in square brackets (-a).

grib_dump -Ha ../data/reduced_gaussian_model_level.grib1

```
***** FILE: ../data/reduced_gaussian_model_level.grib1
======================   MESSAGE 1 ( length=10142 )            ======================
======================   SECTION_0 ( length=0, padding=0 )     ======================
1-4       identifier = GRIB
5-7       totalLength = 10142 ( 0x00 0x27 0x9E )
8         editionNumber = 1 ( 0x01 ) [ls.edition]
======================   SECTION_1 ( length=52, padding=0 )    ======================
1-3       section1Length = 52 ( 0x00 0x00 0x34 )
4         gribTablesVersionNo = 128 ( 0x80 ) [table2Version]
5         identificationOfOriginatingGeneratingCentre = 98 ( 0x62 ) [European Center for Medium-Range Weat
6         generatingProcessIdentifier = 128 ( 0x80 ) [generatingProcessIdentificationNumber, process]
7         gridDefinition = 255 ( 0xFF )
8         section1Flags = 128 [10000000]
9         indicatorOfParameter = 130 ( 0x82 ) [T Temperature K (grib1/2.98.128.table) ]
10        indicatorOfTypeOfLevel = 109 ( 0x6D ) [Hybrid level level number (2 octets) (grib1/3.table) ] [l
11-12     lev = 1 ( 0x00 0x01 ) [topLevel, bottomLevel, ls.level, mars.levelist]
13        yearOfCentury = 7 ( 0x07 )
14        month = 3 ( 0x03 )
15        day = 18 ( 0x12 )
16        hour = 12 ( 0x0C )
17        minute = 0 ( 0x00 )
18        indicatorOfUnitOfTimeRange = 1 ( 0x01 ) [Hour (grib1/4.table) ]
19        periodOfTime = 0 ( 0x00 ) [P1]
20        periodOfTimeIntervals = 0 ( 0x00 ) [P2]
21        timeRangeIndicator = 0 ( 0x00 ) [Forecast product valid at reference time + P1 (P1>0) (grib1/5.t
22-23     numberIncludedInAverage = 0 ( 0x00 0x00 )
24        numberMissingFromAveragesOrAccumulations = 0 ( 0x00 )
25        centuryOfReferenceTimeOfData = 21 ( 0x15 )
26        identificationOfOriginatingGeneratingSubCentre = 0 ( 0x00 ) [Absent (grib1/0.table) ] [subCentre
27-28     decimalScaleFactor = 2 ( 0x00 0x02 )
29-40     reservedNeedNotBePresent = 12 {
               00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
            } # pad reservedNeedNotBePresent
41        localDefinitionNumber = 1 ( 0x01 )
42        marsClass = 1 ( 0x01 ) [Operational archive (mars/class.table) ] [mars.class]
```

```
43        marsType = 2 ( 0x02 ) [Analysis (mars/type.table) ] [ls.dataType, mars.type]
44-45     marsStream = 1025 ( 0x04 0x01 ) [Atmospheric model (mars/stream.table) ] [mars.stream]
46-49     experimentVersionNumber = 0001 [mars.expver]
50        perturbationNumber = 0 ( 0x00 )
51        numberOfForecastsInEnsemble = 0 ( 0x00 )
52        padding_local1_1 = 1 {
                    00
                  } # pad padding_local1_1
======================   SECTION_2 ( length=896, padding=0 )   ======================
1-3       section2Length = 896 ( 0x00 0x03 0x80 )
4         numberOfVerticalCoordinateValues = 184 ( 0xB8 ) [NV, numberOfCoordinatesValues]
5         pvlLocation = 33 ( 0x21 )
6         dataRepresentationType = 4 ( 0x04 ) [Gaussian Latitude/Longitude Grid (grib1/6.table) ]
7-8       numberOfPointsAlongAParallel = MISSING ( 0xFF 0xFF ) [geography.Ni]
9-10      numberOfPointsAlongAMeridian = 64 ( 0x00 0x40 ) [geography.Nj]
11-13     latitudeOfFirstGridPoint = 87864 ( 0x01 0x57 0x38 ) [La1]
14-16     longitudeOfFirstGridPoint = 0 ( 0x00 0x00 0x00 ) [Lo1]
17        resolutionAndComponentFlags = 0 [00000000]
18-20     latitudeOfLastGridPoint = -87864 ( 0x81 0x57 0x38 ) [La2]
21-23     longitudeOfLastGridPoint = 357188 ( 0x05 0x73 0x44 ) [Lo2]
24-25     iDirectionIncrement = MISSING ( 0xFF 0xFF ) [Di]
26-27     numberOfParallelsBetweenAPoleAndTheEquator = 32 ( 0x00 0x20 )
28        scanningMode = 0 [00000000]
29-32     padding_grid4_1 = 4 {
                  00, 00, 00, 00
                } # pad padding_grid4_1
33-768    pv = (184,736) {
          0,    2.00004,    3.98083,    7.38719,    12.9083,    21.4136,    33.9529,    51.7466,
    76.1677,    108.716,    150.986,    204.637,    271.356,    352.824,    450.686,    566.519,
    701.813,    857.946,    1036.17,    1237.59,    1463.16,    1713.71,    1989.87,    2292.16,
     2620.9,     2976.3,    3358.43,     3767.2,    4202.42,    4663.78,    5150.86,    5663.16,
    6199.84,    6759.73,    7341.47,    7942.93,    8564.62,     9208.3,    9873.56,    10558.9,
    11262.5,    11982.7,    12713.9,    13453.2,      14192,    14922.7,    15638.1,    16329.6,
    16990.6,    17613.3,      18191,      18717,    19184.5,    19587.5,    19919.8,    20175.4,
    20348.9,    20434.2,    20426.2,      20319,      20107,    19785.4,    19348.8,    18798.8,
    18141.3,    17385.6,    16544.6,    15633.6,    14665.6,    13653.2,    12608.4,    11543.2,
    10471.3,    9405.22,    8356.25,    7335.16,    6353.92,     5422.8,    4550.21,    3743.46,
    3010.15,     2356.2,    1784.85,    1297.66,    895.194,    576.314,    336.772,    162.043,
    54.2083,    6.57563,    0.00316,          0,          0,          0,          0,          0,
          0,          0,          0,          0
... 84 more values
} # ibmfloat pv
769-896   pl = (64,128) {
        20,         27,         36,         40,         45,         50,         60,         64,
        72,         75,         80,         90,         90,         96,        100,        108,
       108,        120,        120,        120,        128,        128,        128,        128,
       128,        128,        128,        128,        128,        128,        128,        128,
       128,        128,        128,        128,        128,        128,        128,        128,
       128,        128,        128,        128,        120,        120,        120,        108,
       108,        100,         96,         90,         90,         80,         75,         72,
        64,         60,         50,         45,         40,         36,         27,         20
} # unsigned pl
======================   SECTION_4 ( length=9182, padding=0 )   ======================
1-3       section4Length = 9182 ( 0x00 0x23 0xDE )
4         dataFlag = 0 [00000000]
5-6       binaryScaleFactor = 0 ( 0x00 0x00 )
7-10      referenceValue = 17402.8
11        numberOfBitsContainingEachPackedValue = 12 ( 0x0C ) [nbp, numberOfBits, bitsPerValue]
12-9182   values = (6114,9171) {
   203.778,    203.468,    202.958,    202.348,    201.758,    201.278,    200.888,    200.558,
   200.268,    200.078,    200.068,    200.318,    200.808,    201.458,    202.138,    202.758,
   203.248,    203.588,    203.798,    203.878,    205.968,    205.418,    204.438,    203.218,
   202.008,    201.128,    200.708,    200.598,    200.478,    200.228,    199.908,    199.528,
   199.108,    198.708,    198.528,    198.748,    199.458,    200.488,    201.548,    202.478,
   203.358,    204.178,    204.808,    205.198,    205.508,    205.838,    206.068,    207.338,
   206.488,    205.198,    203.798,    202.548,    201.528,    200.848,    200.638,    200.818,
   201.028,    200.888,    200.308,    199.638,    199.228,    199.018,    198.738,    198.328,
```

```
   197.868,    197.358,    196.928,    196.858,    197.348,    198.368,    199.638,    200.758,
   201.538,    202.288,    203.338,    204.438,    205.158,    205.558,    205.938,    206.438,
   207.008,    207.468,    207.638,    207.178,    206.658,    205.398,    203.788,    202.468,
   201.338,    200.298,    199.938,    200.318,    200.608,    200.478,    200.008,    199.208,
   198.278,    197.708,    197.558,    197.318
... 6014 more values
} # data_g1simple_packing values
======================   SECTION_5 ( length=4, padding=0 )     ======================
1-4       7777 = 7777
```

How to obtain all the key names available in a grib file.

grib_dump -D ../data/regular_latlon_surface.grib1

```
***** FILE: ../data/regular_latlon_surface.grib1
======================   MESSAGE 1 ( length=1100 )             ======================
0-0 constant oneConstant = 1
0-0 constant oneMillionConstant = 1000000
0-0 offset_file offset = 0
0-0 count_file count = 1
0-0 count_total countTotal = 1
0-0 lookup kindOfProduct = 1196575042 [GRIB 1196575042 0-4]
0-0 lookup GRIBEditionNumber = 1 [? 1 7-1]
======> section GRIB (1100,1100,0)
   0-0 constant grib1divider = 1000
   0-0 constant ieeeFloats = 0
   0-0 transient dummy = 1
   ======> section section_0 (0,0,0)
      ----> label empty
   <===== section section_0
   0-4 ascii identifier = GRIB
   4-7 g1_message_length totalLength = 1100
   7-8 unsigned editionNumber = 1 [ls.edition]
   ======> section section_1 (52,52,0)
      8-8 constant ECMWF = 98
      8-8 position offsetSection1 = 8
      8-11 section_length section1Length = 52
      11-12 unsigned gribTablesVersionNo = 128 [table2Version]
      12-13 codetable identificationOfOriginatingGeneratingCentre = 98 [European Center for Medium-Range W
      13-14 unsigned generatingProcessIdentifier = 128 [generatingProcessIdentificationNumber, process]
      14-15 unsigned gridDefinition = 255
      15-16 codeflag section1Flags = 128 [10000000:(1=1)  Section 2 included;(2=0)  Section 3 omited:grib1
      16-17 codetable indicatorOfParameter = 167 [2T 2 metre temperature K (grib1/2.98.128.table) ]
      17-17 sprintf marsParam = 167.128 [mars.param, ls.param]
      17-18 codetable indicatorOfTypeOfLevel = 1 [Surface (of the Earth, which includes sea surface) (grib
      18-20 unsigned lev = 0 [topLevel, bottomLevel, ls.level, mars.levelist]
      20-21 unsigned yearOfCentury = 7
      21-22 unsigned month = 3
      22-23 unsigned day = 18
      23-24 unsigned hour = 12
      24-25 unsigned minute = 0
      25-25 constant second = 0
      25-26 codetable indicatorOfUnitOfTimeRange = 1 [Hour (grib1/4.table) ]
      26-27 unsigned periodOfTime = 0 [P1]
      27-28 unsigned periodOfTimeIntervals = 0 [P2]
      28-29 codetable timeRangeIndicator = 0 [Forecast product valid at reference time + P1 (P1>0) (grib1/
      29-31 unsigned numberIncludedInAverage = 0
      31-32 unsigned numberMissingFromAveragesOrAccumulations = 0
      32-33 unsigned centuryOfReferenceTimeOfData = 21
      33-34 codetable identificationOfOriginatingGeneratingSubCentre = 0 [Absent (grib1/0.table) ] [subCen
      34-36 signed decimalScaleFactor = 0
      36-36 transient setLocalDefinition = 0
      36-48 pad reservedNeedNotBePresent = 12 {
         00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
      } # pad reservedNeedNotBePresent
      48-48 g1date dataDate = 20070318 [mars.date, ls.date]
      48-48 evaluate year = 2007
```

```
   48-48 g1monthlydate monthlyDate = 20070301
   48-48 time dataTime = 1200 [mars.time]
   48-48 g1startstep marsStartStep = 0 [mars.startStep]
   48-48 g1endstep marsEndStep = 0 [mars.endStep]
   48-48 g1step marsStep = 0 [mars.step, ls.step, forecastTime]
   48-48 g1verificationdate verificationDate = 20070318
   48-48 g1monthlydate monthlyVerificationDate = 20070301
   48-48 g1day_of_the_year_date dayOfTheYearDate = 2007-078
   48-48 constant wrongPadding = 0
   48-48 constant localUsePresent = 1
   48-48 g1param parameter = 167
   48-49 unsigned localDefinitionNumber = 1
   ======> section localDefinition (11,11,0)
      ======> section mars_labeling (8,8,0)
         49-50 codetable marsClass = 1 [Operational archive (mars/class.table) ] [mars.class]
         50-51 codetable marsType = 2 [Analysis (mars/type.table) ] [ls.dataType, mars.type]
         51-53 codetable marsStream = 1025 [Atmospheric model (mars/stream.table) ] [mars.stream]
         53-57 ksec1expver experimentVersionNumber = 0001 [mars.expver]
         57-57 constant SimulationsOf30Days = s3
         57-57 constant TYPE_S3 = 22
      <===== section mars_labeling
      57-58 unsigned perturbationNumber = 0
      58-59 unsigned numberOfForecastsInEnsemble = 0
      59-60 pad padding_local1_1 = 1 {
         00
      } # pad padding_local1_1
   <===== section localDefinition
   60-60 transient centreForTableNumber = 98
   60-60 section_padding localExtensionPadding = 0 {}
   60-60 section_padding section1Padding = 0 {}
   60-60 padtoeven evenpadding_sec1 = 0 {}
   60-60 concept grib1_short_name = 2T [ls.short_name]
   60-60 concept grib1_name = 2_metre_temperature [name]
   60-60 concept grib1_units = K [units]
<===== section section_1
60-60 bit gridDescriptionSectionPresent = 1 [GDSPresent]
60-60 bit bitmapPresent = 0 [bitmapSectionPresent]
======> section section_2 (32,32,0)
   60-60 position offsetSection2 = 60
   60-63 section_length section2Length = 32
   63-64 unsigned numberOfVerticalCoordinateValues = 0 [NV, numberOfCoordinatesValues]
   64-64 constant neitherPresent = 255
   64-65 unsigned pvlLocation = 255
   65-66 codetable dataRepresentationType = 0 [Latitude/Longitude Grid (grib1/6.table) ]
   ======> section dataRepresentation (22,22,0)
      66-66 constant gridDefinitionTemplateNumber = 0
      66-68 unsigned numberOfPointsAlongAParallel = 16 [Ni]
      68-70 unsigned numberOfPointsAlongAMeridian = 31 [Nj]
      70-73 signed latitudeOfFirstGridPoint = 60000 [La1]
      73-73 scale latitudeOfFirstGridPointInDegrees = 60 [geography.laFirst]
      73-76 signed longitudeOfFirstGridPoint = 0 [Lo1]
      76-76 scale longitudeOfFirstGridPointInDegrees = 0 [geography.loFirst]
      76-77 codeflag resolutionAndComponentFlags = 128 [10000000:(1=1)  Direction increments given;(2=0
      77-77 bit ijDirectionIncrementGiven = 1 [iDirectionIncrementGiven, jDirectionIncrementGiven, DiGi
      77-77 bit earthIsOblate = 0
      77-77 bit resolutionAndComponentFlags3 = 0
      77-77 bit resolutionAndComponentFlags4 = 0
      77-77 bit uvRelativeToGrid = 0
      77-77 bit resolutionAndComponentFlags6 = 0
      77-77 bit resolutionAndComponentFlags7 = 0
      77-77 bit resolutionAndComponentFlags8 = 0
      77-80 signed latitudeOfLastGridPoint = 0 [La2]
      80-80 scale latitudeOfLastGridPointInDegrees = 0 [geography.laLast]
      80-83 signed longitudeOfLastGridPoint = 30000 [Lo2]
      83-83 transient longitudeOfLastGridPointG1to2 = 30000
      83-83 scale longitudeOfLastGridPointInDegrees = 30 [geography.loLast]
      83-85 unsigned iDirectionIncrement = 2000 [Di]
```

```
      85-87 unsigned jDirectionIncrement = 2000 [Dj]
      87-88 codeflag scanningMode = 0 [00000000:(1=0)  Points scan in +i direction;(2=0)  Points scan i
      88-88 bit iScansNegatively = 0
      88-88 bit jScansPositively = 0
      88-88 bit jPointsAreConsecutive = 0
      88-88 constant iScansPositively = 1
      88-88 bit scanningMode4 = 0
      88-88 bit scanningMode5 = 0
      88-88 bit scanningMode6 = 0
      88-88 bit scanningMode7 = 0
      88-88 bit scanningMode8 = 0
      88-88 latlon_increment jDirectionIncrementInDegrees = 2 [geography.jInc, geography.gridNorthSouth
      88-88 latlon_increment iDirectionIncrementInDegrees = 2 [geography.iInc, geography.gridWestEast]
      ----> iterator ITERATOR
   <===== section dataRepresentation
   88-88 position endGridDefinition = 88
   88-88 transient PVPresent = 0
   88-88 position offsetBeforePV = 88
   88-88 position offsetBeforePL = 88
   88-88 transient PLPresent = 0 [reducedGrid]
   88-92 padto padding_sec2_1 = 4 {
      00, 00, 00, 00
   } # padto padding_sec2_1
   92-92 padtoeven padding_sec2_3 = 0 {}
<===== section section_2
92-92 position endOfHeadersMaker = 92
92-92 transient missingValue = 9999
92-92 constant tableReference = 0
======> section section_4 (1004,1004,0)
   92-92 position offsetSection4 = 92
   92-95 g1_section4_length section4Length = 1004
   95-95 g1_half_byte_codeflag halfByte = 8
   95-96 codeflag dataFlag = 8 [00001000:(1=0)  Grid-point data;(2=0)  Simple packing;(3=0)  Floating p
   96-98 signed binaryScaleFactor = -10
   98-102 ibmfloat referenceValue = 269.587
   102-103 unsigned numberOfBitsContainingEachPackedValue = 16 [nbp, numberOfBits, bitsPerValue]
   103-103 bit sphericalHarmonics = 0
   103-103 bit complexPacking = 0
   103-103 bit integerPointValues = 0
   103-103 bit additionalFlagPresent = 0
   ======> section dataValues (993,993,0)
      103-103 constant dataRepresentationTemplateNumber = 0
      103-103 position offsetBeforeData = 103
      103-103 constant bitMapIndicator = 255
      103-1096 data_g1simple_packing values = (496,993) {
            277.704,    277.797,    278.103,    274.598,    269.587,    278.345,    277.213,     278.19
            277.853,    276.747,    274.361,    273.636,    274.593,    273.782,    273.016,    274.316
            278.492,    278.792,    278.836,    278.333,    277.389,    278.525,    278.175,    277.255
            277.383,    278.047,    277.877,    276.213,     273.99,    278.333,     278.58,    277.642
            278.865,    278.997,    278.509,    278.983,    279.527,    279.414,      278.8,    278.749
            278.895,    279.056,    278.699,    278.426,    276.601,    277.491,    279.646,    279.198
            279.108,    279.156,    279.406,    279.527,    280.344,    280.869,    279.951,    281.621
            281.221,    280.676,    281.049,    280.354,    279.025,    278.192,     280.05,    280.375
             280.68,    281.269,    281.406,    281.483,    279.454,    280.641,    282.984,    282.578
            281.797,    281.542,    281.854,      281.5,    279.917,    280.529,    282.008,    281.102
            282.223,    282.727,    280.315,    278.539,    280.066,    280.789,    280.517,    282.883
            283.897,    285.161,    285.779,    285.847,    281.973,    282.869,    281.926,    280.816
             282.48,    281.894,    281.035,    281.722
         ... 396 more values
      } # data_g1simple_packing values
   <===== section dataValues
   1096-1096 size valuesCount = 496
   1096-1096 concept typeOfGrid = regular_ll [ls.gridType]
   1096-1096 concept typeOfPacking = grid_simple [ls.packingType, dataRepresentation]
   1096-1096 padtoeven padding_sec4_1 = 0 {}
<===== section section_4
======> section section_5 (4,4,0)
```

```
      ----> label gribSection5
      1096-1096 position offsetSection5 = 1096
      1096-1100 ascii 7777 = 7777
   <===== section section_5
<===== section GRIB
```

How to obtain a C code example from a grib file.

grib_dump -C ../data/regular_latlon_surface.grib1

```
#include <grib_api.h>

/* This code was generated automatically */


int main(int argc,const char** argv)
{
    grib_handle *h     = NULL;
    size_t size        = 0;
    double* v          = NULL;
    FILE* f            = NULL;
    const char* p      = NULL;
    const void* buffer = NULL;

    if(argc != 2) {
       fprintf(stderr,"usage: %s out\n",argv[0]);
        exit(1);
    }

    h = grib_handle_new_from_template(NULL,"GRIB2");
    if(!h) {
        fprintf(stderr,"Cannot create grib handle\n");
        exit(1);
    }


    /* empty */

    GRIB_CHECK(grib_set_long(h,"editionNumber",1),0);
    GRIB_CHECK(grib_set_long(h,"gribTablesVersionNo",128),0);

    /* 98 = European Center for Medium-Range Weather Forecasts (grib1/0.table)  */
    GRIB_CHECK(grib_set_long(h,"identificationOfOriginatingGeneratingCentre",98),0);

    GRIB_CHECK(grib_set_long(h,"generatingProcessIdentifier",128),0);
    GRIB_CHECK(grib_set_long(h,"gridDefinition",255),0);

    /* 128 = 10000000
    (1=1)  Section 2 included
    (2=0)  Section 3 omited
    See grib1/1.table */
    GRIB_CHECK(grib_set_long(h,"section1Flags",128),0);


    /* 167 = 2T 2 metre temperature K (grib1/2.98.128.table)  */
    GRIB_CHECK(grib_set_long(h,"indicatorOfParameter",167),0);


    /* 1 = Surface (of the Earth, which includes sea surface) (grib1/3.table)  */
    GRIB_CHECK(grib_set_long(h,"indicatorOfTypeOfLevel",1),0);

    GRIB_CHECK(grib_set_long(h,"lev",0),0);
    GRIB_CHECK(grib_set_long(h,"yearOfCentury",7),0);
    GRIB_CHECK(grib_set_long(h,"month",3),0);
    GRIB_CHECK(grib_set_long(h,"day",18),0);
    GRIB_CHECK(grib_set_long(h,"hour",12),0);
    GRIB_CHECK(grib_set_long(h,"minute",0),0);
```

```
/* 1 = Hour (grib1/4.table)  */
GRIB_CHECK(grib_set_long(h,"indicatorOfUnitOfTimeRange",1),0);

GRIB_CHECK(grib_set_long(h,"periodOfTime",0),0);
GRIB_CHECK(grib_set_long(h,"periodOfTimeIntervals",0),0);

/* 0 = Forecast product valid at reference time + P1 (P1>0) (grib1/5.table)  */
GRIB_CHECK(grib_set_long(h,"timeRangeIndicator",0),0);

GRIB_CHECK(grib_set_long(h,"numberIncludedInAverage",0),0);
GRIB_CHECK(grib_set_long(h,"numberMissingFromAveragesOrAccumulations",0),0);
GRIB_CHECK(grib_set_long(h,"centuryOfReferenceTimeOfData",21),0);

/* 0 = Absent (grib1/0.table)  */
GRIB_CHECK(grib_set_long(h,"identificationOfOriginatingGeneratingSubCentre",0),0);

GRIB_CHECK(grib_set_long(h,"decimalScaleFactor",0),0);
GRIB_CHECK(grib_set_long(h,"localDefinitionNumber",1),0);

/* 1 = Operational archive (mars/class.table)  */
GRIB_CHECK(grib_set_long(h,"marsClass",1),0);


/* 2 = Analysis (mars/type.table)  */
GRIB_CHECK(grib_set_long(h,"marsType",2),0);


/* 1025 = Atmospheric model (mars/stream.table)  */
GRIB_CHECK(grib_set_long(h,"marsStream",1025),0);

p    = "0001";
size = strlen(p)+1;
GRIB_CHECK(grib_set_string(h,"experimentVersionNumber",p,&size),0);
GRIB_CHECK(grib_set_long(h,"perturbationNumber",0),0);
GRIB_CHECK(grib_set_long(h,"numberOfForecastsInEnsemble",0),0);
GRIB_CHECK(grib_set_long(h,"numberOfVerticalCoordinateValues",0),0);
GRIB_CHECK(grib_set_long(h,"pvlLocation",255),0);

/* 0 = Latitude/Longitude Grid (grib1/6.table)  */
GRIB_CHECK(grib_set_long(h,"dataRepresentationType",0),0);

GRIB_CHECK(grib_set_long(h,"numberOfPointsAlongAParallel",16),0);
GRIB_CHECK(grib_set_long(h,"numberOfPointsAlongAMeridian",31),0);
GRIB_CHECK(grib_set_long(h,"latitudeOfFirstGridPoint",60000),0);
GRIB_CHECK(grib_set_long(h,"longitudeOfFirstGridPoint",0),0);

/* 128 = 10000000
(1=1)  Direction increments given
(2=0)  Earth assumed spherical with radius = 6367.47 km
(5=0)  u and v components resolved relative to easterly and northerly directions
See grib1/7.table */
GRIB_CHECK(grib_set_long(h,"resolutionAndComponentFlags",128),0);

GRIB_CHECK(grib_set_long(h,"latitudeOfLastGridPoint",0),0);
GRIB_CHECK(grib_set_long(h,"longitudeOfLastGridPoint",30000),0);
GRIB_CHECK(grib_set_long(h,"iDirectionIncrement",2000),0);
GRIB_CHECK(grib_set_long(h,"jDirectionIncrement",2000),0);

/* 0 = 00000000
(1=0)  Points scan in +i direction
(2=0)  Points scan in -j direction
(3=0)  Adjacent points in i direction are consecutive
See grib1/8.table */
GRIB_CHECK(grib_set_long(h,"scanningMode",0),0);
```

```
/* ITERATOR */


/* 8 = 00001000
(1=0)  Grid-point data
(2=0)  Simple packing
(3=0)  Floating point values are represented
(4=0)  No additional flags at octet 14
See grib1/11.table */
GRIB_CHECK(grib_set_long(h,"dataFlag",8),0);

GRIB_CHECK(grib_set_long(h,"numberOfBitsContainingEachPackedValue",16),0);
size = 496;
v    = (double*)calloc(size,sizeof(double));
if(!v) {
    fprintf(stderr,"failed to allocate %d bytes\n",size*sizeof(double));
    exit(1);
}

v[   0] = 277.704; v[   1] = 277.797; v[   2] = 278.103; v[   3] = 274.598;
v[   4] = 269.587; v[   5] = 278.345; v[   6] = 277.213; v[   7] =  278.19;
v[   8] = 277.853; v[   9] = 276.747; v[  10] = 274.361; v[  11] = 273.636;
v[  12] = 274.593; v[  13] = 273.782; v[  14] = 273.016; v[  15] = 274.316;
v[  16] = 278.492; v[  17] = 278.792; v[  18] = 278.836; v[  19] = 278.333;
v[  20] = 277.389; v[  21] = 278.525; v[  22] = 278.175; v[  23] = 277.255;
v[  24] = 277.383; v[  25] = 278.047; v[  26] = 277.877; v[  27] = 276.213;
v[  28] =  273.99; v[  29] = 278.333; v[  30] =  278.58; v[  31] = 277.642;
v[  32] = 278.865; v[  33] = 278.997; v[  34] = 278.509; v[  35] = 278.983;
v[  36] = 279.527; v[  37] = 279.414; v[  38] =   278.8; v[  39] = 278.749;
v[  40] = 278.895; v[  41] = 279.056; v[  42] = 278.699; v[  43] = 278.426;
v[  44] = 276.601; v[  45] = 277.491; v[  46] = 279.646; v[  47] = 279.198;
v[  48] = 279.108; v[  49] = 279.156; v[  50] = 279.406; v[  51] = 279.527;
v[  52] = 280.344; v[  53] = 280.869; v[  54] = 279.951; v[  55] = 281.621;
v[  56] = 281.221; v[  57] = 280.676; v[  58] = 281.049; v[  59] = 280.354;
v[  60] = 279.025; v[  61] = 278.192; v[  62] =  280.05; v[  63] = 280.375;
v[  64] =  280.68; v[  65] = 281.269; v[  66] = 281.406; v[  67] = 281.483;
v[  68] = 279.454; v[  69] = 280.641; v[  70] = 282.984; v[  71] = 282.578;
v[  72] = 281.797; v[  73] = 281.542; v[  74] = 281.854; v[  75] =   281.5;
v[  76] = 279.917; v[  77] = 280.529; v[  78] = 282.008; v[  79] = 281.102;
v[  80] = 282.223; v[  81] = 282.727; v[  82] = 280.315; v[  83] = 278.539;
v[  84] = 280.066; v[  85] = 280.789; v[  86] = 280.517; v[  87] = 282.883;
v[  88] = 283.897; v[  89] = 285.161; v[  90] = 285.779; v[  91] = 285.847;
v[  92] = 281.973; v[  93] = 282.869; v[  94] = 281.926; v[  95] = 280.816;
v[  96] =  282.48; v[  97] = 281.894; v[  98] = 281.035; v[  99] = 281.722;
v[ 100] = 279.978; v[ 101] = 284.138; v[ 102] = 287.234; v[ 103] = 287.831;
v[ 104] = 288.452; v[ 105] = 289.882; v[ 106] = 287.776; v[ 107] = 287.946;
v[ 108] = 281.466; v[ 109] = 284.771; v[ 110] = 283.343; v[ 111] = 282.477;
v[ 112] = 284.723; v[ 113] = 280.869; v[ 114] = 285.693; v[ 115] = 284.132;
v[ 116] = 276.881; v[ 117] = 283.388; v[ 118] = 287.295; v[ 119] = 286.764;
v[ 120] = 291.798; v[ 121] = 291.607; v[ 122] = 290.086; v[ 123] = 286.769;
v[ 124] =  284.24; v[ 125] = 280.884; v[ 126] = 286.866; v[ 127] = 284.694;
v[ 128] = 285.417; v[ 129] = 283.823; v[ 130] = 289.898; v[ 131] = 290.317;
v[ 132] = 287.031; v[ 133] = 287.949; v[ 134] = 289.263; v[ 135] = 289.869;
v[ 136] = 289.926; v[ 137] = 289.535; v[ 138] = 289.817; v[ 139] = 287.768;
v[ 140] = 290.394; v[ 141] = 290.294; v[ 142] = 287.069; v[ 143] = 281.759;
v[ 144] = 289.132; v[ 145] = 287.316; v[ 146] = 287.548; v[ 147] = 287.181;
v[ 148] = 287.645; v[ 149] = 289.492; v[ 150] = 288.956; v[ 151] = 286.634;
v[ 152] =   289.7; v[ 153] = 289.189; v[ 154] = 287.704; v[ 155] = 291.151;
v[ 156] = 286.208; v[ 157] = 291.093; v[ 158] = 284.818; v[ 159] = 282.097;
v[ 160] = 289.244; v[ 161] = 288.263; v[ 162] = 289.545; v[ 163] = 290.018;
v[ 164] = 289.881; v[ 165] = 290.215; v[ 166] = 289.999; v[ 167] = 289.447;
v[ 168] = 284.105; v[ 169] = 290.686; v[ 170] = 288.128; v[ 171] = 290.241;
v[ 172] = 289.116; v[ 173] = 289.576; v[ 174] =   291.8; v[ 175] =  286.35;
v[ 176] = 289.239; v[ 177] = 289.525; v[ 178] =  289.45; v[ 179] = 290.114;
v[ 180] = 290.301; v[ 181] = 289.429; v[ 182] = 290.005; v[ 183] = 287.195;
v[ 184] = 289.823; v[ 185] = 290.313; v[ 186] = 290.792; v[ 187] = 286.693;
v[ 188] = 291.941; v[ 189] = 290.783; v[ 190] = 290.818; v[ 191] = 287.234;
```

```
v[ 192] = 287.001; v[ 193] =  287.49; v[ 194] = 286.791; v[ 195] =  286.71;
v[ 196] = 287.182; v[ 197] =  290.49; v[ 198] = 290.322; v[ 199] = 289.957;
v[ 200] = 290.056; v[ 201] = 289.915; v[ 202] = 289.917; v[ 203] = 290.251;
v[ 204] = 290.502; v[ 205] = 290.782; v[ 206] = 291.367; v[ 207] = 291.025;
v[ 208] = 290.326; v[ 209] = 285.912; v[ 210] = 290.003; v[ 211] = 294.341;
v[ 212] = 294.048; v[ 213] = 291.771; v[ 214] = 290.675; v[ 215] = 291.203;
v[ 216] = 291.478; v[ 217] = 290.939; v[ 218] = 290.555; v[ 219] = 289.821;
v[ 220] = 290.126; v[ 221] = 291.021; v[ 222] = 291.243; v[ 223] = 290.761;
v[ 224] =  291.05; v[ 225] = 291.556; v[ 226] = 292.386; v[ 227] = 293.149;
v[ 228] = 293.301; v[ 229] = 291.821; v[ 230] = 290.157; v[ 231] = 293.427;
v[ 232] = 292.629; v[ 233] =  292.25; v[ 234] =  294.59; v[ 235] = 296.421;
v[ 236] =  296.16; v[ 237] = 290.221; v[ 238] = 290.882; v[ 239] = 290.864;
v[ 240] =  294.69; v[ 241] = 294.224; v[ 242] = 294.332; v[ 243] = 293.917;
v[ 244] = 292.863; v[ 245] = 293.005; v[ 246] = 292.814; v[ 247] = 295.443;
v[ 248] = 296.665; v[ 249] = 298.566; v[ 250] = 298.846; v[ 251] = 298.165;
v[ 252] = 297.105; v[ 253] = 294.729; v[ 254] = 294.968; v[ 255] = 293.305;
v[ 256] = 298.003; v[ 257] = 296.402; v[ 258] =  295.03; v[ 259] = 295.649;
v[ 260] = 295.811; v[ 261] = 297.203; v[ 262] = 298.222; v[ 263] =  297.12;
v[ 264] = 299.167; v[ 265] = 298.919; v[ 266] = 298.372; v[ 267] = 297.932;
v[ 268] =  296.47; v[ 269] = 295.208; v[ 270] = 294.647; v[ 271] = 294.034;
v[ 272] = 300.407; v[ 273] = 301.659; v[ 274] = 300.621; v[ 275] = 297.093;
v[ 276] = 295.676; v[ 277] = 298.434; v[ 278] = 298.906; v[ 279] = 302.369;
v[ 280] = 300.815; v[ 281] = 299.277; v[ 282] = 298.643; v[ 283] = 298.381;
v[ 284] = 296.632; v[ 285] = 294.887; v[ 286] = 295.411; v[ 287] = 293.665;
v[ 288] = 303.051; v[ 289] = 304.741; v[ 290] = 304.555; v[ 291] = 301.901;
v[ 292] = 301.846; v[ 293] = 300.793; v[ 294] = 302.141; v[ 295] = 300.521;
v[ 296] =  300.74; v[ 297] = 301.164; v[ 298] = 299.811; v[ 299] = 298.146;
v[ 300] = 298.443; v[ 301] = 293.905; v[ 302] = 295.545; v[ 303] = 296.185;
v[ 304] = 306.254; v[ 305] = 307.698; v[ 306] = 307.503; v[ 307] =  304.62;
v[ 308] = 304.458; v[ 309] = 303.097; v[ 310] =  303.69; v[ 311] = 303.482;
v[ 312] = 303.514; v[ 313] = 304.001; v[ 314] = 299.346; v[ 315] = 298.529;
v[ 316] = 297.935; v[ 317] = 295.495; v[ 318] = 295.846; v[ 319] = 296.122;
v[ 320] = 309.596; v[ 321] = 308.059; v[ 322] = 305.473; v[ 323] = 305.581;
v[ 324] =  306.11; v[ 325] = 303.994; v[ 326] = 304.602; v[ 327] = 304.286;
v[ 328] =  304.18; v[ 329] = 305.511; v[ 330] = 300.083; v[ 331] =  299.69;
v[ 332] = 297.061; v[ 333] = 296.252; v[ 334] = 296.508; v[ 335] = 298.427;
v[ 336] = 309.837; v[ 337] = 309.568; v[ 338] = 308.175; v[ 339] = 306.983;
v[ 340] = 307.399; v[ 341] = 303.002; v[ 342] = 303.582; v[ 343] = 303.765;
v[ 344] = 304.829; v[ 345] = 303.815; v[ 346] = 302.952; v[ 347] = 301.263;
v[ 348] = 296.397; v[ 349] = 298.184; v[ 350] = 297.765; v[ 351] = 299.807;
v[ 352] = 311.829; v[ 353] =  309.43; v[ 354] = 307.672; v[ 355] = 307.068;
v[ 356] = 306.384; v[ 357] = 304.862; v[ 358] = 304.397; v[ 359] = 303.944;
v[ 360] = 304.673; v[ 361] = 304.326; v[ 362] = 303.948; v[ 363] = 302.827;
v[ 364] = 297.377; v[ 365] = 296.722; v[ 366] = 298.711; v[ 367] = 300.744;
v[ 368] = 310.353; v[ 369] = 309.716; v[ 370] =  309.28; v[ 371] = 308.163;
v[ 372] = 306.711; v[ 373] =  305.75; v[ 374] =  304.74; v[ 375] = 305.384;
v[ 376] = 304.885; v[ 377] = 305.735; v[ 378] =  307.71; v[ 379] = 303.764;
v[ 380] = 303.073; v[ 381] =  300.87; v[ 382] = 300.858; v[ 383] = 302.205;
v[ 384] = 311.264; v[ 385] = 311.085; v[ 386] = 310.432; v[ 387] =  308.94;
v[ 388] = 305.619; v[ 389] =     307; v[ 390] = 306.413; v[ 391] = 307.649;
v[ 392] = 308.429; v[ 393] = 309.358; v[ 394] = 309.365; v[ 395] = 307.933;
v[ 396] =  306.15; v[ 397] = 305.126; v[ 398] = 305.611; v[ 399] = 303.336;
v[ 400] = 309.947; v[ 401] = 309.562; v[ 402] = 309.339; v[ 403] = 310.316;
v[ 404] = 308.055; v[ 405] = 307.565; v[ 406] = 310.605; v[ 407] =   308.4;
v[ 408] = 309.219; v[ 409] = 310.801; v[ 410] = 310.525; v[ 411] =  309.65;
v[ 412] = 306.611; v[ 413] = 306.033; v[ 414] = 307.988; v[ 415] = 308.941;
v[ 416] =   308.4; v[ 417] = 307.615; v[ 418] = 307.404; v[ 419] = 308.381;
v[ 420] = 309.778; v[ 421] = 311.715; v[ 422] = 308.409; v[ 423] = 307.156;
v[ 424] = 308.715; v[ 425] = 307.201; v[ 426] = 310.448; v[ 427] =  309.24;
v[ 428] = 306.716; v[ 429] = 307.307; v[ 430] = 309.062; v[ 431] = 309.776;
v[ 432] = 303.033; v[ 433] =  302.76; v[ 434] = 303.071; v[ 435] = 306.578;
v[ 436] = 309.819; v[ 437] = 305.046; v[ 438] = 309.764; v[ 439] = 307.857;
v[ 440] = 301.171; v[ 441] = 302.783; v[ 442] = 301.107; v[ 443] = 300.429;
v[ 444] = 303.189; v[ 445] = 304.585; v[ 446] = 303.709; v[ 447] = 307.132;
v[ 448] = 302.315; v[ 449] = 302.922; v[ 450] = 302.593; v[ 451] = 302.476;
v[ 452] = 302.132; v[ 453] = 305.953; v[ 454] = 300.132; v[ 455] = 301.361;
v[ 456] = 302.355; v[ 457] = 304.042; v[ 458] = 302.175; v[ 459] = 297.057;
```

```
    v[ 460] = 296.072; v[ 461] = 296.644; v[ 462] = 296.895; v[ 463] =  296.22;
    v[ 464] = 300.897; v[ 465] = 300.839; v[ 466] = 300.899; v[ 467] = 301.941;
    v[ 468] = 302.709; v[ 469] = 301.495; v[ 470] = 302.248; v[ 471] = 301.468;
    v[ 472] = 303.598; v[ 473] = 304.599; v[ 474] = 299.779; v[ 475] =   297.9;
    v[ 476] = 295.564; v[ 477] = 296.015; v[ 478] = 293.688; v[ 479] = 294.294;
    v[ 480] = 300.801; v[ 481] = 300.724; v[ 482] = 301.204; v[ 483] = 302.463;
    v[ 484] = 302.885; v[ 485] = 305.413; v[ 486] = 305.523; v[ 487] = 303.672;
    v[ 488] = 304.547; v[ 489] = 303.334; v[ 490] = 301.616; v[ 491] = 298.654;
    v[ 492] = 297.975; v[ 493] = 295.379; v[ 494] =  293.83; v[ 495] = 300.082;

    GRIB_CHECK(grib_set_double_array(h,"values",v,size),0);
    free(v);

    /* gribSection5 */

/* Save the message */

    f = fopen(argv[1],"w");
    if(!f) {
        perror(argv[1]);
        exit(1);
    }

    GRIB_CHECK(grib_get_message(h,&buffer,&size),0);

    if(fwrite(buffer,1,size,f) != size) {
        perror(argv[1]);
        exit(1);
    }

    if(fclose(f)) {
        perror(argv[1]);
        exit(1);
    }

    grib_handle_delete(h);
    return 0;
}
```

2.9 grib_convert

2.9.1 DESCRIPTION

It converts grib messages applying the rules from a conversion_rules file. The rules are of the type "keyname = value;" and if blocks are allowed as if (keyname1 == value1 || keyname2 != value2 && keyname3 == value3) { keyname4 = value4; }

2.9.2 USAGE

grib_convert [options] conversion_rules grib_file grib_file ... output_grib_file

2.9.3 OPTIONS

-f

Force. Force the execution not to fail on error.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-V

Version.

-7

Does not fail when the message has wrong length

-v

Verbose.

2.9.4 grib_convert examples

The following grib_convert rules convert all the grib messages contained in the input files in grib edition 2 and if a 2 metre temperature is found also the keys contained in the culy bracket are changed.

```
editionNumber = 2;
if( indicatorOfParameter == 11 && indicatorOfTypeOfLevel == 105)
{
    productDefinitionTemplateNumber = 1;
    typeOfFirstFixedSurface         = 103;
    scaleFactorOfFirstFixedSurface  = 0;
    scaledValueOfFirstFixedSurface  = 2;
}
```

2.10 grib_filter

2.10.1 DESCRIPTION

Apply the rules defined in rules_file to each grib message in the grib files provided as arguments.

2.10.2 USAGE

grib_filter [options] rules_file grib_file grib_file ...

2.10.3 OPTIONS

-f

Force. Force the execution not to fail on error.

-o output_grib_file

Output grib is written to ouput_grib_file. If an ouput grib file is required and -o is not used, theouput grib is written to filtered.out

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-V

Version.

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-7

Does not fail when the message has wrong length

-v

Verbose.

2.10.4 grib_filter examples

1. The rules accepted by grib_filter are different from the grib_convert rules due to the kind of work grib_filter it is supposed to do.

   The main difference between grib_filter and grib_convert is that the convert is a 1 field in input 1 field in output tool, while the filter is a 1 field in input as many field you need in output. At this aim the filter syntax allows a write in the form: write "filename". So that it is possible repeating as many write you need or using a parametrised write to send the output to many files.

   The grib_filter processes sequentially all the grib messages contained in the input file and it applies the rules to each one.

   Since the filename used in the write statement can contain some key values, taken from the grib processed when applying the "write rule", several files are produced in output containing fields with the same value of the keys used in the file name.

Indeed if we write a rules_file containing the only statement:

```
write "../data/split/[centre]_[date]_[dataType]_[levelType].grib[editionNumber]";
```

Applying this rules_file to the ../data/tigge_pf_ecmwf.grib2 grib file we obtain several files in the ../data/split directory containting fields splitted according their keys values

```
>grib_filter rules_file ../data/tigge_pf_ecmwf.grib2
>ls ../data/split
ecmf_20060619_pf_sfc.grib2
ecmf_20060630_pf_sfc.grib2
ecmf_20070122_pf_pl.grib2
ecmf_20070122_pf_pt.grib2
ecmf_20070122_pf_pv.grib2
ecmf_20070122_pf_sfc.grib2
```

2. The key values in the file name can also be obtained in a different format by indicating explicitly the type required after a colon.

   - :l for long
   - :d for double
   - :s for string

   The following statement works in a slightly different way from the previous example, including in the output file name the long values for centre and dataType.

   ```
   write "../data/split/[centre:l]_[date]_[dataType:l]_[levelType].grib[editionNumber]";
   ```

   Running the same command again we obtain a different list of files.

   ```
   >grib_filter rules_file ../data/tigge_pf_ecmwf.grib2
   >ls ../data/split
   98_20060619_4_sfc.grib2
   98_20060630_4_sfc.grib2
   98_20070122_4_pl.grib2
   98_20070122_4_pt.grib2
   98_20070122_4_pv.grib2
   98_20070122_4_sfc.grib2
   ```

3. Other statements are allowed in the grib_filter syntax:

   - if (condition) { statement;}
     The condition can be made using ==,!= and joining single block conditions with || and &&
     The statement can be any valid statement also another nested condition
   - set keyname = keyvalue;
   - print "string to print also with key values like in the file name"
   - transient keyname1 = keyname2;
   - comments beginning with #

   A complex example of grib_filter rules is the following to change temperature in a grib edition 1 file.

   ```
   # Temperature
   if ( level == 850 && indicatorOfParameter == 11 ) {
       print "found indicatorOfParameter=[indicatorOfParameter] level=[level] date=[date]";
       transient oldtype = type ;
       set identificationOfOriginatingGeneratingSubCentre=98;
       set gribTablesVersionNo = 128;
   ```

```
    set indicatorOfParameter = 130;
    set localDefinitionNumber=1;
    set marsClass="od";
    set marsStream="kwbc";
    # Negatively/Positively Perturbed Forecast
    if ( oldtype == 2 || oldtype == 3 ) {
      set marsType="pf";
      set experimentVersionNumber="4001";
    }
    # Control Forecast
    if ( oldtype == 1 ) {
      set marsType="cf";
      set experimentVersionNumber="0001";
    }
    set numberOfForecastsInEnsemble=11;
    write;
    print "indicatorOfParameter=[indicatorOfParameter] level=[level] date=[date]";
    print;
}
```

2.11 grib_compare

2.11.1 DESCRIPTION

Compare grib messages contained in two files. If some differences are found it fails returning an error code. Floating point values are compared exactly by default, different tolerance can be defined see -P -A -R. Default behaviour: absolute error=0, bit-by-bit compare, same order in files.

2.11.2 USAGE

grib_compare [options] grib_file grib_file

2.11.3 OPTIONS

-r

Compare files in which the messages are not in the same order. This option is time expensive.

-b key,key,...

All the keys in this list are skipped in the comparison. Bit-by-bit compare on.

-e

edition independent compare. It is used to compare grib edition 1 and 2.

-c key[:l/d/s/n],key[:l/d/s/n],...

Only the listed keys or namespaces (:n) are compared. The optional letter after the colon is used to force the type in the comparison: l->integer, d->float, s->string, n->namespace. See -a option. Incompatible with -H option.

-a

-c option modifier. The keys listed with the option -c will be added to the list of keys compared without -c.

-H

Compare only message headers. Bit-by-bit compare on. Incompatible with -c option.

-R key1=relative_error1,key2=relative_error2,...

Compare floating point values using the relative error as tolerance. key1=relative_error will compare key1 using relative_error1. all=relative_error will compare all the floating point keys using relative_error. Default all=0.

-A absolute error

Compare floating point values using the absolute error as tolerance. Default is absolute error=0

-P

Compare data values using the packing error as tolerance.

-T factor

Compare data values using factor multipied by the tolerance specified in options -P -R -A.

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

Where clause. Grib messages are processed only if they match all the key/value constraints. A valid constraint is of type key=value or key!=value. For each key a string (key:s) or a double (key:d) or a long

(key:l) type can be specified. Default type is string.

-f

Force. Force the execution not to fail on error.

-V

Version.

-7

Does not fail when the message has wrong length

-v

Verbose.

2.11.4 grib_compare examples

1. The default behaviour for grib_compare without any option is to perform a bit by bit comparison of the two messages. If the messages are found to be bitwise different then grib_compare switches to a "key based" mode to find out which coded keys are different. To see how grib_compare works we first set the shortName=2d (2 metre dew point temperature) in the file regular_latlon_surface.grib1

   ```
   >grib_set -s shortName=2d regular_latlon_surface.grib1 2d.grib1
   ```

   Then we can compare the two fields with grib_compare.

   ```
   >grib_compare regular_latlon_surface.grib1 2d.grib1

   -- GRIB #1 -- shortName=2t paramId=167 stepRange=0 levelType=sfc level=0 packingType=grid_simple gric
   long [indicatorOfParameter]: [167] != [168]
   ```

   In the output we see that the only "coded" key with different values in the two messages is indicatorOfParameter which is the relevant key for the parameter information. The comparison can be forced to be successful listing the keys with different values in the -b option.

   ```
   >grib_compare -b indicatorOfParameter regular_latlon_surface.grib1 2d.grib1
   ```

2. Two grib messages can be very different because they have different edition, but they can contain the same identical information in the header and the same data. To see how grib_compare can help in comparing messages with different edition we do

   ```
   >grib_set edition=2 reduced_gaussian_model_level.grib1 reduced_gaussian_model_level.grib2
   ```

   Then we compare the two fields with grib_compare.

   ```
   >grib_compare reduced_gaussian_model_level.grib1 reduced_gaussian_model_level.grib2

   -- GRIB #1 -- shortName=t paramId=130 stepRange=0 levelType=ml level=1 packingType=grid_simple gridTy
   long [totalLength]: [10908] != [10996]
   long [editionNumber]: [1] != [2]
   long [section1Length]: [52] != [21]
   [table2Version] not found in 2nd field
   [gridDefinition] not found in 2nd field
   [indicatorOfParameter] not found in 2nd field
   [indicatorOfTypeOfLevel] not found in 2nd field
   [yearOfCentury] not found in 2nd field
   [unitOfTimeRange] not found in 2nd field
   ```

```
[P1] not found in 2nd field
[P2] not found in 2nd field
[numberIncludedInAverage] not found in 2nd field
[numberMissingFromAveragesOrAccumulations] not found in 2nd field
[centuryOfReferenceTimeOfData] not found in 2nd field
[reservedNeedNotBePresent] not found in 2nd field
[localDefinitionNumber] not found in 2nd field
[perturbationNumber] not found in 2nd field
[numberOfForecastsInEnsemble] not found in 2nd field
[padding_local1_1] not found in 2nd field
long [section2Length]: [896] != [17]
[pvlLocation] not found in 2nd field
[dataRepresentationType] not found in 2nd field
long [latitudeOfFirstGridPoint]: [87864] != [87864000]
long [latitudeOfLastGridPoint]: [-87864] != [-87864000]
long [longitudeOfLastGridPoint]: [357188] != [357188000]
[padding_grid4_1] not found in 2nd field
long [section4Length]: [9948] != [770]
[dataFlag] not found in 2nd field
```

It is clear that the two messages are coded in a very different way. If we now add the -e option, the tool will compare only the higher level information common between the two messages.

```
>grib_compare -e reduced_gaussian_model_level.grib1 reduced_gaussian_model_level.grib2
```

The comparison is successful because the two messages contain the same information coded in two different ways. We can display the list of keys used by grib_compare adding the option -v (verbose).

```
>grib_compare -ve reduced_gaussian_model_level.grib1 reduced_gaussian_model_level.grib2
reduced_gaussian_model_level.grib2
  comparing centre as string
  comparing paramId as string
  comparing shortName as string
  comparing typeOfLevel as string
  comparing level as long
  comparing pv as double  (184 values) tolerance=0
  comparing latitudeOfFirstGridPointInDegrees as double  (1 values) tolerance=0.0005
  comparing longitudeOfFirstGridPointInDegrees as double  (1 values) tolerance=0.0005
  comparing latitudeOfLastGridPointInDegrees as double  (1 values) tolerance=0.0005
  comparing longitudeOfLastGridPointInDegrees as double  (1 values) tolerance=0.0005
  comparing iDirectionIncrementInDegrees is set to missing in both fields
  comparing iScansNegatively as long
  comparing jScansPositively as long
  comparing jPointsAreConsecutive as long
  comparing pl as long
  comparing gridType as string
  comparing packedValues as double  (6114 values) tolerance=0
  comparing param as string
  comparing levtype as string
  comparing levelist as long
  comparing date as long
  comparing time as long
  comparing step as long
  comparing class as long
  comparing type as long
  comparing stream as long
  comparing expver as string
  comparing domain as string

1 of 1 grib messages in reduced_gaussian_model_level.grib2

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

For each key the type used in the comparison is reported and for the floating point keys also the tolerance used is printed.

3. Some options are provided to compare only a set of keys in the messages. The option -H is used to compare only the headers coded in the message, it doesn't compare the data values. The option "-c key1:[l/d/s/n],key2:[l/d/s/n],... " can be used to compare a set of keys or namespaces. The letter after the colon is optional and it is used to force the type used in the comparison which is otherwise assumed to be the native type of the key. The possible types are:

   - :l -> integer (C type long)
   - :d -> floating point (C type double)
   - :s -> string
   - :n -> namespace.

   When the type "n" is used all the set of keys belonging to the specified namespace are compared assuming their own native type. To illustrate how these options work we change the values coded in a message using grib_filter with the following rules file (see grib_filter).

```
set bitsPerValue=10;
set values={1,2.5,3,4,5,6,70};
write "first.grib1";
set values={1,2.5,5,4,5,6,70};
write "second.grib1";
```

   We first compare the two files using the -H option (only headers are compared).

```
>grib_compare -H first.grib1 second.grib1
```

   The comparison is successful because the data are not compared. To compare only the data we have to compare the "data namespace".

```
>grib_compare -c data:n first.grib1 second.grib1

-- GRIB #1 -- shortName=t paramId=130 stepRange=0 levelType=ml level=1 packingType=grid_simple gridTy
double [packedValues]: 1 out of 7 different,  max absolute diff. = 2, relative diff. = 0.4
max diff. element 2: 3.00000000000000000000e+00 5.00000000000000000000e+00
tolerance=0 packingError: [0.04] [0.04]
values max= [70.04]  [70.04]         min= [1] [1]
```

   The comparison is showing that one of seven values is different in a comparison with the (default) absolute tolerance=0. We can change the tolerance with the -A option:

```
>grib_compare -A 2 -c data:n first.grib1 second.grib1
```

   and we see that the comparison is successful if the absolute tolerance is set to 2. We can also set the relative tolerance for each key with the option -R:

```
>grib_compare -R packedValues=0.4 -c data:n first.grib1 second.grib1
```

   and we get again a successful comparison because the relative tolerance is bigger than the relative absolute difference of two corresponding values. Another possible choice for the tolerance is to be equal to the packingError, which is the error due to the packing algorithm. If we change the decimalPrecision of a packed field we introduce a packing error sometimes bigger than the original packing error.

```
>grib_set -s changeDecimalPrecision=0 first.grib1 third.grib1
```

   and we compare the two fields using the -P option (tolerance=packingError).

```
>grib_compare -P -c data:n first.grib1 third.grib1
```

the comparison is successful because their difference is within the biggest of the two packing error. With the option -P the comparison is failing only if the original data coded are different, not if the packing precision is changed. If we try again to compare the fields without the -P option:

```
>grib_compare -c data:n first.grib1 third.grib1

-- GRIB #1 -- shortName=t paramId=130 stepRange=0 levelType=ml level=1 packingType=grid_simple gridTy
double [packedValues]: 4 out of 7 different,  max absolute diff. = 0.48, relative diff. = 0.16
max diff. element 1: 2.520000000000000001776e+00 3.000000000000000000000e+00
tolerance=0 packingError: [0.04] [0.5]
values max= [70.04]  [70]         min= [1] [1]
```

we see that some values are different and that the maximum absolute differenc is close to the biggest packing error (max diff=0.48 packingError=0.5). The packing error was chosen to be 0.5 by setting decimalPrecision to 0 which means that we don't need to preserve any decimal figure.

4. When we already know that the fields are not numerically identical, but have similar statistical characteristics we can compare their statistics namespaces:

```
>grib_compare -c statistics:n first.grib1 third.grib1

-- GRIB #1 -- shortName=t paramId=130 stepRange=0 levelType=ml level=1 packingType=grid_simple gridTy
double [max]: [7.004000000000000062528e+01] != [7.000000000000000000000e+01]
absolute diff. = 0.04, relative diff. = 0.000571102
tolerance=0
double [avg]: [1.309142857142857485e+01] != [1.314285714285714234960e+01]
absolute diff. = 0.0514286, relative diff. = 0.00391304
tolerance=0
double [sd]: [2.329946868098770096370e+01] != [2.325896798735349690900e+01]
absolute diff. = 0.0405007, relative diff. = 0.00173827
tolerance=0
double [skew]: [-1.415928757005156235490e+01] != [-1.416699713804938554060e+01]
absolute diff. = 0.00770957, relative diff. = 0.000544192
tolerance=0
double [kurt]: [7.323647107856595672710e-01] != [7.327237974894553751430e-01]
absolute diff. = 0.000359087, relative diff. = 0.000490071
tolerance=0
```

and we see that maximum, minimum, average, standard deviation, skewness and kurtosis are compared. While the values are different by 0.48 the statistics comparison shows that the difference in the statistical values is never bigger than 0.052

```
>grib_compare -A 0.052 -c statistics:n first.grib1 third.grib1
```

The statistics namespace is available also for spherical harmonics data and provides information about the field in the geographic space computing them in the spectral space for performance reasons.

5. When a file contains several fields and some keys are different, it is useful to have a summary report of the keys found different in the messages. This can be obtained with the option -f. We change few keys in a file:

```
>grib_set -w typeOfLevel=surface -s step=48 tigge_pf_ecmwf.grib2 out.grib2
```

and comparing with the -f option:

```
>grib_compare -f tigge_pf_ecmwf.grib2 out.grib2

-- GRIB #9 -- shortName=skt paramId=235 stepRange=96 levelType=sfc level=0 packingType=grid_simple gr
long [forecastTime]: [96] != [48]

-- GRIB #10 -- shortName=sd paramId=228141 stepRange=96 levelType=sfc level=0 packingType=grid_simple
long [forecastTime]: [96] != [48]

-- GRIB #11 -- shortName=sf paramId=228144 stepRange=0-96 levelType=sfc level=0 packingType=grid_simp
long [dayOfEndOfOverallTimeInterval]: [26] != [24]
long [lengthOfTimeRange]: [96] != [48]

...  output deleted

## ERRORS SUMMARY #######
##
## Summary of different key values
## forecastTime ( 3 different )
## dayOfEndOfOverallTimeInterval ( 11 different )
## lengthOfTimeRange ( 11 different )
##
## 14 different messages out of 38
```

we get a list of all the different messages in the files and a summary report of the different keys.

6. We can change the order of the messages in a file using grib_copy with the -B option:

```
>grib_copy -B typeOfLevel tigge_pf_ecmwf.grib2 out.grib2
```

If we now compare the two files:

```
>grib_compare -f tigge_pf_ecmwf.grib2 out.grib2

-- GRIB #1 -- shortName=10u paramId=165 stepRange=96 levelType=sfc level=10 packingType=grid_simple g
long [discipline]: [0] != [2]
long [totalLength]: [1555] != [990]
long [parameterCategory]: [2] != [0]
long [parameterNumber]: [2] != [22]
long [scaledValueOfFirstFixedSurface]: [10] != [0]
long [typeOfSecondFixedSurface]: [255] != [106]
scaleFactorOfSecondFixedSurface is set to missing in 1st field is not missing in 2nd field
scaledValueOfSecondFixedSurface is set to missing in 1st field is not missing in 2nd field
long [numberOfValues]: [684] != [239]
double [referenceValue]: [-1.5722932815551757812 5e+01] != [4.1584381103515625 0000e+01]
absolute diff. = 57.3073, relative diff. = 1.3781
tolerance=3.8147e-06
long [binaryScaleFactor]: [-10] != [-15]
long [bitsPerValue]: [16] != [24]
long [section6Length]: [6] != [92]
long [bitMapIndicator]: [255] != [0]
long [section7Length]: [1373] != [722]
[codedValues] has different size: 1st field: 684, 2nd field: 239
...    very long output
```

the comparison is failing because of the different order of the messages. We can use the -r option to compare the files assuming that the messages are not in the same order:

```
>grib_compare -r tigge_pf_ecmwf.grib2 out.grib2
```

and we have a successful comparison because for each message in the first file an identical message is found in the second file. This option should be used carefully as it is very time expensive.

2.12 grib_keys

2.12.1 DESCRIPTION

Lists the keys available for a type of grib (-T option) or in a grib message from a file (-F option).

2.12.2 USAGE

grib_keys [options]

2.12.3 OPTIONS

-T type

To print the keys available in the given grib type. For a list of the availeble types see -L option.

-F file

To print the keys available in the grib file.

-x

Print the extended set of keys.

-c

Print only coded keys.

-L

List of available types.

-t

Print type information.

-a

Dump aliases.

2.12.4 grib_keys examples

1. With the -L option a list of the available templates is printed

```
> grib_keys -L \n
GRIB1
GRIB2
reduced_gg_ml_grib2
reduced_gg_pl_grib1
reduced_gg_sfc_grib1
reduced_gg_ml_grib1
reduced_gg_pl_grib2
reduced_gg_sfc_grib2
reduced_gg_sfc_jpeg_grib2
reduced_ll_sfc_grib1
reduced_ll_sfc_grib2
regular_gg_ml_grib1
regular_gg_ml_grib2
regular_gg_pl_grib1
regular_gg_pl_grib2
regular_ll_sfc_grib1
```

```
regular_ll_sfc_grib2
regular_ll_pl_grib1
regular_ll_pl_grib2
sh_ml_grib1
sh_ml_grib2
sh_pl_grib1
sh_pl_grib2
```

2. To print the standard set of key available for a given type

```
> grib_keys -T regular_ll_sfc_grib1
=================== regular_ll_sfc_grib1
editionNumber
====> SECTION 1 <====
table2Version
originatingCentre
generatingProcessIdentifier
indicatorOfParameter
marsParam (read only)
indicatorOfTypeOfLevel
level
timeRangeIndicator
subCentre
decimalScaleFactor
dataDate
dataTime
stepUnits
stepRange
startStep
endStep
monthlyVerificationDate (read only)
dayOfTheYearDate (read only)
paramId
localDefinitionNumber
marsClass
marsType
marsStream
experimentVersionNumber
perturbationNumber
numberOfForecastsInEnsemble
name (read only)
bitmapPresent
====> SECTION 2 <====
numberOfVerticalCoordinateValues
Ni
Nj
latitudeOfFirstGridPointInDegrees
longitudeOfFirstGridPointInDegrees
earthIsOblate
uvRelativeToGrid
latitudeOfLastGridPointInDegrees
longitudeOfLastGridPointInDegrees
DjInDegrees
DiInDegrees
iScansNegatively
jScansPositively
jPointsAreConsecutive
alternativeRowScanning (read only)
numberOfDataPoints (read only)
numberOfValues (read only)
missingValue
====> SECTION 4 <====
binaryScaleFactor (read only)
referenceValue (read only)
bitsPerValue
sphericalHarmonics
complexPacking
```

```
integerPointValues
additionalFlagPresent
typeOfPacking
values
numberOfCodedValues (read only)
max (read only)
min (read only)
average (read only)
numberOfMissing (read only)
typeOfGrid
getNumberOfValues (read only)
====> SECTION 5 <====
```

2.13 grib_copy

2.13.1 DESCRIPTION

Copies the content of grib files printing values of some keys.

2.13.2 USAGE

grib_copy [options] grib_file grib_file ... output_grib_file

2.13.3 OPTIONS

-f

Force. Force the execution not to fail on error.

-r

Repack data. Sometimes after setting some keys involving properties of the packing algorithm a repacking of data is needed. This repacking is performed setting this -r option.

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

Declaration of keys to print. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be requested. Default type is string.

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

As -p adding the declared keys to the default list.

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

Where clause. Only grib messages matching the key/value constraints are copied to the output_grib_file. For each key a string (key:s) or a double (key:d) or a long (key:l) type can be defined. Default type is string.

-B order by directive

Order by. The output will be ordered according the order by directive. Order by example: "step asc, centre desc" (step ascending and centre discending)

-V

Version.

-W width

Minimum width of each column in output. Default is 10.

-M

Multi-grib support off. Turn off support for multiple fields in single grib message

-g

Copy GTS header.

-G

GRIBEX compatibility mode.

-7

Does not fail when the message has wrong length

-v

Verbose.

2.13.4 grib_copy examples

1. To copy only the pressure levels from a file

```
> grib_copy -w levtype=pl ../data/tigge_pf_ecmwf.grib2 out.grib
```

2. To copy only the fields that are not on pressure levels from a file

```
> grib_copy -w levtype!=pl ../data/tigge_pf_ecmwf.grib2 out.grib
```

3. A grib_file with multi field messages can be converted in single field messages with a simple grib_-copy.

```
> grib_copy multi.grib simple.grib
```

Chapter 3

Examples

3.1 Grib API examples

The main features of the grib_api are explained here through some simple examples that can be taken as a starting point to write more complex programs.

3.1.1 Fortran 90

- index.f90 how to access a grib file through an index.
- get.f90 how to get values through the key names.
- count_messages.f90 count the messages in a file and loop through them.
- get_pl.f90 how to get the list of number of points for each parallel in reduced grids.
- get_pv.f90 how to get the list of levels.
- get_data.f90 how to get latitude/longitude/values.
- set.f90 how to set values through the key names.
- set_bitmap.f90 how to set and use a bitmap.
- set_missing.f90 how to set a missing value in the header.
- set_pv.f90 how to set the list of levels.
- samples.f90 how to create a new message from a template.
- clone.f90 how to clone a message.
- copy_message.f90 how to copy a message in memory and create a new message.
- keys_iterator.f90 how to get the names of all the keys defined in a message and how to iterate through them.
- precision.f90 how to control precision when coding a grib field.
- multi_write.f90 how to encode a grib message containing many fields.
- multi.f90 how to decode a grib message containing many fields.

- print_data.f90 how to print all the data contained in a grib file.
- nearest.f90 how to find the nearest grid points.

3.1.2 C

- get.c is an example showing how to get values through the key names.
- set.c is an example illustrating how to set values through the key names.
- keys_iterator.c explains how to get the names of all the keys defined in a message and how to iterate through them.
- iterator.c shows how to use an iterator on latitude, longitude, values.
- precision.c illustrates how to control precision when coding a grib field.
- multi.c is an example describing how to decode a grib message containing many fields.
- print_data.c is an example on how to print all the data contained in a grib file.
- nearest.c is an example on how to find the nearest grid points.
- multi_write.c how to encode a grib message containing many fields.
- multi.c how to decode a grib message containing many fields.

3.1.3 Fortran 77

- get_fortran.F is an example showing how to get values through the key names.
- set_fortran.F is an example illustrating how to set values through the key names.
- keys_iterator_fortran.F explains how to get the names of all the keys defined in a message and how to iterate through them.
- iterator_fortran.F shows how to use an iterator on latitude, longitude, values.
- precision_fortran.F illustrates how to control precision when coding a grib field.
- multi_fortran.F is an example describing how to decode a grib message containing many fields.
- print_data_fortran.F is an example on how to print all the data contained in a grib file.

3.2 clone.f90

How to clone a message.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to create a new GRIB message by cloning
10 !               an existing message.
```

```
11 !
12 !
13 !  Author: Anne Fouilloux
14 !
15 !
16 program clone
17   use grib_api
18   implicit none
19   integer                                        :: err,i,iret
20   integer                                        :: nx, ny
21   integer                                        :: infile,outfile
22   integer                                        :: igrib_in
23   integer                                        :: igrib_out
24   character(len=2)                               :: step
25   double precision, dimension(:,:), allocatable :: field2D
26
27
28   call grib_open_file(infile,'../../data/constant_field.grib1','r')
29   call grib_open_file(outfile,'out.grib1','w')
30
31   !     a new grib message is loaded from file
32   !     igrib is the grib id to be used in subsequent calls
33   call grib_new_from_file(infile,igrib_in)
34
35   call grib_get(igrib_in,"numberOfPointsAlongAParallel", nx)
36
37   call grib_get(igrib_in,"numberOfPointsAlongAMeridian",ny)
38
39   allocate(field2D(nx,ny),stat=err)
40
41   if (err .ne. 0) then
42      print*, 'Failed to allocate ', nx*ny, ' values'
43      STOP
44   end if
45   ! clone the constant field to create 4 new GRIB messages
46   do i=0,18,6
47     call grib_clone(igrib_in, igrib_out)
48     write(step,'(i2)') i
49 ! Careful: stepRange is a string (could be 0-6, 12-24, etc.)
50 ! use adjustl to remove blank from the left.
51     call grib_set(igrib_out,'stepRange',adjustl(step))
52
53     call generate_field(field2D)
54
55 ! use pack to create 1D values
56     call grib_set(igrib_out,'values',pack(field2D, mask=.true.))
57
58   !     write cloned messages to a file
59     call grib_write(igrib_out,outfile)
60     call grib_release(igrib_out)
61   end do
62
63   call grib_release(igrib_in)
64
65   call grib_close_file(infile)
66
67   call grib_close_file(outfile)
68   deallocate(field2D)
69
70 contains
71 !=====================================
72 subroutine generate_field(gfield2D)
73  double precision, dimension(:,:) :: gfield2D
74
75  call random_number(gfield2D)
76 end subroutine generate_field
77 !=====================================
```

```
78
79 end program clone
```

3.3 copy_message.f90

How to copy a message in memory and create a new message.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to copy a message in memory
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program copy
16   use grib_api
17   implicit none
18   integer                             :: err, centre
19   integer(kind=kindOfSize)            :: byte_size
20   integer                             :: infile,outfile
21   integer                             :: igrib_in,iret
22   integer                             :: igrib_out
23   character(len=1), dimension(:), allocatable :: message
24
25
26   call grib_open_file(infile,'../../data/constant_field.grib1','r')
27   call grib_open_file(outfile,'out.grib1','w')
28
29   !     a new grib message is loaded from file
30   !     igrib is the grib id to be used in subsequent calls
31   call grib_new_from_file(infile,igrib_in)
32
33   call grib_get_message_size(igrib_in, byte_size)
34   allocate(message(byte_size), stat=err)
35
36   call grib_copy_message(igrib_in,message)
37
38   call grib_new_from_message(igrib_out, message)
39
40   centre=80
41   call grib_set(igrib_out,"centre",centre)
42
43 !  write messages to a file
44   call grib_write(igrib_out,outfile)
45
46   call grib_release(igrib_out)
47
48   call grib_release(igrib_in)
49
50   call grib_close_file(infile)
51   call grib_close_file(outfile)
52   deallocate(message)
53
54 end program copy
```

3.4 count_messages.f90

count the messages in a file and loop through them.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: count messages before processing
10 !
11 !  Author: Enrico Fucile
12 !
13 !
14 program get
15   use grib_api
16   implicit none
17
18   integer                             ::  ifile
19   integer                             ::  iret
20   integer                             ::  n
21   integer                             ::  i
22   integer,dimension(:),allocatable    ::  igrib
23   real                                ::  latitudeOfFirstPointInDegrees
24   real                                ::  longitudeOfFirstPointInDegrees
25   real                                ::  latitudeOfLastPointInDegrees
26   real                                ::  longitudeOfLastPointInDegrees
27   integer                             ::  numberOfPointsAlongAParallel
28   integer                             ::  numberOfPointsAlongAMeridian
29   real, dimension(:), allocatable     ::  values
30   integer                             ::  numberOfValues
31   real                                ::  average,min_val, max_val
32
33   call grib_open_file(ifile, &
34        '../../data/tigge_pf_ecmwf.grib2','r')
35
36   ! count the messages in the file
37   call grib_count_in_file(ifile,n)
38   allocate(igrib(n))
39   igrib=-1
40
41   ! Load the messages from the file.
42   DO i=1,n
43      call grib_new_from_file(ifile,igrib(i), iret)
44   END DO
45
46   ! we can close the file
47   call grib_close_file(ifile)
48
49   ! Loop on all the messages in memory
50   DO i=1,n
51      write(*,*) 'processing message number ',i
52      !     get as a integer
53      call grib_get(igrib(i),'numberOfPointsAlongAParallel', &
54           numberOfPointsAlongAParallel)
55      write(*,*) 'numberOfPointsAlongAParallel=', &
56           numberOfPointsAlongAParallel
57
58      !     get as a integer
59      call grib_get(igrib(i),'numberOfPointsAlongAMeridian', &
60           numberOfPointsAlongAMeridian)
61      write(*,*) 'numberOfPointsAlongAMeridian=', &
62           numberOfPointsAlongAMeridian
```

```
63
64      !     get as a real
65      call grib_get(igrib(i), 'latitudeOfFirstGridPointInDegrees', &
66           latitudeOfFirstPointInDegrees)
67      write(*,*) 'latitudeOfFirstGridPointInDegrees=', &
68           latitudeOfFirstPointInDegrees
69
70      !     get as a real
71      call grib_get(igrib(i), 'longitudeOfFirstGridPointInDegrees', &
72           longitudeOfFirstPointInDegrees)
73      write(*,*) 'longitudeOfFirstGridPointInDegrees=', &
74           longitudeOfFirstPointInDegrees
75
76      !     get as a real
77      call grib_get(igrib(i), 'latitudeOfLastGridPointInDegrees', &
78           latitudeOfLastPointInDegrees)
79      write(*,*) 'latitudeOfLastGridPointInDegrees=', &
80           latitudeOfLastPointInDegrees
81
82      !     get as a real
83      call grib_get(igrib(i), 'longitudeOfLastGridPointInDegrees', &
84           longitudeOfLastPointInDegrees)
85      write(*,*) 'longitudeOfLastGridPointInDegrees=', &
86           longitudeOfLastPointInDegrees
87
88
89      !     get the size of the values array
90      call grib_get_size(igrib(i),'values',numberOfValues)
91      write(*,*) 'numberOfValues=',numberOfValues
92
93      allocate(values(numberOfValues), stat=iret)
94      !     get data values
95      call grib_get(igrib(i),'values',values)
96      call grib_get(igrib(i),'min',min_val) ! can also be obtained through minval(values)
97      call grib_get(igrib(i),'max',max_val) ! can also be obtained through maxval(values)
98      call grib_get(igrib(i),'average',average) ! can also be obtained through maxval(values)
99
100      write(*,*)'There are ',numberOfValues, &
101           ' average is ',average, &
102           ' min is ',  min_val, &
103           ' max is ',  max_val
104      write(*,*) '----------------------'
105   END DO
106
107   DO i=1,n
108     call grib_release(igrib(i))
109   END DO
110
111   deallocate(values)
112   deallocate(igrib)
113
114 end program get
```

3.5 get.f90

How to get values through the key names.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to get values using keys.
10 !
11 !  Author: Enrico Fucile
12 !
13 !
14 program get
15   use grib_api
16   implicit none
17
18   integer                             ::  ifile
19   integer                             ::  iret
20   integer                             ::  igrib
21   real                                ::  latitudeOfFirstPointInDegrees
22   real                                ::  longitudeOfFirstPointInDegrees
23   real                                ::  latitudeOfLastPointInDegrees
24   real                                ::  longitudeOfLastPointInDegrees
25   integer                             ::  numberOfPointsAlongAParallel
26   integer                             ::  numberOfPointsAlongAMeridian
27   real, dimension(:), allocatable     ::  values
28   integer                             ::  numberOfValues
29   real                                ::  average,min_val, max_val
30   integer                             ::  is_missing
31
32   call grib_open_file(ifile, &
33        '../../data/reduced_latlon_surface.grib1','r')
34
35   ! Loop on all the messages in a file.
36
37   !     a new grib message is loaded from file
38   !     igrib is the grib id to be used in subsequent calls
39   call  grib_new_from_file(ifile,igrib, iret)
40
41   LOOP: DO WHILE (iret /= GRIB_END_OF_FILE)
42
43      !check if the value of the key is MISSING
44      is_missing=0;
45      call grib_is_missing(igrib,'numberOfPointsAlongAParallel', &
46           is_missing);
47      if ( is_missing /= 1 ) then
48         !     get as a integer
49         call grib_get(igrib,'numberOfPointsAlongAParallel', &
50              numberOfPointsAlongAParallel)
51         write(*,*) 'numberOfPointsAlongAParallel=', &
52              numberOfPointsAlongAParallel
53      else
54         write(*,*) 'numberOfPointsAlongAParallel is missing'
55      endif
56      !     get as a integer
57      call grib_get(igrib,'numberOfPointsAlongAMeridian', &
58           numberOfPointsAlongAMeridian)
59      write(*,*) 'numberOfPointsAlongAMeridian=', &
60           numberOfPointsAlongAMeridian
61
62      !     get as a real
```

```
63      call grib_get(igrib, 'latitudeOfFirstGridPointInDegrees', &
64           latitudeOfFirstPointInDegrees)
65      write(*,*) 'latitudeOfFirstGridPointInDegrees=', &
66           latitudeOfFirstPointInDegrees
67
68      !     get as a real
69      call grib_get(igrib, 'longitudeOfFirstGridPointInDegrees', &
70           longitudeOfFirstPointInDegrees)
71      write(*,*) 'longitudeOfFirstGridPointInDegrees=', &
72           longitudeOfFirstPointInDegrees
73
74      !     get as a real
75      call grib_get(igrib, 'latitudeOfLastGridPointInDegrees', &
76           latitudeOfLastPointInDegrees)
77      write(*,*) 'latitudeOfLastGridPointInDegrees=', &
78           latitudeOfLastPointInDegrees
79
80      !     get as a real
81      call grib_get(igrib, 'longitudeOfLastGridPointInDegrees', &
82           longitudeOfLastPointInDegrees)
83      write(*,*) 'longitudeOfLastGridPointInDegrees=', &
84           longitudeOfLastPointInDegrees
85
86
87      !     get the size of the values array
88      call grib_get_size(igrib,'values',numberOfValues)
89      write(*,*) 'numberOfValues=',numberOfValues
90
91      allocate(values(numberOfValues), stat=iret)
92      !     get data values
93      call grib_get(igrib,'values',values)
94      call grib_get(igrib,'min',min_val) ! can also be obtained through minval(values)
95      call grib_get(igrib,'max',max_val) ! can also be obtained through maxval(values)
96      call grib_get(igrib,'average',average) ! can also be obtained through maxval(values)
97
98      write(*,*)'There are ',numberOfValues, &
99           ' average is ',average, &
100           ' min is ',  min_val, &
101           ' max is ',  max_val
102
103      call grib_release(igrib)
104
105      call grib_new_from_file(ifile,igrib, iret)
106
107   end do LOOP
108
109   call grib_close_file(ifile)
110
111   deallocate(values)
112 end program get
```

3.6 get_data.f90

How to get latitude/longitude/values.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to get lat/lon/values.
10 !
11 !
12 !  Author: Enrico Fucile
13 !
14 !
15 program get_data
16 use grib_api
17 implicit none
18   integer            :: ifile
19   integer            :: iret,i
20   real(kind=8),dimension(:),allocatable     :: lats,lons,values
21   integer(4)        :: numberOfPoints
22   real(8)  :: missingValue=9999
23   integer           :: count=0
24   character(len=256) :: filename
25
26 !     Message identifier.
27   integer            :: igrib
28
29   ifile=5
30
31   call grib_open_file(ifile, &
32        '../../data/reduced_latlon_surface.grib1','R')
33
34 ! Loop on all the messages in a file.
35
36   call grib_new_from_file(ifile,igrib,iret)
37
38   do while (iret/=GRIB_END_OF_FILE)
39     count=count+1
40     print *, "===== Message #",count
41     call grib_get(igrib,'numberOfPoints',numberOfPoints)
42     call grib_set(igrib,'missingValue',missingValue)
43
44     allocate(lats(numberOfPoints))
45     allocate(lons(numberOfPoints))
46     allocate(values(numberOfPoints))
47
48     call grib_get_data(igrib,lats,lons,values)
49
50     do i=1,numberOfPoints
51       if (values(i) /= missingValue) then
52         print *, lats(i),lons(i),values(i)
53       end if
54     enddo
55
56     deallocate(lats)
57     deallocate(lons)
58     deallocate(values)
59
60     call grib_release(igrib)
61     call grib_new_from_file(ifile,igrib, iret)
62
```

```
63   end do
64
65
66   call grib_close_file(ifile)
67
68 end program
```

3.7 get_pl.f90

How to get the list of number of points for each parallel in reduced grids.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to get PL values.
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program get_pl
16   use grib_api
17   implicit none
18   integer                         :: infile
19   integer                         :: igrib
20   integer                         :: PLPresent, nb_pl
21   real, dimension(:), allocatable :: pl
22
23
24   call grib_open_file(infile, &
25        '../../data/reduced_gaussian_surface.grib1','r')
26
27   !     a new grib message is loaded from file
28   !     igrib is the grib id to be used in subsequent calls
29   call grib_new_from_file(infile,igrib)
30
31   !     set PVPresent as an integer
32   call grib_get(igrib,'PLPresent',PLPresent)
33   print*, "PLPresent= ", PLPresent
34   if (PLPresent == 1) then
35      call grib_get_size(igrib,'pl',nb_pl)
36      print*, "there are ", nb_pl, " PL values"
37      allocate(pl(nb_pl))
38      call grib_get(igrib,'pl',pl)
39      print*, "pl = ", pl
40      deallocate(pl)
41   else
42      print*, "There is no PL values in your GRIB message!"
43   end if
44   call grib_release(igrib)
45
46   call grib_close_file(infile)
47
48 end program get_pl
```

3.8 get_pv.f90

How to get the list of levels.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to get PV values.
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program get_pv
16   use grib_api
17   implicit none
18   integer                         :: infile
19   integer                         :: igrib
20   integer                         :: PVPresent, nb_pv
21   real, dimension(:), allocatable :: pv
22
23
24   call grib_open_file(infile, &
25        '../../data/reduced_gaussian_model_level.grib1','r')
26
27   !     a new grib message is loaded from file
28   !     igrib is the grib id to be used in subsequent calls
29   call grib_new_from_file(infile,igrib)
30
31   !     set PVPresent as an integer
32   call grib_get(igrib,'PVPresent',PVPresent)
33   print*, "PVPresent = ", PVPresent
34   if (PVPresent == 1) then
35      call grib_get_size(igrib,'pv',nb_pv)
36      print*, "There are ", nb_pv, " PV values"
37      allocate(pv(nb_pv))
38      call grib_get(igrib,'pv',pv)
39      print*, "pv = ", pv
40      deallocate(pv)
41   else
42      print*, "There is no PV values in your GRIB message!"
43   end if
44   call grib_release(igrib)
45
46   call grib_close_file(infile)
47
48 end program get_pv
```

3.9 index.f90

How access a grib file through and index.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: How to create and use and index to access messages from a file
10 !
11 !
12 !  Author: Enrico Fucile
13 !
14 !
15 program index
16   use grib_api
17   implicit none
18
19   integer              :: iret
20   character(len = 256) :: error
21   integer,dimension(:),allocatable :: paramId,step,level,number
22   integer                          :: oparamId,ostep,olevel,onumber
23   integer                          :: paramIdSize,numberSize,levelSize,stepSize
24   integer              :: i,j,k,l
25   integer              :: idx,igrib,count
26
27   ! create an index from a grib file using some keys
28   call grib_index_create(idx,'../../data/index.grib','paramId,number,level,step')
29
30   ! get the number of distinct values of paramId in the index
31   call grib_index_get_size(idx,'paramId',paramIdSize)
32   ! allocate the array to contain the list of distinct paramId
33   allocate(paramId(paramIdSize))
34   ! get the list of distinct paramId from the index
35   call grib_index_get(idx,'paramId',paramId)
36   write(*,'(a,i3)') 'paramIdSize=',paramIdSize
37
38   ! get the number of distinct values of number in the index
39   call grib_index_get_size(idx,'number',numberSize)
40   ! allocate the array to contain the list of distinct numbers
41   allocate(number(numberSize))
42   ! get the list of distinct numbers from the index
43   call grib_index_get(idx,'number',number)
44   write(*,'(a,i3)') 'numberSize=',numberSize
45
46   ! get the number of distinct values of level in the index
47   call grib_index_get_size(idx,'level',levelSize)
48   ! allocate the array to contain the list of distinct levels
49   allocate(level(levelSize))
50   ! get the list of distinct levels from the index
51   call grib_index_get(idx,'level',level)
52   write(*,'(a,i3)') 'levelSize=',levelSize
53
54   ! get the number of distinct values of step in the index
55   call grib_index_get_size(idx,'step',stepSize)
56   ! allocate the array to contain the list of distinct steps
57   allocate(step(stepSize))
58   ! get the list of distinct steps from the index
59   call grib_index_get(idx,'step',step)
60   write(*,'(a,i3)') 'stepSize=',stepSize
61
62   count=0
```

```
63   do i=1,paramIdSize ! loop on paramId
64     ! select paramId=paramId(i)
65     call grib_index_select(idx,'paramId',paramId(i))
66
67     do j=1,numberSize ! loop on number
68       ! select number=number(j)
69       call grib_index_select(idx,'number',number(j))
70
71       do k=1,levelSize ! loop on level
72         ! select level=level(k)
73         call grib_index_select(idx,'level',level(k))
74
75         do l=1,stepSize ! loop on step
76           ! select step=step(l)
77           call grib_index_select(idx,'step',step(l))
78
79           call grib_new_from_index(idx,igrib, iret)
80           do while (iret /= GRIB_END_OF_INDEX)
81              count=count+1
82              call grib_get(igrib,'paramId',oparamId)
83              call grib_get(igrib,'number',onumber)
84              call grib_get(igrib,'level',olevel)
85              call grib_get(igrib,'step',ostep)
86              write(*,'(A,i3,A,i2,A,i4,A,i3)') 'paramId=',oparamId,&
87                      '   number=',onumber,&
88                      '   level=' ,olevel, &
89                      '   step='  ,ostep
90
91              call grib_new_from_index(idx,igrib, iret)
92           end do
93
94         end do ! loop on step
95       end do ! loop on level
96     end do ! loop on number
97   end do ! loop on paramId
98   write(*,'(i4,a)') count,' messages selected'
99   call grib_index_release(idx)
100
101 end program index
102
```

3.10 keys_iterator.f90

How to get the names of all the keys defined in a message and how to iterate through them.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description:
10 !       How to use keys_iterator to get all the available
11 !       keys in a message.
12 !
13 !  Author: Enrico Fucile
14 !
15 !
16 program keys_iterator
17   use grib_api
18   implicit none
19   character(len=20)  :: name_space
20   integer            :: kiter,ifile,igrib,iret
21   character(len=256) :: key
22   character(len=256) :: value
23   character(len=512) :: all
24   integer            :: grib_count
25
26   call grib_open_file(ifile, &
27        '../../data/regular_latlon_surface.grib1','r')
28
29   ! Loop on all the messages in a file.
30
31   call grib_new_from_file(ifile,igrib, iret)
32
33   do while (iret /= GRIB_END_OF_FILE)
34
35      grib_count=grib_count+1
36      write(*,*) '-- GRIB N. ',grib_count,' --'
37
38      ! valid name_spaces are ls and mars
39      name_space='ls'
40
41      call grib_keys_iterator_new(igrib,kiter,name_space)
42
43      do
44         call grib_keys_iterator_next(kiter, iret)
45
46         if (iret .ne. 1) exit
47
48         call grib_keys_iterator_get_name(kiter,key)
49         call grib_get(igrib,trim(key),value)
50         all=trim(key)// ' = ' // trim(value)
51         write(*,*) trim(all)
52
53      end do
54
55      call grib_keys_iterator_delete(kiter)
56      call grib_release(igrib)
57      call grib_new_from_file(ifile,igrib, iret)
58   end do
59
60
61   call grib_close_file(ifile)
62
```

```
63 end program keys_iterator
64
```

3.11 multi_write.f90

How to encode a grib message containing many fields.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to create a multi field message in memory and write
10 !               it in a file. The multi field messages can be created
11 !               only in grib edition 2.
12 !
13 !
14 !
15 program multi_write
16   use grib_api
17   implicit none
18   integer                          :: infile,outfile
19   integer                          :: in_gribid,iret
20   integer                          :: multi_gribid
21   integer                          :: step,startsection
22
23   ! multi field messages can be created only in edition 2
24   call grib_open_file(infile,'../../data/sample.grib2','r')
25
26   call grib_open_file(outfile,'multi_created.grib2','w')
27
28   !     a grib message is loaded from file
29   !     in_gribid is the grib id to be used in subsequent calls
30   call grib_new_from_file(infile,in_gribid)
31
32   startsection=4
33   do step=0,240,12
34
35     call grib_set(in_gribid,"step",step)
36     call grib_multi_append(in_gribid,startsection,multi_gribid)
37
38   enddo
39
40 !  write messages to a file
41   call grib_multi_write(multi_gribid,outfile)
42
43   call grib_close_file(infile)
44   call grib_close_file(outfile)
45
46 end program multi_write
```

3.12 multi.f90

How to decode a grib message containing many fields.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: How to decode grib messages containing multiple
10 !               fields. Try to turn on and off multi support to
11 !               see the difference. Default is OFF.
12 !               For all the tools defalut is multi support ON.
13 !
14 !
15 !  Author: Enrico Fucile
16 !
17 !
18 program multi
19   use grib_api
20   implicit none
21
22   integer              :: iret
23   character(len = 256) :: error
24   integer(kind = 4)    :: step
25   integer              :: ifile,igrib
26
27   call grib_open_file(ifile, '../../data/multi_created.grib2','r')
28
29   !     turn on support for multi fields messages */
30   call grib_multi_support_on()
31
32   !     turn off support for multi fields messages */
33   !call grib_multi_support_off()
34
35   call grib_new_from_file(ifile,igrib, iret)
36   !     Loop on all the messages in a file.
37
38   write(*,*) 'step'
39   do while (iret /= GRIB_END_OF_FILE)
40
41      call grib_get(igrib,'step', step)
42      write(*,'(i3)') step
43
44      call grib_new_from_file(ifile,igrib, iret)
45
46   end do
47   call grib_close_file(ifile)
48
49 end program multi
50
```

3.13 nearest.f90

How to find the nearest grid points.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to use grib_find_nearest and grib_get_element
10 !
11 !
12 !  Author: Enrico Fucile
13 !
14 !
15 !
16 program find
17   use grib_api
18   implicit none
19   integer                                      :: npoints
20   integer                                      :: infile
21   integer                                      :: igrib, ios, i
22   real(8), dimension(:), allocatable  :: lats, lons
23   real(8), dimension(:), allocatable  :: nearest_lats, nearest_lons
24   real(8), dimension(:), allocatable  :: distances, values, lsm_values
25   integer(kind=kindOfInt), dimension(:), allocatable  :: indexes
26   real(kind=8)                         :: value
27
28 ! initialization
29   open( unit=1, file="../../data/list_points",form="formatted",action="read")
30   read(unit=1,fmt=*) npoints
31   allocate(lats(npoints))
32   allocate(lons(npoints))
33   allocate(nearest_lats(npoints))
34   allocate(nearest_lons(npoints))
35   allocate(distances(npoints))
36   allocate(lsm_values(npoints))
37   allocate(values(npoints))
38   allocate(indexes(npoints))
39   do i=1,npoints
40      read(unit=1,fmt=*, iostat=ios) lats(i), lons(i)
41      if (ios /= 0) then
42         npoints = i - 1
43         exit
44      end if
45   end do
46   close(unit=1)
47   call grib_open_file(infile, &
48        '../../data/reduced_gaussian_lsm.grib1','r')
49
50   !     a new grib message is loaded from file
51   !     igrib is the grib id to be used in subsequent calls
52   call grib_new_from_file(infile,igrib)
53
54
55   call grib_find_nearest(igrib, .true., lats, lons, nearest_lats, nearest_lons,lsm_values, distances, i
56   call grib_release(igrib)
57
58   call grib_close_file(infile)
59
60 ! will apply it to another GRIB
61   call grib_open_file(infile, &
62        '../../data/reduced_gaussian_pressure_level.grib1','r')
```

```
63   call grib_new_from_file(infile,igrib)
64
65   call grib_get_element(igrib,"values", indexes, values)
66   call grib_release(igrib)
67   call grib_close_file(infile)
68
69   do i=1, npoints
70      print*,lats(i), lons(i), nearest_lats(i), nearest_lons(i), distances(i), lsm_values(i), values(i)
71   end do
72
73   deallocate(lats)
74   deallocate(lons)
75   deallocate(nearest_lats)
76   deallocate(nearest_lons)
77   deallocate(distances)
78   deallocate(lsm_values)
79   deallocate(values)
80   deallocate(indexes)
81
82 end program find
```

3.14 precision.f90

How to control precision when coding a grib field.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !
10 !  Description: how to control decimal precision when packing fields.
11 !
12 !
13 !  Author: Enrico Fucile
14 !
15 !
16 !
17 program precision
18   use grib_api
19   implicit none
20   integer(kind = 4)                              :: size
21   integer                                        :: infile,outfile
22   integer                                        :: igrib
23   real(kind = 8), dimension(:), allocatable      :: values1
24   real(kind = 8), dimension(:), allocatable      :: values2
25   real(kind = 8)                                 ::  maxa,a,maxv,minv,maxr,r
26   integer( kind = 4)                             :: decimalPrecision,bitsPerValue1,bitsPerValue2
27   integer                                        :: i, iret
28
29   call grib_open_file(infile, &
30        '../../data/regular_latlon_surface_constant.grib1','r')
31
32   call grib_open_file(outfile, &
33        '../../data/regular_latlon_surface_prec.grib1','w')
34
35   !     a new grib message is loaded from file
36   !     igrib is the grib id to be used in subsequent calls
37   call grib_new_from_file(infile,igrib)
38
39   !     bitsPerValue before changing the packing parameters
40   call grib_get(igrib,'bitsPerValue',bitsPerValue1)
41
42   !     get the size of the values array
43   call grib_get_size(igrib,"values",size)
44
45   allocate(values1(size), stat=iret)
46   allocate(values2(size), stat=iret)
47   !     get data values before changing the packing parameters*/
48   call grib_get(igrib,"values",values1)
49
50   !     setting decimal precision=2 means that 2 decimal digits
51   !     are preserved when packing.
52   decimalPrecision=2
53   call grib_set(igrib,"changeDecimalPrecision", &
54        decimalPrecision)
55
56   !     bitsPerValue after changing the packing parameters
57   call grib_get(igrib,"bitsPerValue",bitsPerValue2)
58
59   !     get data values after changing the packing parameters
60   call grib_get(igrib,"values",values2)
61
62   !     computing error
```

```
63   maxa=0
64   maxr=0
65   maxv=values2(1)
66   minv=maxv
67   do i=1,size
68      a=abs(values2(i)-values1(i))
69      if ( values2(i) .gt. maxv ) maxv=values2(i)
70      if ( values2(i) .lt. maxv ) minv=values2(i)
71      if ( values2(i) .ne. 0 ) then
72         r=abs((values2(i)-values1(i))/values2(i))
73      endif
74      if ( a .gt. maxa ) maxa=a
75      if ( r .gt. maxr ) maxr=r
76   enddo
77   write(*,*) "max absolute error = ",maxa
78   write(*,*) "max relative error = ",maxr
79   write(*,*) "min value = ",minv
80   write(*,*) "max value = ",maxv
81
82   write(*,*) "old number of bits per value=",bitsPerValue1
83   write(*,*) "new number of bits per value=",bitsPerValue2
84
85   !     write modified message to a file
86   call grib_write(igrib,outfile)
87
88   call grib_release(igrib)
89
90   call grib_close_file(infile)
91
92   call grib_close_file(outfile)
93
94   deallocate(values1)
95   deallocate(values2)
96 end program precision
97
```

3.15 print_data.f90

How to print all the data contained in a grib file.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: prints all the data contained in a grib file
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program print_data
16 use grib_api
17 implicit none
18 integer            :: ifile
19 integer            :: iret
20 integer            :: igrib
21 integer            :: i
22 real(kind=8), dimension(:), allocatable       :: values
23 integer(kind=4)    :: numberOfValues
24 real(kind=8)       :: average
25 real(kind=8)       :: max
26 real(kind=8)       :: min
27 character(len=256) :: error
28
29 call grib_open_file(ifile, &
30            '../../data/constant_field.grib1','r')
31
32 !     a new grib message is loaded from file
33 !     igrib is the grib id to be used in subsequent calls
34       call grib_new_from_file(ifile,igrib)
35
36
37 !     get the size of the values array
38       call grib_get_size(igrib,'values',numberOfValues)
39
40 !     get data values
41   print*, 'number of values ', numberOfValues
42   allocate(values(numberOfValues), stat=iret)
43
44   call grib_get(igrib,'values',values)
45
46   do i=1,numberOfValues
47     write(*,*)'  ',i,values(i)
48   enddo
49
50
51   write(*,*)numberOfValues,' values found '
52
53   call grib_get(igrib,'max',max)
54   write(*,*) 'max=',max
55   call grib_get(igrib,'min',min)
56   write(*,*) 'min=',min
57   call grib_get(igrib,'average',average)
58   write(*,*) 'average=',average
59
60   call grib_release(igrib)
61
62   call grib_close_file(ifile)
```

```
63
64   deallocate(values)
65
66 end program print_data
```

3.16 samples.f90

How to create a new message from a samples.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to create a new GRIB message from a sample.
10 !
11 !
12 program sample
13   use grib_api
14   implicit none
15   integer  :: err
16   integer  :: outfile, infile, datafile
17   integer  :: igribsample,igribclone,igribdata, size
18   integer  :: date, startStep, endStep, table2Version, indicatorOfParameter
19   integer  :: decimalPrecision
20   real     :: missingValue
21   character(len=10) stepType
22   double precision, dimension(:), allocatable   :: v1,v2,v
23
24   date = 20080104
25   startStep = 0
26   endStep = 12
27   stepType = 'accum'
28   table2Version = 2
29   indicatorOfParameter = 61
30   decimalPrecision = 2
31
32   !     a new grib message is loaded from an existing sample
33   !     samples are searched in a default sample path (use grib_info
34   !     to see where it is. The default sample path can be changed by
35   !     setting the environment variable GRIB_SAMPLES_PATH
36   call grib_new_from_samples(igribsample, "regular_latlon_surface.grib1")
37
38   call grib_open_file(outfile, 'out.grib1','w')
39   call grib_open_file(datafile,'../../data/tp_ecmwf.grib','r')
40
41   call grib_new_from_file(datafile,igribdata,err)
42
43   call grib_get_size(igribdata,'values',size)
44   allocate(v(size))
45   allocate(v1(size))
46   allocate(v2(size))
47
48   call grib_get(igribdata,'values',v)
49
50   v=v*1000.0 ! different units for the output grib
51   v1=v
52
53   do while (err/=GRIB_END_OF_FILE)
54
55     call grib_clone(igribsample,igribclone) ! clone sample before modifying it
56
57     call grib_set(igribclone,'date',date)
58     call grib_set(igribclone,'table2Version',table2Version)
59     call grib_set(igribclone,'indicatorOfParameter',indicatorOfParameter)
60
61     call grib_set(igribclone,'stepType',stepType)
62     call grib_set(igribclone,'startStep',startStep)
```

```
63     call grib_set(igribclone,'endStep',endStep)
64
65     call grib_set(igribclone,'decimalPrecision',decimalPrecision)
66
67     call grib_set(igribclone,'values',v)
68
69     call grib_write(igribclone,outfile)
70
71     call grib_new_from_file(datafile,igribdata,err)
72
73     if (err==0) then
74       call grib_get(igribdata,'values',v2)
75
76       v2=v2*1000.0 ! different units for the output grib
77
78       v=v2-v1 ! accumulation from startStep to endStep
79
80       v1=v2 ! save previous step field
81
82       startStep=startStep+12
83       endStep=endStep+12
84
85     endif
86
87   enddo
88
89   deallocate(v)
90   deallocate(v1)
91   deallocate(v2)
92
93   call grib_close_file(outfile)
94
95 end program sample
```

3.17 set.f90

How to set values through the key names.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to set key values.
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program set
16   use grib_api
17   implicit none
18   integer(kind = 4)    :: centre, date
19   integer              :: infile,outfile
20   integer              :: igrib
21
22
23   centre = 80
24   call current_date(date)
25   call grib_open_file(infile, &
26        '../../data/regular_latlon_surface_constant.grib1','r')
27
28   call grib_open_file(outfile, &
29        'out.grib1','w')
30
31   !     a new grib message is loaded from file
32   !     igrib is the grib id to be used in subsequent calls
33   call grib_new_from_file(infile,igrib)
34
35   call grib_set(igrib,'date',date)
36   !     set centre as a integer */
37   call grib_set(igrib,'centre',centre)
38
39 ! check if it is correct in the actual GRIB message
40
41   call check_settings(igrib)
42
43   !     write modified message to a file
44   call grib_write(igrib,outfile)
45
46   call grib_release(igrib)
47
48   call grib_close_file(infile)
49
50   call grib_close_file(outfile)
51
52 contains
53
54 !====================================
55 subroutine current_date(date)
56 integer, intent(out) :: date
57
58 integer              :: val_date(8)
59 call date_and_time ( values = val_date)
60
61 date = val_date(1)* 10000 + val_date(2)*100 + val_date(3)
62 end subroutine current_date
```

```
63 !====================================
64 subroutine check_settings(gribid)
65   use grib_api
66   implicit none
67   integer, intent(in) :: gribid
68
69   integer(kind = 4)    :: int_value
70   character(len = 10)  :: string_value
71
72   !     get centre as a integer
73   call grib_get(gribid,'centre',int_value)
74   write(*,*) "get centre as a integer - centre = ",int_value
75
76   !     get centre as a string
77   call grib_get(gribid,'centre',string_value)
78   write(*,*) "get centre as a string  - centre = ",string_value
79
80   !     get date as a string
81   call grib_get(gribid,'dataDate',string_value)
82   write(*,*) "get date as a string    - date = ",string_value
83
84 end subroutine check_settings
85 end program set
```

3.18 set_bitmap.f90

How to set and use a bitmap.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to set a bitmap in a grib message
10 !
11 !
12 !  Author: Enrico Fucile
13 !
14 !
15 program set_bitmap
16   use grib_api
17   implicit none
18   integer                         :: infile,outfile
19   integer                         :: igrib, iret
20   integer                         :: numberOfValues
21   real, dimension(:), allocatable :: values
22   real                            :: missingValue
23   logical                         :: grib1Example
24
25   grib1Example=.true.
26
27   if (grib1Example) then
28     ! GRIB 1 example
29     call grib_open_file(infile,'../../data/regular_latlon_surface.grib1','r')
30   else
31     ! GRIB 2 example
32     call grib_open_file(infile,'../../data/regular_latlon_surface.grib2','r')
33   end if
34
35   call grib_open_file(outfile,'out.grib','w')
36
37   !     a new grib message is loaded from file
38   !     igrib is the grib id to be used in subsequent calls
39   call grib_new_from_file(infile,igrib)
40
41   ! The missingValue is not coded in the message.
42   ! It is a value we define as a placeholder for a missing value
43   ! in a point of the grid.
44   ! It should be choosen in a way that it cannot be confused
45   ! with a valid field value
46   missingValue=9999
47   call grib_set(igrib, 'missingValue',missingValue)
48   write(*,*) 'missingValue=',missingValue
49
50   ! get the size of the values array
51   call grib_get_size(igrib,'values',numberOfValues)
52   write(*,*) 'numberOfValues=',numberOfValues
53
54   allocate(values(numberOfValues), stat=iret)
55
56   ! get data values
57   call grib_get(igrib,'values',values)
58
59   ! enable bitmap
60   call grib_set(igrib,"bitmapPresent",1)
61
62   ! some values are missing
```

```
63   values(1:10) = missingValue
64
65   ! set the values (the bitmap will be automatically built)
66   call grib_set(igrib,'values', values)
67
68   !  write modified message to a file
69   call grib_write(igrib,outfile)
70
71   ! FREE MEMORY
72   call grib_release(igrib)
73
74   call grib_close_file(infile)
75
76   call grib_close_file(outfile)
77
78   deallocate(values)
79
80 end program set_bitmap
```

3.19 set_missing.f90

How to set a missing value in the header.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to set missing a key value.
10 !
11 !
12 !  Author: Enrico Fucile
13 !
14 !
15 !
16 program set
17   use grib_api
18   implicit none
19   integer              :: infile,outfile
20   integer              :: igrib
21
22   infile=5
23   outfile=6
24
25   call grib_open_file(infile, &
26        '../../data/reduced_gaussian_pressure_level.grib2','r')
27
28   call grib_open_file(outfile, &
29        'out_surface_level.grib2','w')
30
31   !     a new grib message is loaded from file
32   !     igrib is the grib id to be used in subsequent calls
33   call grib_new_from_file(infile,igrib)
34
35   call grib_set(igrib,'typeOfFirstFixedSurface','sfc')
36   call grib_set_missing(igrib,'scaleFactorOfFirstFixedSurface')
37   call grib_set_missing(igrib,'scaledValueOfFirstFixedSurface')
38
39   call grib_write(igrib,outfile)
40
41   call grib_release(igrib)
42
43   call grib_close_file(infile)
44
45   call grib_close_file(outfile)
46
47 end program set
```

3.20 set_pv.f90

How to set the list of levels.

```
1 ! Copyright 2005-2007 ECMWF
2 !
3 ! Licensed under the GNU Lesser General Public License which
4 ! incorporates the terms and conditions of version 3 of the GNU
5 ! General Public License.
6 ! See LICENSE and gpl-3.0.txt for details.
7 !
8 !
9 !  Description: how to set pv values.
10 !
11 !
12 !  Author: Anne Fouilloux
13 !
14 !
15 program set_pv
16   use grib_api
17   implicit none
18   integer                         :: numberOfLevels
19   integer                         :: numberOfCoefficients
20   integer                         :: outfile, igrib
21   integer                         :: i, ios
22   real, dimension(:),allocatable  :: pv
23
24   numberOfLevels=60
25   numberOfCoefficients=2*(numberOfLevels+1)
26
27   allocate(pv(numberOfCoefficients))
28
29   ! read the model level coefficients from file
30   open( unit=1, file="../../data/60_model_levels", &
31                 form="formatted",action="read")
32
33   do i=1,numberOfCoefficients,2
34      read(unit=1,fmt=*, iostat=ios) pv(i), pv(i+1)
35      if (ios /= 0) then
36         print *, "I/O error: ",ios
37         exit
38      end if
39   end do
40
41   ! print coefficients
42   !do i=1,numberOfCoefficients,2
43   !  print *,"  a=",pv(i)," b=",pv(i+1)
44   !end do
45
46   close(unit=1)
47
48   call grib_open_file(outfile, 'out.grib1','w')
49
50   !     a new grib message is loaded from file
51   !     igrib is the grib id to be used in subsequent calls
52   call grib_new_from_samples(igrib, "reduced_gg_sfc_grib1")
53
54   !     set levtype to ml (model level)
55   call grib_set(igrib,'levtype','ml')
56
57   !     set level
58   call grib_set(igrib,'level',2)
59
60   !     set PVPresent as an integer
61   call grib_set(igrib,'PVPresent',1)
62
```

```
63   call grib_set(igrib,'pv',pv)
64
65   !     write modified message to a file
66   call grib_write(igrib,outfile)
67
68   !  FREE MEMORY
69   call grib_release(igrib)
70   deallocate(pv)
71
72   call grib_close_file(outfile)
73
74 end program set_pv
```

3.21 get.c

get.c How to get values through the key names.

```
1
10 /*
11  * C Implementation: get
12  *
13  * Description: how to get values using keys.
14  *
15  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
16  *
17  *
18  */
19 #include <stdio.h>
20 #include <stdlib.h>
21
22 #include "grib_api.h"
23
24 int main(int argc, char** argv) {
25   int err = 0;
26   double *values = NULL;
27   size_t values_len= 0;
28
29   size_t i = 0;
30
31   double latitudeOfFirstGridPointInDegrees;
32   double longitudeOfFirstGridPointInDegrees;
33   double latitudeOfLastGridPointInDegrees;
34   double longitudeOfLastGridPointInDegrees;
35
36   double jDirectionIncrementInDegrees;
37   double iDirectionIncrementInDegrees;
38
39   long numberOfPointsAlongAParallel;
40   long numberOfPointsAlongAMeridian;
41
42   double average = 0;
43
44   FILE* in = NULL;
45   char* filename = "../../data/regular_latlon_surface.grib1";
46   grib_handle *h = NULL;
47
48   in = fopen(filename,"r");
49   if(!in) {
50     printf("ERROR: unable to open file %s\n",filename);
51     return 1;
52   }
53
54   /* create new handle from a message in a file*/
55   h = grib_handle_new_from_file(0,in,&err);
56   if (h == NULL) {
57     printf("Error: unable to create handle from file %s\n",filename);
58   }
59
60   /* get as a long*/
61   GRIB_CHECK(grib_get_long(h,"numberOfPointsAlongAParallel",&numberOfPointsAlongAParallel),0);
62   printf("numberOfPointsAlongAParallel=%ld\n",numberOfPointsAlongAParallel);
63
64   /* get as a long*/
65   GRIB_CHECK(grib_get_long(h,"numberOfPointsAlongAMeridian",&numberOfPointsAlongAMeridian),0);
66   printf("numberOfPointsAlongAMeridian=%ld\n",numberOfPointsAlongAMeridian);
67
68   /* get as a double*/
69   GRIB_CHECK(grib_get_double(h,"latitudeOfFirstGridPointInDegrees",&latitudeOfFirstGridPointInDegrees),
70   printf("latitudeOfFirstGridPointInDegrees=%g\n",latitudeOfFirstGridPointInDegrees);
```

```
71
72   /* get as a double*/
73   GRIB_CHECK(grib_get_double(h,"longitudeOfFirstGridPointInDegrees",&longitudeOfFirstGridPointInDegrees
74   printf("longitudeOfFirstGridPointInDegrees=%g\n",longitudeOfFirstGridPointInDegrees);
75
76   /* get as a double*/
77   GRIB_CHECK(grib_get_double(h,"latitudeOfLastGridPointInDegrees",&latitudeOfLastGridPointInDegrees),0)
78   printf("latitudeOfLastGridPointInDegrees=%g\n",latitudeOfLastGridPointInDegrees);
79
80   /* get as a double*/
81   GRIB_CHECK(grib_get_double(h,"longitudeOfLastGridPointInDegrees",&longitudeOfLastGridPointInDegrees),
82   printf("longitudeOfLastGridPointInDegrees=%g\n",longitudeOfLastGridPointInDegrees);
83
84   /* get as a double*/
85   GRIB_CHECK(grib_get_double(h,"jDirectionIncrementInDegrees",&jDirectionIncrementInDegrees),0);
86   printf("jDirectionIncrementInDegrees=%g\n",jDirectionIncrementInDegrees);
87
88   /* get as a double*/
89   GRIB_CHECK(grib_get_double(h,"iDirectionIncrementInDegrees",&iDirectionIncrementInDegrees),0);
90   printf("iDirectionIncrementInDegrees=%g\n",iDirectionIncrementInDegrees);
91
92   /* get the size of the values array*/
93   GRIB_CHECK(grib_get_size(h,"values",&values_len),0);
94
95   values = malloc(values_len*sizeof(double));
96
97   /* get data values*/
98   GRIB_CHECK(grib_get_double_array(h,"values",values,&values_len),0);
99
100   average = 0;
101   for(i = 0; i < values_len; i++)
102     average += values[i];
103
104   average /=(double)values_len;
105
106   free(values);
107
108   printf("There are %d values, average is %g\n",(int)values_len,average);
109
110   grib_handle_delete(h);
111
112   fclose(in);
113   return 0;
114 }
```

3.22 iterator.c

iterator.c How to use an iterator on latitude, longitude, values.

```
1
10 /*
11  * C Implementation: iterator
12  *
13  * Description: how to use an iterator on lat/lon/values.
14  *
15  *
16  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
17  *
18  *
19  */
20
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24
25 #include "grib_api.h"
26
27 void usage(char* prog) {
28   printf("Usage: %s grib_file\n",prog);
29   exit(1);
30 }
31
32 int main(int argc, char** argv) {
33   FILE* in = NULL;
34   int err = 0;
35   double lat,lon,value,missingValue=0;
36   int n=0;
37   char* filename = NULL;
38
39   /* Message handle. Required in all the grib_api calls acting on a message.*/
40   grib_handle *h = NULL;
41   /* Iterator on lat/lon/values.*/
42   grib_iterator* iter=NULL;
43
44   if (argc != 2) usage(argv[0]);
45
46   filename=strdup(argv[1]);
47
48   in = fopen(filename,"r");
49   if(!in) {
50     printf("ERROR: unable to open file %s\n",filename);
51     return 1;
52   }
53
54   /* Loop on all the messages in a file.*/
55   while ((h = grib_handle_new_from_file(0,in,&err)) != NULL ) {
56        /* Check of errors after reading a message. */
57     if (err != GRIB_SUCCESS) GRIB_CHECK(err,0);
58
59        /* Get the double representing the missing value in the field. */
60        GRIB_CHECK(grib_get_double(h,"missingValue",&missingValue),0);
61
62        /* A new iterator on lat/lon/values is created from the message handle h. */
63        iter=grib_iterator_new(h,0,&err);
64     if (err != GRIB_SUCCESS) GRIB_CHECK(err,0);
65
66     n = 0;
67        /* Loop on all the lat/lon/values. */
68     while(grib_iterator_next(iter,&lat,&lon,&value)) {
69          /* You can now print lat and lon,  */
70       printf("- %d - lat=%f lon=%f value=",n,lat,lon);
```

```
71           /* decide what to print if a missing value is found. */
72       if (value == missingValue ) printf("missing\n");
73           /* and print the value if is not missing. */
74           else printf("%f\n",value);
75       n++;
76     }
77
78         /* At the end the iterator is deleted to free memory. */
79     grib_iterator_delete(iter);
80
81     /* At the end the grib_handle is deleted to free memory. */
82     grib_handle_delete(h);
83   }
84
85
86   fclose(in);
87
88   return 0;
89 }
```

3.23 keys_iterator.c

keys_iterator.c How to get the names of all the keys defined in a message and how to iterate through them.

```
1
10 /*
11  * C Implementation: keys_iterator
12  *
13  * Description:
14  * Example on how to use keys_iterator functions and the
15  * grib_keys_iterator structure to get all the available
16  * keys in a message.
17  *
18  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
19  *
20  *
21  */
22
23 #include <assert.h>
24 #include <stdlib.h>
25 #include <stdio.h>
26 #include <unistd.h>
27
28 #include "grib_api.h"
29
30 #define MAX_KEY_LEN  255
31 #define MAX_VAL_LEN  1024
32
33 static void usage(char* progname);
34
35 int main(int argc, char *argv[])
36 {
37   /* To skip read only and not coded keys
38      unsigned long key_iterator_filter_flags=GRIB_KEYS_ITERATOR_SKIP_READ_ONLY ||
39      GRIB_KEYS_ITERATOR_SKIP_COMPUTED;
40   */
41   unsigned long key_iterator_filter_flags=GRIB_KEYS_ITERATOR_ALL_KEYS;
42
43   /* valid name_spaces are ls and mars */
44   char* name_space="ls";
45
46   /* name_space=NULL to get all the keys */
47   /* char* name_space=0; */
48
49   FILE* f;
50   grib_handle* h=NULL;
51   grib_keys_iterator* kiter=NULL;
52   int err=0;
53   int grib_count=0;
54
55   char value[MAX_VAL_LEN];
56   size_t vlen=MAX_VAL_LEN;
57
58   if (argc != 2) usage(argv[0]);
59
60   f = fopen(argv[1],"r");
61   if(!f) {
62     perror(argv[1]);
63     exit(1);
64   }
65
66   while((h = grib_handle_new_from_file(0,f,&err)) != NULL) {
67
68     grib_count++;
69     printf("-- GRIB N. %d --\n",grib_count);
70     if(!h) {
```

```
71       printf("ERROR: Unable to create grib handle\n");
72       exit(1);
73     }
74
75     kiter=grib_keys_iterator_new(h,key_iterator_filter_flags,name_space);
76     if (!kiter) {
77       printf("ERROR: Unable to create keys iterator\n");
78       exit(1);
79     }
80
81     while(grib_keys_iterator_next(kiter))
82     {
83       const char* name = grib_keys_iterator_get_name(kiter);
84       vlen=MAX_VAL_LEN;
85       GRIB_CHECK(grib_get_string(h,name,value,&vlen),name);
86       printf("%s = %s\n",name,value);
87     }
88
89     grib_keys_iterator_delete(kiter);
90
91   }
92
93   return 0;
94
95 }
96
97 static void usage(char* progname) {
98   printf("\nUsage: %s grib_file\n",progname);
99   exit(1);
100 }
101
```

3.24 multi.c

multi.c How to decode a grib message containing many fields.

```
1
10 /*
11  * C Implementation: multi
12  *
13  * Description: How to decode grib messages containing multiple
14  *              fields. Try to turn on and off multi support to
15  *              see the difference. Default is OFF.
16  *                      For all the tools defalut is multi support ON.
17  *
18  *
19  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
20  *
21  *
22  */
23 #include <stdio.h>
24 #include <stdlib.h>
25
26 #include "grib_api.h"
27
28 int main(int argc, char** argv) {
29   int err = 0;
30   long parameterCategory=0,parameterNumber=0,discipline=0;
31   FILE* in = NULL;
32   char* filename = "../../data/multi.grib2";
33   grib_handle *h = NULL;
34
35   /* turn on support for multi fields messages */
36   grib_multi_support_on(0);
37
38   /* turn off support for multi fields messages */
39   /* grib_multi_support_off(0); */
40
41   in = fopen(filename,"r");
42   if(!in) {
43     printf("ERROR: unable to open file %s\n",filename);
44     return 1;
45   }
46
47
48   while ((h = grib_handle_new_from_file(0,in,&err)) != NULL ) {
49
50     GRIB_CHECK(err,0);
51
52     GRIB_CHECK(grib_get_long(h,"discipline",&discipline),0);
53     printf("discipline=%ld\n",discipline);
54
55     GRIB_CHECK(grib_get_long(h,"parameterCategory",&parameterCategory),0);
56     printf("parameterCategory=%ld\n",parameterCategory);
57
58     GRIB_CHECK(grib_get_long(h,"parameterNumber",&parameterNumber),0);
59     printf("parameterNumber=%ld\n",parameterNumber);
60
61     if ( discipline == 0 && parameterCategory==2) {
62     if (parameterNumber == 2) printf("-------- u -------\n");
63     if (parameterNumber == 3) printf("-------- v -------\n");
64     }
65   }
66
67   grib_handle_delete(h);
68
69   fclose(in);
70   return 0;
```

```
71 }
```

3.25 multi_write.c

multi_write.c How to encode a grib message containing many fields.

```
1
10 /*
11  * C Implementation: multi
12  *
13  * Description: How to encode grib messages containing multiple
14  *              fields.
15  *
16  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
17  *
18  *
19  */
20 #include <stdio.h>
21 #include <stdlib.h>
22
23 #include "grib_api.h"
24
25 void usage(char* prog) {
26   printf("usage: %s in.grib out.grib\n",prog);
27   exit(1);
28 }
29
30 int main(int argc, char** argv) {
31   int err = 0;
32   FILE* in = NULL;
33   FILE* of = NULL;
34   long step;
35   char* filename=NULL;
36   char* ofilename=NULL;
37   grib_handle *h = NULL;
38   grib_multi_handle *mh=NULL;
39
40   if (argc < 3) usage(argv[0]);
41   filename=argv[1];
42   ofilename=argv[2];
43
44   /* open input file */
45   in = fopen(filename,"r");
46   if(!in) {
47     printf("ERROR: unable to open file %s\n",filename);
48     return 1;
49   }
50
51   /* new grib handle from input file */
52   h = grib_handle_new_from_file(0,in,&err);
53   GRIB_CHECK(err,0);
54   /* create a new empty multi field handle */
55   mh=grib_multi_handle_new(0);
56   if (!mh) {
57     printf("unable to create multi field handle\n");
58         exit(1);
59   }
60
61   for (step=12;step<=120;step+=12) {
62     /* set step */
63     grib_set_long(h,"step",step);
64         /* append h to mh repeating the sections preceding section 4 */
65         grib_multi_handle_append(h,4,mh);
66   }
67
68   /* open output file */
69   of=fopen(ofilename,"w");
70   if(!of) {
```

```
71         printf("ERROR: unable to open file %s\n",ofilename);
72     exit(1);
73   }
74
75   /* write multi fields handle to output file */
76   grib_multi_handle_write(mh,of);
77   fclose(of);
78
79   /* clean memory */
80   grib_handle_delete(h);
81   grib_multi_handle_delete(mh);
82
83   fclose(in);
84   return 0;
85 }
```

3.26 nearest.c

nearest.c How to find the nearest grid points.

```
1
10 /*
11  * C Implementation: fieldset
12  *
13  * Description: how to use a fieldset.
14  *
15  *
16  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
17  *
18  *
19  */
20
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24
25 #include "grib_api.h"
26
27 void usage(char* prog) {
28   printf("Usage: %s grib_file grib_file ...\n",prog);
29   exit(1);
30 }
31
32 int main(int argc, char** argv) {
33   int err = 0;
34   long step=0;
35   size_t nfiles;
36   int i=0;
37   grib_fieldset* set=NULL;
38   grib_handle* h=NULL;
39   char param[20]={0,};
40   size_t len=20;
41   double lats[4]={0,};
42   double lons[4]={0,};
43   double values[4]={0,};
44   double distances[4]={0,};
45   int indexes[4]={0,};
46   char* order_by="param,step";
47
48   size_t size=4;
49   double lat=-40,lon=15;
50   int mode=0;
51   int count;
52   char** filenames;
53   grib_nearest* nearest=NULL;
54
55   if (argc < 2) usage(argv[0]);
56
57   nfiles=argc-1;
58   filenames=(char**)malloc(sizeof(char*)*nfiles);
59   for (i=0;i<nfiles;i++)
60     filenames[i]=(char*)strdup(argv[i+1]);
61
62   set=grib_fieldset_new_from_files(0,filenames,nfiles,0,0,0,order_by,&err);
63   GRIB_CHECK(err,0);
64
65   printf("\nordering by %s\n",order_by);
66   printf("\n%d fields in the fieldset\n",grib_fieldset_count(set));
67   printf("n,step,param\n");
68
69   mode=GRIB_NEAREST_SAME_GRID |  GRIB_NEAREST_SAME_POINT;
70   count=1;
```

```
71   while ((h=grib_fieldset_next_handle(set,&err))!=NULL) {
72     GRIB_CHECK(grib_get_long(h,"step",&step),0);
73         len=20;
74     GRIB_CHECK(grib_get_string(h,"param",param,&len),0);
75
76     printf("%d %ld %s  ",count,step,param);
77     if (!nearest) nearest=grib_nearest_new(h,&err);
78     GRIB_CHECK(err,0);
79     GRIB_CHECK(grib_nearest_find(nearest,h,lat,lon,mode,lats,lons,values,distances,indexes,&size),0);
80     for (i=0;i<4;i++) printf("%d %.2f %.2f %g %g - ",
81          (int)indexes[i],lats[i],lons[i],distances[i],values[i]);
82     printf("\n");
83
84     grib_handle_delete(h);
85     count++;
86   }
87
88   if (nearest) grib_nearest_delete(nearest);
89
90   if (set) grib_fieldset_delete(set);
91
92   return 0;
93 }
```

3.27 precision.c

precision.c How to control precision when coding a grib field.

```
1
10 /*
11  * C Implementation: precision
12  *
13  * Description: how to control decimal precision when packing fields.
14  *
15  *
16  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
17  *
18  *
19  */
20 #include <stdio.h>
21 #include <stdlib.h>
22 #include <math.h>
23
24 #include "grib_api.h"
25
26 int main(int argc, char** argv) {
27   int err = 0;
28   size_t size=0;
29
30   FILE* in = NULL;
31   char* infile = "../../data/regular_latlon_surface.grib1";
32   FILE* out = NULL;
33   char* outfile = "out.grib1";
34   grib_handle *h = NULL;
35   const void* buffer = NULL;
36   double* values1=NULL;
37   double* values2=NULL;
38   double maxa=0,a=0;
39   double maxv=0,minv=0;
40   double maxr=0,r=0;
41   long decimalPrecision;
42   long bitsPerValue1=0, bitsPerValue2=0;
43   int i=0;
44
45   in = fopen(infile,"r");
46   if(!in) {
47     printf("ERROR: unable to open file %s\n",infile);
48     return 1;
49   }
50
51   out = fopen(outfile,"w");
52   if(!in) {
53     printf("ERROR: unable to open file %s\n",outfile);
54     return 1;
55   }
56
57   /* create a new handle from a message in a file */
58   h = grib_handle_new_from_file(0,in,&err);
59   if (h == NULL) {
60     printf("Error: unable to create handle from file %s\n",infile);
61   }
62
63   /* bitsPerValue before changing the packing parameters */
64   GRIB_CHECK(grib_get_long(h,"bitsPerValue",&bitsPerValue1),0);
65
66   /* get the size of the values array*/
67   GRIB_CHECK(grib_get_size(h,"values",&size),0);
68
69   values1 = malloc(size*sizeof(double));
70   /* get data values before changing the packing parameters*/
```

```
71   GRIB_CHECK(grib_get_double_array(h,"values",values1,&size),0);
72
73   /* changing decimal precition to 2 means that 2 decimal digits
74      are preserved when packing.  */
75   decimalPrecision=2;
76   GRIB_CHECK(grib_set_long(h,"changeDecimalPrecision",decimalPrecision),0);
77
78   /* bitsPerValue after changing the packing parameters */
79   GRIB_CHECK(grib_get_long(h,"bitsPerValue",&bitsPerValue2),0);
80
81   values2 = malloc(size*sizeof(double));
82   /* get data values after changing the packing parameters*/
83   GRIB_CHECK(grib_get_double_array(h,"values",values2,&size),0);
84
85   /* computing error */
86   maxa=0;
87   maxr=0;
88   maxv=values2[0];
89   minv=maxv;
90   for (i=0;i<size;i++) {
91      a=fabs(values2[i]-values1[i]);
92      if ( values2[i] > maxv ) maxv=values2[i];
93      if ( values2[i] < maxv ) minv=values2[i];
94      if ( values2[i] !=0 ) r=fabs((values2[i]-values1[i])/values2[i]);
95      if ( a > maxa ) maxa=a;
96      if ( r > maxr ) maxr=r;
97   }
98   printf("max absolute error = %g\n",maxa);
99   printf("max relative error = %g\n",maxr);
100   printf("min value = %g\n",minv);
101   printf("max value = %g\n",maxv);
102
103   printf("old number of bits per value=%ld\n",(long)bitsPerValue1);
104   printf("new number of bits per value=%ld\n",(long)bitsPerValue2);
105
106   /* get the coded message in a buffer */
107   GRIB_CHECK(grib_get_message(h,&buffer,&size),0);
108
109   /* write the buffer in a file*/
110   if(fwrite(buffer,1,size,out) != size)
111   {
112      perror(argv[1]);
113      exit(1);
114   }
115
116   /* delete handle */
117   grib_handle_delete(h);
118
119   fclose(in);
120   fclose(out);
121
122   return 0;
123 }
124
```

3.28 set.c

set.c How to set values through the key names.

```
1
10 /*
11  * C Implementation: set
12  *
13  * Description: how to set key values.
14  *
15  *
16  * Author: Enrico Fucile <enrico.fucile@ecmwf.int>
17  *
18  *
19  */
20 #include <stdio.h>
21 #include <stdlib.h>
22
23 #include "grib_api.h"
24
25 int main(int argc, char** argv) {
26   int err = 0;
27   long centre=80;
28   long long_value=0;
29   char string_value[100];
30   size_t len = sizeof(string_value)/sizeof(char);
31   size_t size=0;
32
33   FILE* in = NULL;
34   char* infile = "../../data/regular_latlon_surface.grib1";
35   FILE* out = NULL;
36   char* outfile = "out.grib1";
37   grib_handle *h = NULL;
38   const void* buffer = NULL;
39
40   in = fopen(infile,"r");
41   if(!in) {
42     printf("ERROR: unable to open file %s\n",infile);
43     return 1;
44   }
45
46   out = fopen(outfile,"w");
47   if(!in) {
48     printf("ERROR: unable to open file %s\n",outfile);
49     return 1;
50   }
51
52   /* create a new handle from a message in a file */
53   h = grib_handle_new_from_file(0,in,&err);
54   if (h == NULL) {
55     printf("Error: unable to create handle from file %s\n",infile);
56   }
57
58   /* set centre as a long */
59   GRIB_CHECK(grib_set_long(h,"centre",centre),0);
60
61   /* get centre as a long */
62   GRIB_CHECK(grib_get_long(h,"centre",&long_value),0);
63   printf("centre long value=%ld\n",long_value);
64
65   /* get centre as a string */
66   GRIB_CHECK(grib_get_string(h,"centre",string_value,&len),0);
67   printf("centre string value=%s\n",string_value);
68
69   /* get the coded message in a buffer */
70   GRIB_CHECK(grib_get_message(h,&buffer,&size),0);
```

```
71
72   /* write the buffer in a file*/
73   if(fwrite(buffer,1,size,out) != size)
74   {
75      perror(argv[1]);
76      exit(1);
77   }
78
79   /* delete handle */
80   grib_handle_delete(h);
81
82   fclose(in);
83   fclose(out);
84
85   return 0;
86 }
```

Chapter 4

Fortran 90 interface

4.1 grib_api Namespace Reference

Module grib_api.

Data Structures

- interface grib_index_get

  Get the distinct values of the key in argument contained in the index.

- interface grib_index_get_size

  Get the number of distinct values of the key in argument contained in the index.

- interface grib_index_select

  Select the message subset with key==value.

- interface grib_get

  Get the value for a key from a grib message.

- interface grib_get_size

  Get the size of an array key.

- interface grib_set

  Set the value for a key in a grib message.

- interface grib_get_element

  Get a value of specified index from an array key.

- interface grib_find_nearest

  Find the nearest point/points of a given latitude/longitude point.

- interface grib_get_data

  Get latitude/longitude and data values.

Functions

- subroutine grib_set_missing (gribid, key, status)

  Set as missing the value for a key in a grib message.

- subroutine grib_index_create (indexid, filename, keys, status)

  Create a new index form a file.

- subroutine grib_index_get_size_long (indexid, key, size, status)

  Get the number of distinct values of the key in argument contained in the index.

- subroutine grib_index_get_size_int (indexid, key, size, status)

  Get the number of distinct values of the key in argument contained in the index.

- subroutine grib_index_get_int (indexid, key, values, status)

  Get the distinct values of the key in argument contained in the index.

- subroutine grib_index_get_long (indexid, key, values, status)

  Get the distinct values of the key in argument contained in the index.

- subroutine grib_index_get_real8 (indexid, key, values, status)

  Get the distinct values of the key in argument contained in the index.

- subroutine grib_index_select_int (indexid, key, value, status)

  Select the message subset with key==value.

- subroutine grib_index_select_long (indexid, key, value, status)

  Select the message subset with key==value.

- subroutine grib_index_select_real8 (indexid, key, value, status)

  Select the message subset with key==value.

- subroutine grib_new_from_index (indexid, gribid, status)

  Create a new handle from an index after having selected the key values.

- subroutine grib_index_release (indexid, status)

  Delete the index.

- subroutine grib_open_file (ifile, filename, mode, status)

  Open a file according to a mode.

- subroutine grib_pbopen (ifile, filename, mode, status)

  Open a file according to a mode.

- subroutine grib_pbread (ifile, buffer, nbytes, status)

  Reads nbytes bytes into the buffer from a file opened with grib_open_file.

- subroutine grib_close_file (ifile, status)

  Close a file.

- subroutine grib_count_in_file (ifile, n, status)

  Counts the messages in a file.

- subroutine grib_new_from_file (ifile, gribid, status)

  Load in memory a grib message from a file.

- subroutine grib_new_from_message (gribid, message, status)

  Create a new message in memory from an integer array containting the coded message.

- subroutine grib_new_from_samples (gribid, samplename, status)

  Create a new valid gribid from a sample contained in a samples directory pointed by the environment variable GRIB_SAMPLES_PATH.

- subroutine grib_new_from_template (gribid, templatename, status)

  THIS FUNCTION IS DEPRECATED AND IT WILL DISAPPEAR FROM THE VERSION 2.0 Create a new valid gribid from a template.

- subroutine grib_release (gribid, status)

  Free the memory for the message referred as gribid.

- subroutine grib_clone (gribid_src, gribid_dest, status)

  Create a copy of a message.

- subroutine grib_copy_namespace (gribid_src, namespace, gribid_dest, status)

  Copy the value of all the keys belonging to a namespace from the source message to the destination message.

- subroutine grib_check (status, caller, string)

  Check the status returned by a subroutine.

- subroutine grib_get_data_real4 (gribid, lats, lons, values, status)

  Get latitudes/longitudes/data values (real(4)).

- subroutine grib_get_data_real8 (gribid, lats, lons, values, status)

  Get latitudes/longitudes/data values (real(8)).

- subroutine grib_keys_iterator_new (gribid, iterid, namespace, status)

  Create a new iterator on the keys.

- subroutine grib_keys_iterator_next (iterid, status)

  Advance to the next keys iterator value.

- subroutine grib_keys_iterator_delete (iterid, status)

  Delete a keys iterator and free memory.

- subroutine grib_keys_iterator_get_name (iterid, name, status)

  Get the name of a key from a keys iterator.

- subroutine grib_keys_iterator_rewind (iterid, status)

  Rewind a keys iterator.

- subroutine grib_dump (gribid, status)
  Dump the content of a grib message.
- subroutine grib_get_error_string (error, error_message, status)
  Get the error message given an error code.
- subroutine grib_get_size_int (gribid, key, size, status)
  Get the size of an array key.
- subroutine grib_get_size_long (gribid, key, size, status)
  Get the size of an array key.
- subroutine grib_get_int (gribid, key, value, status)
  Get the integer value of a key from a grib message.
- subroutine grib_get_long (gribid, key, value, status)
  Get the integer value of a key from a grib message.
- subroutine grib_is_missing (gribid, key, is_missing, status)
  Check if the value of a key is MISSING.
- subroutine grib_get_real4 (gribid, key, value, status)
  Get the real(4) value of a key from a grib message.
- subroutine grib_get_real8 (gribid, key, value, status)
  Get the real(8) value of a key from a grib message.
- subroutine grib_get_string (gribid, key, value, status)
  Get the character value of a key from a grib message.
- subroutine grib_get_int_array (gribid, key, value, status)
  Get the integer array of values for a key from a grib message.
- subroutine grib_get_long_array (gribid, key, value, status)
  Get the integer array of values for a key from a grib message.
- subroutine grib_get_real4_array (gribid, key, value, status)
  Get the real(4) array of values for a key from a grib message.
- subroutine grib_get_real8_array (gribid, key, value, status)
  Get the real(8) array of values for a key from a grib message.
- subroutine grib_get_real4_element (gribid, key, index, value, status)
  Get a real(4) value of specified index from an array key.
- subroutine grib_get_real8_element (gribid, key, index, value, status)
  Get a real(8) value of specified index from an array key.
- subroutine grib_get_real4_elements (gribid, key, index, value, status)
  Get the real(4) values whose indexes are stored in the array "index" from an array key.

- subroutine grib_get_real8_elements (gribid, key, index, value, status)

  Get the real(8) values whose indexes are stored in the array "index" from an array key.

- subroutine grib_set_int (gribid, key, value, status)

  Set the integer value for a key in a grib message.

- subroutine grib_set_long (gribid, key, value, status)

  Set the integer value for a key in a grib message.

- subroutine grib_set_real4 (gribid, key, value, status)

  Set the real(4) value for a key in a grib message.

- subroutine grib_set_real8 (gribid, key, value, status)

  Set the real(8) value for a key in a grib message.

- subroutine grib_set_int_array (gribid, key, value, status)

  Set the integers values for an array key in a grib message.

- subroutine grib_set_long_array (gribid, key, value, status)

  Set the integers values for an array key in a grib message.

- subroutine grib_set_real4_array (gribid, key, value, status)

  Set the real(4) values for an array key in a grib message.

- subroutine grib_set_real8_array (gribid, key, value, status)

  Set the real(8) values for an array key in a grib message.

- subroutine grib_set_string (gribid, key, value, status)

  Set the character value for a string key in a grib message.

- subroutine grib_get_message_size (gribid, byte_size, status)

  Get the size of a coded message.

- subroutine grib_copy_message (gribid, message, status)

  Copy the coded message into an integer array.

- subroutine grib_write (gribid, ifile, status)

  Write the coded message to a file.

- subroutine grib_multi_write (multigribid, ifile, status)

  Write a multi field message to a file.

- subroutine grib_multi_append (ingribid, startsection, multigribid, status)

  Append a single field grib message to a multi field grib message.

- subroutine grib_find_nearest_multiple (gribid, is_lsm, inlats, inlons, outlats, outlons, values, distances, indexes, status)

  Find the nearest point of a given latitude/longitude point.

- subroutine grib_find_nearest_single (gribid, is_lsm, inlat, inlon, outlat, outlon, value, distance, index, status)

  Find the nearest point of a given latitude/longitude point.

- subroutine grib_find_nearest_four_single (gribid, is_lsm, inlat, inlon, outlat, outlon, value, distance, index, status)

  Find the nearest point of a given latitude/longitude point.

- subroutine grib_multi_support_on (status)

  Turn on the support for multiple fields in a single message.

- subroutine grib_multi_support_off (status)

  Turn off the support for multiple fields in a single message.

- subroutine grib_gribex_mode_on (status)

  Turn on the compatibility mode with gribex.

- subroutine grib_gribex_mode_off (status)

  Turn off the compatibility mode with gribex.

- subroutine grib_skip_computed (iterid, status)

  Skip the computed keys in a keys iterator.

- subroutine grib_skip_coded (iterid, status)

  Skip the coded keys in a keys iterator.

- subroutine grib_skip_duplicates (iterid, status)

  Skip the duplicated keys in a keys iterator.

- subroutine grib_skip_read_only (iterid, status)

  Skip the read_only keys in a keys iterator.

4.1.1 Detailed Description

Module grib_api.

The grib_api module provides the Fortran 90 interface of the GRIB API.

4.1.2 Function Documentation

4.1.2.1 subroutine grib_api::grib_check (integer,intent(in) *status*, character(len=∗),intent(in) *caller*, character(len=∗),intent(in) *string*)

Check the status returned by a subroutine.

In case of error it stops the program, returns the error code to the shell and prints the error message.

Parameters:

status the status to be checked

caller name of the caller soubroutine

string a string variable from the caller routine (e.g. key,filename)

4.1.2.2 subroutine grib_api::grib_clone (integer(kind=kindOfInt),intent(in) *gribid_src*, integer(kind=kindOfInt),intent(out) *gribid_dest*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a copy of a message.

Create a copy of a given message (*gribid_src*) giving a new

message in memory (*gribid_dest*) exactly identical to the original one.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: clone.f90

Parameters:

gribid_src grib to be cloned

gribid_dest new grib returned

status GRIB_SUCCESS if OK, integer value on error

4.1.2.3 subroutine grib_api::grib_close_file (integer(kind=kindOfInt),intent(in) *ifile*, integer(kind=kindOfInt),intent(out),optional *status*)

Close a file.

If the *fileid* does not refer to an opened file an error code is returned in status.

Examples: get.f90

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

ifile is the id of the file to be closed.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.4 subroutine grib_api::grib_copy_message (integer(kind=kindOfInt),intent(in) *gribid*, character(len=1),dimension(:),intent(out) *message*, integer(kind=kindOfInt),intent(out),optional *status*)

Copy the coded message into an integer array.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

message integer array containing the coded message to be copied

status GRIB_SUCCESS if OK, integer value on error

4.1.2.5 subroutine grib_api::grib_copy_namespace (integer(kind=kindOfInt),intent(in) *gribid_src*, character(LEN=*),intent(in) *namespace*, integer(kind=kindOfInt),intent(in) *gribid_dest*, integer(kind=kindOfInt),intent(out),optional *status*)

Copy the value of all the keys belonging to a namespace from the source message to the destination message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid_src source message

gribid_dest destination message

namespace namespace to be copied

status GRIB_SUCCESS if OK, integer value on error

4.1.2.6 subroutine grib_api::grib_count_in_file (integer(kind=kindOfInt),intent(in) *ifile*, integer(kind=kindOfInt),intent(out) *n*, integer(kind=kindOfInt),intent(out),optional *status*)

Counts the messages in a file.

Examples: count_messages.f90

Parameters:

ifile id of the file opened with grib_open_file

n number of messages in the file

status GRIB_SUCCESS if OK or error code

4.1.2.7 subroutine grib_api::grib_dump (integer(kind=kindOfInt),intent(in) *gribid*, integer(kind=kindOfInt),intent(out),optional *status*)

Dump the content of a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

status GRIB_SUCCESS if OK, integer value on error

4.1.2.8 subroutine grib_api::grib_find_nearest_four_single (integer(kind=kindOfInt),intent(in) *gribid*, logical,intent(in) *is_lsm*, real(kind = kindOfDouble),intent(in) *inlat*, real(kind = kindOfDouble),intent(in) *inlon*, real(kind = kindOfDouble),dimension(4),intent(out) *outlat*, real(kind = kindOfDouble),dimension(4),intent(out) *outlon*, real(kind = kindOfDouble),dimension(4),intent(out) *value*, real(kind = kindOfDouble),dimension(4),intent(out) *distance*, integer(kind = kindOfInt),dimension(4),intent(out) *index*, integer(kind=kindOfInt),intent(out),optional *status*)

Find the nearest point of a given latitude/longitude point.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

is_lsm .true. if the nearest land point is required otherwise .false.

inlat latitude of the point

inlon longitudes of the point

outlat latitude of the nearest point

outlon longitude of the nearest point

distance distance between the given point and its nearest

index zero based index

value value of the field in the nearest point

status GRIB_SUCCESS if OK, integer value on error

4.1.2.9 subroutine grib_api::grib_find_nearest_multiple (integer(kind=kindOfInt),intent(in) *gribid*, logical,intent(in) *is_lsm*, real(kind = kindOfDouble),dimension(:),intent(in) *inlats*, real(kind = kindOfDouble),dimension(:),intent(in) *inlons*, real(kind = kindOfDouble),dimension(:),intent(out) *outlats*, real(kind = kindOfDouble),dimension(:),intent(out) *outlons*, real(kind = kindOfDouble),dimension(:),intent(out) *values*, real(kind = kindOfDouble),dimension(:),intent(out) *distances*, integer(kind = kindOfInt),dimension(:),intent(out) *indexes*, integer(kind=kindOfInt),intent(out),optional *status*)

Find the nearest point of a given latitude/longitude point.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

is_lsm .true. if the nearest land point is required otherwise .false.

inlats input real(8) array of the latitudes of the points

inlons input real(8) array of the longitudes of the points

outlats output real(8) array of the latitudes of the nearest points

outlons output real(8) array of the longitudes of the nearest points

distances output real(8) array of the distances

indexes output integer(4) array of the zero based indexes

values output real(8) array of the values

status GRIB_SUCCESS if OK, integer value on error

4.1.2.10 subroutine grib_api::grib_find_nearest_single (integer(kind=kindOfInt),intent(in) *gribid*, logical,intent(in) *is_lsm*, real(kind = kindOfDouble),intent(in) *inlat*, real(kind = kindOfDouble),intent(in) *inlon*, real(kind = kindOfDouble),intent(out) *outlat*, real(kind = kindOfDouble),intent(out) *outlon*, real(kind = kindOfDouble),intent(out) *value*, real(kind = kindOfDouble),intent(out) *distance*, integer(kind = kindOfInt),intent(out) *index*, integer(kind=kindOfInt),intent(out),optional *status*)

Find the nearest point of a given latitude/longitude point.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

is_lsm .true. if the nearest land point is required otherwise .false.

inlat latitude of the point

inlon longitudes of the point

outlat latitude of the nearest point

outlon longitude of the nearest point

distance distance between the given point and its nearest

index zero based index

value value of the field in the nearest point

status GRIB_SUCCESS if OK, integer value on error

4.1.2.11 subroutine grib_api::grib_get_data_real4 (integer(kind=kindOfInt),intent(in) *gribid*, real (kind = kindOfFloat),dimension(:),intent(out) *lats*, real (kind = kindOfFloat),dimension(:),intent(out) *lons*, real (kind = kindOfFloat),dimension(:),intent(out) *values*, integer(kind=kindOfInt),intent(out),optional *status*)

Get latitudes/longitudes/data values (real(4)).

Latitudes, longitudes, data values arrays are returned. They must be properly allocated by the caller and their required dimension can be obtained with grib_get_size or by getting (with grib_get) the value of the integer key "numberOfPoints".

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

lats latitudes array with dimension "size"

lons longitudes array with dimension "size"

values data values array with dimension "size"

status GRIB_SUCCESS if OK, integer value on error

4.1.2.12 subroutine grib_api::grib_get_data_real8 (integer(kind=kindOfInt),intent(in) *gribid*, real (kind = kindOfDouble),dimension(:),intent(out) *lats*, real (kind = kindOfDouble),dimension(:),intent(out) *lons*, real (kind = kindOfDouble),dimension(:),intent(out) *values*, integer(kind=kindOfInt),intent(out),optional *status*)

Get latitudes/longitudes/data values (real(8)).

Latitudes, longitudes, data values arrays are returned. They must be properly allocated by the calling program/function. Their required dimension can be obtained by getting (with grib_get) the value of the integer key "numberOfPoints". In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

lats latitudes array

lons longitudes array

values data values array

status GRIB_SUCCESS if OK, integer value on error

4.1.2.13 subroutine grib_api::grib_get_error_string (integer(kind=kindOfInt),intent(in) *error*, character(len=∗),intent(out) *error_message*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the error message given an error code.

Parameters:

error error code

error_message error message

status GRIB_SUCCESS if OK, integer value on error

4.1.2.14 subroutine grib_api::grib_get_int (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind = kindOfInt),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the integer value of a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value the integer(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.15 subroutine grib_api::grib_get_int_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the integer array of values for a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.16 subroutine grib_api::grib_get_long (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind = kindOfLong),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the integer value of a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value the integer(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.17 subroutine grib_api::grib_get_long_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfLong),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the integer array of values for a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.18 subroutine grib_api::grib_get_message_size (integer(kind=kindOfInt),intent(in) *gribid*, integer(kind=kindOfSize_t),intent(out) *byte_size*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the size of a coded message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

byte_size size in bytes of the message

status GRIB_SUCCESS if OK, integer value on error

4.1.2.19 subroutine grib_api::grib_get_real4 (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfFloat),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(4) value of a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value the real(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.20 subroutine grib_api::grib_get_real4_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfFloat),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(4) array of values for a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.21 subroutine grib_api::grib_get_real4_element (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),intent(in) *index*, real(kind = kindOfFloat),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get a real(4) value of specified index from an array key.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

index integer(4) index

value real(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.22 subroutine grib_api::grib_get_real4_elements (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),dimension(:),intent(in) *index*, real(kind = kindOfFloat),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(4) values whose indexes are stored in the array "index" from an array key.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

index integer(4) array indexes

value real(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.23 subroutine grib_api::grib_get_real8 (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfDouble),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(8) value of a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value the real(8) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.24 subroutine grib_api::grib_get_real8_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfDouble),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(8) array of values for a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(8) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.25 subroutine grib_api::grib_get_real8_element (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),intent(in) *index*, real(kind = kindOfDouble),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get a real(8) value of specified index from an array key.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

index integer(4) index

value real(8) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.26 subroutine grib_api::grib_get_real8_elements (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),dimension(:),intent(in) *index*, real(kind = kindOfDouble),dimension(:),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the real(8) values whose indexes are stored in the array "index" from an array key.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

index integer(4) array index

value real(8) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.27 subroutine grib_api::grib_get_size_int (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),intent(out) *size*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the size of an array key.

To get the size of a key representing an array.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key name of the key

size size of the array key

status GRIB_SUCCESS if OK, integer value on error

4.1.2.28 subroutine grib_api::grib_get_size_long (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind=kindOfLong),intent(out) *size*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the size of an array key.

To get the size of a key representing an array.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key name of the key

size size of the array key

status GRIB_SUCCESS if OK, integer value on error

4.1.2.29 subroutine grib_api::grib_get_string (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, character(len=∗),intent(out) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the character value of a key from a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value the real(8) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.30 subroutine grib_api::grib_gribex_mode_off (integer(kind=kindOfInt),intent(out),optional *status*)

Turn off the compatibility mode with gribex.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

status GRIB_SUCCESS if OK, integer value on error

4.1.2.31 subroutine grib_api::grib_gribex_mode_on (integer(kind=kindOfInt),intent(out),optional *status*)

Turn on the compatibility mode with gribex.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

status GRIB_SUCCESS if OK, integer value on error

4.1.2.32 subroutine grib_api::grib_index_create (integer(kind=kindOfInt),intent(inout) *indexid*, character(len=∗),intent(in) *filename*, character(len=∗),intent(in) *keys*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a new index form a file.

The file is indexed with the keys in argument.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of the newly created index

filename name of the file of messages to be indexed

keys : comma separated list of keys for the index. The type of the key can be explicitly declared appending :l for long, :d for double, :s for string to the key name. If the type is not declared explicitly, the native type is assumed.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.33 subroutine grib_api::grib_index_get_int (integer(kind=kindOfInt),intent(in) *indexid*, character(len=∗),intent(in) *key*, integer(kind=kindOfInt),dimension(:),intent(out) *values*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the distinct values of the key in argument contained in the index.

The key must belong to the index. This function is used when the type of the key was explicitly defined as long or when the native type of the key is long.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for wich the values are returned

values array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.34 subroutine grib_api::grib_index_get_long (integer(kind=kindOfInt),intent(in) *indexid*, character(len=∗),intent(in) *key*, integer(kind=kindOfLong),dimension(:),intent(out) *values*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the distinct values of the key in argument contained in the index.

The key must belong to the index. This function is used when the type of the key was explicitly defined as long or when the native type of the key is long.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for wich the values are returned

values array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.35 subroutine grib_api::grib_index_get_real8 (integer(kind=kindOfInt),intent(in) *indexid*, character(len=*),intent(in) *key*, real(kind=kindOfDouble),dimension(:),intent(out) *values*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the distinct values of the key in argument contained in the index.

The key must belong to the index. This function is used when the type of the key was explicitly defined as long or when the native type of the key is long.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for wich the values are returned

values array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.36 subroutine grib_api::grib_index_get_size_int (integer(kind=kindOfInt),intent(in) *indexid*, character(len=*),intent(in) *key*, integer(kind=kindOfInt),intent(out) *size*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the number of distinct values of the key in argument contained in the index.

The key must belong to the index.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for which the number of values is computed

size number of distinct values of the key in the index

status GRIB_SUCCESS if OK, integer value on error

4.1.2.37 subroutine grib_api::grib_index_get_size_long (integer(kind=kindOfInt),intent(in) *indexid*, character(len=*),intent(in) *key*, integer(kind=kindOfLong),intent(out) *size*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the number of distinct values of the key in argument contained in the index.

The key must belong to the index.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for which the number of values is computed

size number of distinct values of the key in the index

status GRIB_SUCCESS if OK, integer value on error

4.1.2.38 subroutine grib_api::grib_index_release (integer(kind=kindOfInt),intent(in) *indexid*, integer(kind=kindOfInt),intent(out),optional *status*)

Delete the index.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

indexid id of an index created from a file.

status GRIB_SUCCESS if OK, integer value on error

4.1.2.39 subroutine grib_api::grib_index_select_int (integer(kind=kindOfInt),intent(in) *indexid*, character(len=*),intent(in) *key*, integer(kind=kindOfInt),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Select the message subset with key==value.

The value is a integer. The key must have been created with integer type or have integer as native type if the type was not explicitly defined in the index creation.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key to be selected

value value of the key to select

status GRIB_SUCCESS if OK, integer value on error

4.1.2.40 subroutine grib_api::grib_index_select_long (integer(kind=kindOfInt),intent(in) *indexid*, character(len=*),intent(in) *key*, integer(kind=kindOfLong),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Select the message subset with key==value.

The value is a integer. The key must have been created with integer type or have integer as native type if the type was not explicitly defined in the index creation.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key to be selected

value value of the key to select

status GRIB_SUCCESS if OK, integer value on error

4.1.2.41 subroutine grib_api::grib_index_select_real8 (integer(kind=kindOfInt),intent(in) *indexid*, character(len=∗),intent(in) *key*, real(kind=kindOfDouble),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Select the message subset with key==value.

The value is a real. The key must have been created with real type or have real as native type if the type was not explicitly defined in the index creation.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key to be selected

value value of the key to select

status GRIB_SUCCESS if OK, integer value on error

4.1.2.42 subroutine grib_api::grib_is_missing (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, integer(kind = kindOfInt),intent(out) *is_missing*, integer(kind=kindOfInt),intent(out),optional *status*)

Check if the value of a key is MISSING.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

is_missing 0->not missing, 1->missing

status GRIB_SUCCESS if OK, integer value on error

4.1.2.43 subroutine grib_api::grib_keys_iterator_delete (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Delete a keys iterator and free memory.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

iterid keys iterator id created with grib_keys_iterator_new

status GRIB_SUCCESS if OK, integer value on error

4.1.2.44 subroutine grib_api::grib_keys_iterator_get_name (integer(kind=kindOfInt),intent(in) *iterid*, character(LEN=∗),intent(out) *name*, integer(kind=kindOfInt),intent(out),optional *status*)

Get the name of a key from a keys iterator.

If the status parameter (optional) is not given the program will exit with an error message

otherwise the error message can be gathered with grib_get_error_string.

Parameters:

iterid keys iterator id created with grib_keys_iterator_new

name key name to be retrieved

status GRIB_SUCCESS if OK, integer value on error

4.1.2.45 subroutine grib_api::grib_keys_iterator_new (integer(kind=kindOfInt),intent(in) *gribid*, integer(kind=kindOfInt),intent(out) *iterid*, character(LEN=∗),intent(in) *namespace*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a new iterator on the keys.

The keys iterator can be navigated to give all the key names which can then be used to get or set the key values with grib_get or grib_set. The set of keys returned can be controlled with the input variable namespace or using the functions grib_skip_read_only, grib_skip_duplicates, grib_skip_coded,grib_skip_computed. If namespace is a non empty string only the keys belonging to that namespace are returned. Available namespaces are "ls" (to get the same default keys as the grib_ls and "mars" to get the keys used by mars.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

iterid keys iterator id to be used in the keys iterator functions

namespace the namespace of the keys to search for (all the keys if empty)

status GRIB_SUCCESS if OK, integer value on error

4.1.2.46 subroutine grib_api::grib_keys_iterator_next (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Advance to the next keys iterator value.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

iterid keys iterator id created with grib_keys_iterator_new

status GRIB_SUCCESS if OK, integer value on error

4.1.2.47 subroutine grib_api::grib_keys_iterator_rewind (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Rewind a keys iterator.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

iterid keys iterator id created with grib_keys_iterator_new

status GRIB_SUCCESS if OK, integer value on error

4.1.2.48 subroutine grib_api::grib_multi_append (integer(kind=kindOfInt),intent(in) *ingribid*, integer(kind=kindOfInt),intent(in) *startsection*, integer(kind=kindOfInt),intent(out) *multigribid*, integer(kind=kindOfInt),intent(out),optional *status*)

Append a single field grib message to a multi field grib message.

Only the sections with section number greather or equal "startsection" are copied from the input single message to the multi field output grib.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

ingribid id of the input single grib

startsection starting from startsection (included) all the sections are copied from the input single grib to the output multi grib

multigribid id of the output multi filed grib

status GRIB_SUCCESS if OK, integer value on error

4.1.2.49 subroutine grib_api::grib_multi_support_off (integer(kind=kindOfInt),intent(out),optional *status*)

Turn off the support for multiple fields in a single message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

status GRIB_SUCCESS if OK, integer value on error

4.1.2.50 subroutine grib_api::grib_multi_support_on (integer(kind=kindOfInt),intent(out),optional *status*)

Turn on the support for multiple fields in a single message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

status GRIB_SUCCESS if OK, integer value on error

4.1.2.51 subroutine grib_api::grib_multi_write (integer(kind=kindOfInt),intent(in) *multigribid*, integer(kind=kindOfInt),intent(in) *ifile*, integer(kind=kindOfInt),intent(out),optional *status*)

Write a multi field message to a file.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

multigribid id of the multi field grib loaded in memory

ifile file id of a file opened with grib_open_file

status GRIB_SUCCESS if OK, integer value on error

4.1.2.52 subroutine grib_api::grib_new_from_file (integer(kind=kindOfInt),intent(in) *ifile*, integer(kind=kindOfInt),intent(out) *gribid*, integer(kind=kindOfInt),intent(out),optional *status*)

Load in memory a grib message from a file.

The message can be accessed through its gribid and it will be available

until grib_release is called.

Examples: get.f90

Parameters:

ifile id of the file opened with grib_open_file

gribid id of the grib loaded in memory

status GRIB_SUCCESS if OK, GRIB_END_OF_FILE at the end of file, or error code

4.1.2.53 subroutine grib_api::grib_new_from_index (integer(kind=kindOfInt),intent(in) *indexid*, integer(kind=kindOfInt),intent(out) *gribid*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a new handle from an index after having selected the key values.

All the keys belonging to the index must be selected before calling this function. Successive calls to this function will return all the handles compatible with the constraints defined selecting the values of the index keys. When no more handles are available from the index a NULL pointer is returned and the err variable is set to GRIB_END_OF_INDEX.

The message can be accessed through its gribid and it will be available

until grib_release is called.

Examples: index.f90

Parameters:

indexid id of an index created from a file.

gribid id of the grib loaded in memory

status GRIB_SUCCESS if OK, GRIB_END_OF_FILE at the end of file, or error code

4.1.2.54 subroutine grib_api::grib_new_from_message (integer(kind=kindOfInt),intent(out) *gribid*, character,dimension(:),intent(in) *message*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a new message in memory from an integer array containting the coded message.

The message can be accessed through its gribid and it will be available

until grib_release is called. A reference to the original coded

message is kept in the new message structure.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: copy_message.f90

Parameters:

gribid id of the grib loaded in memory

message integer(4) array containing the coded message

status GRIB_SUCCESS if OK, integer value on error

4.1.2.55 subroutine grib_api::grib_new_from_samples (integer(kind=kindOfInt),intent(out) *gribid*, character(len=*),intent(in) *samplename*, integer(kind=kindOfInt),intent(out),optional *status*)

Create a new valid gribid from a sample contained in a samples directory pointed by the environment variable GRIB_SAMPLES_PATH.

To know where the samples directory is run the grib_info tool.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: sample.f90

Parameters:

gribid id of the grib loaded in memory

samplename name of the sample to be used

status GRIB_SUCCESS if OK, integer value on error

4.1.2.56 subroutine grib_api::grib_new_from_template (integer(kind=kindOfInt),intent(out) *gribid*, character(len=∗),intent(in) *templatename*, integer(kind=kindOfInt),intent(out),optional *status*)

THIS FUNCTION IS DEPRECATED AND IT WILL DISAPPEAR FROM THE VERSION 2.0 Create a new valid gribid from a template.

Valid templates are stored in the directory pointed by the

environment variable GRIB_TEMPLATES_PATH or in a templates

default directory if this variable is not defined.

To know where the templates directory is run the grib_info tool.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: template.f90

Parameters:

gribid id of the grib loaded in memory

templatename name of the template to be used

status GRIB_SUCCESS if OK, integer value on error

4.1.2.57 subroutine grib_api::grib_open_file (integer(kind=kindOfInt),intent(out) *ifile*, character(len=∗),intent(in) *filename*, character(LEN=∗),intent(in) *mode*, integer(kind=kindOfInt),intent(out),optional *status*)

Open a file according to a mode.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: get.f90

Parameters:

ifile id of the opened file to be used in all the file functions.

filename name of the file to be open

mode open mode can be 'r' (read only) or 'w' (write only)

status GRIB_SUCCESS if OK, integer value on error

4.1.2.58 subroutine grib_api::grib_pbopen (integer(kind=kindOfInt),intent(out) *ifile*, character(len=*),intent(in) *filename*, character(LEN=*),intent(in) *mode*, integer(kind=kindOfInt),intent(out),optional *status*)

Open a file according to a mode.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: get.f90

Parameters:

ifile id of the opened file to be used in all the file functions.

filename name of the file to be open

mode open mode can be 'r' (read only) or 'w' (write only)

status GRIB_SUCCESS if OK, integer value on error

4.1.2.59 subroutine grib_api::grib_pbread (integer(kind=kindOfInt),intent(in) *ifile*, character(len=1),intent(out) *buffer*, integer(kind=kindOfInt),intent(in) *nbytes*, integer(kind=kindOfInt),intent(out),optional *status*)

Reads nbytes bytes into the buffer from a file opened with grib_open_file.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

ifile id of the opened file to be used in all the file functions.

buffer binary buffer to be read

nbytes number of bytes to be read

status GRIB_SUCCESS if OK, integer value on error

4.1.2.60 subroutine grib_api::grib_release (integer(kind=kindOfInt),intent(in) *gribid*, integer(kind=kindOfInt),intent(out),optional *status*)

Free the memory for the message referred as gribid.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: get.f90

Parameters:

gribid id of the grib loaded in memory

status GRIB_SUCCESS if OK, integer value on error

4.1.2.61 subroutine grib_api::grib_set_int (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, integer(kind=kindOfInt),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the integer value for a key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.62 subroutine grib_api::grib_set_int_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, integer(kind=kindOfInt),dimension(:),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the integers values for an array key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.63 subroutine grib_api::grib_set_long (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, integer(kind=kindOfLong),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the integer value for a key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.64 subroutine grib_api::grib_set_long_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, integer(kind=kindOfLong),dimension(:),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the integers values for an array key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value integer(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.65 subroutine grib_api::grib_set_missing (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, integer(kind=kindOfInt),intent(out),optional *status*)

Set as missing the value for a key in a grib message.

It can be used to set a missing value in the grib header but not in

the data values. To set missing data values see the bitmap examples.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: set_missing.f90

Parameters:

gribid id of the grib loaded in memory

key key name

status GRIB_SUCCESS if OK, integer value on error

4.1.2.66 subroutine grib_api::grib_set_real4 (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, real(kind = kindOfFloat),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the real(4) value for a key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(4) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.67 subroutine grib_api::grib_set_real4_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfFloat),dimension(:),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the real(4) values for an array key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(4) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.68 subroutine grib_api::grib_set_real8 (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfDouble),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the real(8) value for a key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(8) value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.69 subroutine grib_api::grib_set_real8_array (integer(kind=kindOfInt),intent(in) *gribid*, character(len=∗),intent(in) *key*, real(kind = kindOfDouble),dimension(:),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the real(8) values for an array key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value real(8) array value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.70 subroutine grib_api::grib_set_string (integer(kind=kindOfInt),intent(in) *gribid*, character(len=*),intent(in) *key*, character(len=*),intent(in) *value*, integer(kind=kindOfInt),intent(out),optional *status*)

Set the character value for a string key in a grib message.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key key name

value character value

status GRIB_SUCCESS if OK, integer value on error

4.1.2.71 subroutine grib_api::grib_skip_coded (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Skip the coded keys in a keys iterator.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

The coded keys are actually coded in the message.

See also:

grib_keys_iterator_new,grib_keys_iterator_next,grib_keys_iterator_release

Parameters:

iterid keys iterator id

status GRIB_SUCCESS if OK, integer value on error

4.1.2.72 subroutine grib_api::grib_skip_computed (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Skip the computed keys in a keys iterator.

The computed keys are not coded in the message they are computed from other keys.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

See also:

grib_keys_iterator_new,grib_keys_iterator_next,grib_keys_iterator_release

Parameters:

iterid keys iterator id

status GRIB_SUCCESS if OK, integer value on error

4.1.2.73 subroutine grib_api::grib_skip_duplicates (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Skip the duplicated keys in a keys iterator.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

See also:

grib_keys_iterator_new,grib_keys_iterator_next,grib_keys_iterator_release

Parameters:

iterid keys iterator id

status GRIB_SUCCESS if OK, integer value on error

4.1.2.74 subroutine grib_api::grib_skip_read_only (integer(kind=kindOfInt),intent(in) *iterid*, integer(kind=kindOfInt),intent(out),optional *status*)

Skip the read_only keys in a keys iterator.

Read only keys cannot be set.

See also:

grib_keys_iterator_new,grib_keys_iterator_next,grib_keys_iterator_release

Parameters:

iterid keys iterator id

status GRIB_SUCCESS if OK, integer value on error

4.1.2.75 subroutine grib_api::grib_write (integer(kind=kindOfInt),intent(in) *gribid*, integer(kind=kindOfInt),intent(in) *ifile*, integer(kind=kindOfInt),intent(out),optional *status*)

Write the coded message to a file.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

ifile file id of a file opened with grib_open_file

status GRIB_SUCCESS if OK, integer value on error

4.2 grib_find_nearest Interface Reference

Find the nearest point/points of a given latitude/longitude point.

Public Member Functions

- subroutine **grib_find_nearest_single** (gribid, is_lsm, inlat, inlon, outlat, outlon, value, distance, index, status)
- subroutine **grib_find_nearest_four_single** (gribid, is_lsm, inlat, inlon, outlat, outlon, value, distance, index, status)
- subroutine **grib_find_nearest_multiple** (gribid, is_lsm, inlats, inlons, outlats, outlons, values, distances, indexes, status)

4.2.1 Detailed Description

Find the nearest point/points of a given latitude/longitude point.

The value in the nearest point (or the four nearest points) is returned as well as the zero based index (which can be used in grib_get_element) and its distance from the given point using the following formula (acos(sin(lat1)∗sin(lat2)+cos(lat1)∗cos(lat2)∗cos(lon1-lon2))).

If the is_lsm flag is .true. the input field gribid is considered as a land sea mask and the nearest land point is returned.

The nearest land point among the four neighbours is:

- the nearest point with land sea mask value >= 0.5
- the nearest without any other condition if all the four have land sea mask value <0.5.

Arrays (real(8)) of latitude/longitude can be provided to find with one call the values,indexes and distances for all the lat/lon points listed in the arrays.

If a single latitude/longitude point is provided and outlat,outlon,value,distance,index are defined as arrays with four elements the lat/lon coordinates and values, distances and indexes of the four nearest points are returned.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: nearest.f90

Parameters:

← ***gribid*** id of the grib loaded in memory
← ***is_lsm*** .true. if the nearest land point is required otherwise .false.
← ***inlat*** latitude of the point in degrees
← ***inlon*** longitudes of the point in degrees
→ ***outlat*** latitude of the nearest point in degrees
→ ***outlon*** longitude of the nearest point in degrees
→ ***distance*** distance between the given point and its nearest
→ ***index*** zero based index
→ ***value*** value of the field in the nearest point
→ ***status*** GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.3 grib_get Interface Reference

Get the value for a key from a grib message.

Public Member Functions

- subroutine **grib_get_int** (gribid, key, value, status)
- subroutine **grib_get_real4** (gribid, key, value, status)
- subroutine **grib_get_real8** (gribid, key, value, status)
- subroutine **grib_get_string** (gribid, key, value, status)
- subroutine **grib_get_int_array** (gribid, key, value, status)
- subroutine **grib_get_real4_array** (gribid, key, value, status)
- subroutine **grib_get_real8_array** (gribid, key, value, status)

4.3.1 Detailed Description

Get the value for a key from a grib message.

Given a *gribid* and *key* as input a *value* for the *key* is returned. In some cases the *value* can be an array rather than a scalar. As examples of array keys we have "values","pl", "pv" respectively the data values, the list of number of points for each latitude in a reduced grid and the list of vertical levels. In these cases the *value* array must be allocated by the caller and their required dimension can be obtained with grib_get_size.

The *value* can be integer(4), real(4), real(8), character. Although each key has its own native type, a key of type integer can be retrieved (with grib_get) as real(4), real(8) or character. Analogous conversions are always provided when possible. Illegal conversions are real to integer and character to any other type.

The *gribid* references to a grib message loaded in memory.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: get.f90, print_data.f90

See also:

grib_new_from_file, grib_release, grib_set

Parameters:

← ***gribid*** id of the grib loaded in memory

← ***key*** key name

→ ***value*** value can be a scalar or array of integer(4),real(4),real(8),character

→ ***status*** GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.4 grib_get_data Interface Reference

Get latitude/longitude and data values.

Public Member Functions

- subroutine **grib_get_data_real4** (gribid, lats, lons, values, status)
- subroutine **grib_get_data_real8** (gribid, lats, lons, values, status)

4.4.1 Detailed Description

Get latitude/longitude and data values.

Latitudes, longitudes, data values arrays are returned. They must be properly allocated by the caller and their required dimension can be obtained with grib_get_size or by getting (with grib_get) the value of the integer key "numberOfPoints".

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: get_data.f90

Parameters:

← ***gribid*** id of the grib loaded in memory

→ ***lats*** latitudes array with dimension "size"

→ ***lons*** longitudes array with dimension "size"

→ ***values*** data values array with dimension "size"

→ ***status*** GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.5 grib_get_element Interface Reference

Get a value of specified index from an array key.

Public Member Functions

- subroutine **grib_get_real4_element** (gribid, key, index, value, status)
- subroutine **grib_get_real8_element** (gribid, key, index, value, status)
- subroutine **grib_get_real4_elements** (gribid, key, index, value, status)
- subroutine **grib_get_real8_elements** (gribid, key, index, value, status)

4.5.1 Detailed Description

Get a value of specified index from an array key.

Given a gribid and key name as input a value corresponding to the given index is returned. The index is zero based i.e. the first element has zero index, the second element index one and so on. If the parameter index is an array all the values correspondig to the indexes list is returned. The gribid references to a grib message loaded in memory.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: nearest.f90

See also:

grib_new_from_file, grib_release, grib_get

Parameters:

← ***gribid*** id of the grib loaded in memory

← ***key*** key name

← ***index*** index can be a scalar or array of integer(4)

→ ***value*** value can be a scalar or array of integer(4),real(4),real(8)

→ ***status*** GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.6 grib_get_size Interface Reference

Get the size of an array key.

Public Member Functions

- subroutine **grib_get_size_int** (gribid, key, size, status)

4.6.1 Detailed Description

Get the size of an array key.

To get the size of a key representing an array.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Parameters:

gribid id of the grib loaded in memory

key name of the key

size size of the array key

status GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.7 grib_index_get Interface Reference

Get the distinct values of the key in argument contained in the index.

Public Member Functions

- subroutine **grib_index_get_int** (indexid, key, values, status)
- subroutine **grib_index_get_real8** (indexid, key, values, status)

4.7.1 Detailed Description

Get the distinct values of the key in argument contained in the index.

The key must belong to the index.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for wich the values are returned

values array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

status GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.8 grib_index_get_size Interface Reference

Get the number of distinct values of the key in argument contained in the index.

Public Member Functions

- subroutine **grib_index_get_size_int** (indexid, key, size, status)

4.8.1 Detailed Description

Get the number of distinct values of the key in argument contained in the index.

The key must belong to the index.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key for which the number of values is computed

size number of distinct values of the key in the index

status GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.9 grib_index_select Interface Reference

Select the message subset with key==value.

Public Member Functions

- subroutine **grib_index_select_int** (indexid, key, value, status)
- subroutine **grib_index_select_real8** (indexid, key, value, status)

4.9.1 Detailed Description

Select the message subset with key==value.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: index.f90

Parameters:

indexid id of an index created from a file. The index must have been created with the key in argument.

key key to be selected

value value of the key to select

status GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

4.10 grib_set Interface Reference

Set the value for a key in a grib message.

Public Member Functions

- subroutine **grib_set_int** (gribid, key, value, status)
- subroutine **grib_set_real4** (gribid, key, value, status)
- subroutine **grib_set_real8** (gribid, key, value, status)
- subroutine **grib_set_string** (gribid, key, value, status)
- subroutine **grib_set_int_array** (gribid, key, value, status)
- subroutine **grib_set_real4_array** (gribid, key, value, status)
- subroutine **grib_set_real8_array** (gribid, key, value, status)

4.10.1 Detailed Description

Set the value for a key in a grib message.

The given *value* is set for the *key* in the *gribid* message. In some cases the *value* can be an array rather than a scalar. As examples of array keys we have "values","pl", "pv" respectively the data values, the list of number of points for each latitude in a reduced grid and the list of vertical levels. In these cases the *value* array must be allocated by the caller and their required dimension can be obtained with grib_get_size.

The gribid references to a grib message loaded in memory.

In case of error, if the status parameter (optional) is not given, the program will exit with an error message.

Otherwise the error message can be gathered with grib_get_error_string.

Examples: set.f90

See also:

grib_new_from_file, grib_release, grib_get

Parameters:

← ***gribid*** id of the grib loaded in memory

← ***key*** key name

→ ***value*** value can be a scalar or array of integer(4),real(4),real(8)

→ ***status*** GRIB_SUCCESS if OK, integer value on error

The documentation for this interface was generated from the following file:

- grib_f90.f90

Chapter 5

C interface

5.1 grib_api Modules

Here is a list of all modules:

5.2 The grib_index

Typedefs

- typedef struct grib_index grib_index

Functions

- grib_index * grib_index_new_from_file (grib_context *c, char *filename, const char *keys, int *err)

  Create a new index form a file.

- 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_long (grib_index *index, const char *key, long *values, size_t *size)

  Get the 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.

- int grib_index_get_string (grib_index *index, const char *key, char **values, size_t *size)

Get the distinct values of the key in argument contained in the index.

- int grib_index_select_long (grib_index *index, const char *key, long value)

  Select the message subset with key==value.

- int grib_index_select_double (grib_index *index, const char *key, double value)

  Select the message subset with key==value.

- int grib_index_select_string (grib_index *index, const char *key, char *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.

- void grib_index_delete (grib_index *index)

  Delete the index.

5.2.1 Detailed Description

The grib_index is the structure giving indexed access to messages in a file.

5.2.2 Typedef Documentation

5.2.2.1 typedef struct grib_index grib_index

index structure to access messages in a file.

5.2.3 Function Documentation

5.2.3.1 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.

All the keys belonging to the index must be selected before calling this function. Successive calls to this function will return all the handles compatible with the constraints defined selecting the values of the index keys. When no more handles are available from the index a NULL pointer is returned and the err variable is set to GRIB_END_OF_INDEX.

Parameters:

index : an index created from a file.

err : 0 if OK, integer value on error. GRIB_END_OF_INDEX when no more handles are contained in the index.

Returns:

grib handle.

5.2.3.2 void grib_index_delete (grib_index * *index*)

Delete the index.

Parameters:

index : index to be deleted.

5.2.3.3 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.

The key must belong to the index. This function is used when the type of the key was explicitly defined as double or when the native type of the key is double.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.

key : key for wich the values are returned

values : array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

size : size of the values array

Returns:

0 if OK, integer value on error

5.2.3.4 int grib_index_get_long (grib_index * *index*, const char * *key*, long * *values*, size_t * *size*)

Get the distinct values of the key in argument contained in the index.

The key must belong to the index. This function is used when the type of the key was explicitly defined as long or when the native type of the key is long.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.

key : key for wich the values are returned

values : array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

size : size of the values array

Returns:

0 if OK, integer value on error

5.2.3.5 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.

The key must belong to the index.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.

key : key for which the number of values is computed

size : number of distinct values of the key in the index

Returns:

0 if OK, integer value on error

Examples:

index.f90.

5.2.3.6 int grib_index_get_string (grib_index * *index*, const char * *key*, char ** *values*, size_t * *size*)

Get the distinct values of the key in argument contained in the index.

The key must belong to the index. This function is used when the type of the key was explicitly defined as string or when the native type of the key is string.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.

key : key for wich the values are returned

values : array of values. The array must be allocated before entering this function and its size must be enough to contain all the values.

size : size of the values array

Returns:

0 if OK, integer value on error

5.2.3.7 grib_index* grib_index_new_from_file (grib_context * *c*, char * *filename*, const char * *keys*, int * *err*)

Create a new index form a file.

The file is indexed with the keys in argument.

Parameters:

c : context (NULL for default context)

filename : name of the file of messages to be indexed

keys : comma separated list of keys for the index. The type of the key can be explicitly declared appending :l for long, :d for double, :s for string to the key name. If the type is not declared explicitly, the native type is assumed.

err : 0 if OK, integer value on error

Returns:

the newly created index

5.2.3.8 int grib_index_select_double (grib_index * *index*, const char * *key*, double *value*)

Select the message subset with key==value.

The value is a double. The key must have been created with double type or have double as native type if the type was not explicitly defined in the index creation.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.
key : key to be selected
value : value of the key to select

Returns:

0 if OK, integer value on error

5.2.3.9 int grib_index_select_long (grib_index * *index*, const char * *key*, long *value*)

Select the message subset with key==value.

The value is a long. The key must have been created with long type or have long as native type if the type was not explicitly defined in the index creation.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.
key : key to be selected
value : value of the key to select

Returns:

0 if OK, integer value on error

5.2.3.10 int grib_index_select_string (grib_index * *index*, const char * *key*, char * *value*)

Select the message subset with key==value.

The value is a string. The key must have been created with string type or have string as native type if the type was not explicitly defined in the index creation.

Parameters:

index : an index created from a file. The index must have been created with the key in argument.
key : key to be selected
value : value of the key to select

Returns:

0 if OK, integer value on error

5.3 The grib_handle

Typedefs

- typedef struct grib_handle grib_handle
- typedef struct grib_multi_handle grib_multi_handle

Functions

- int grib_count_in_file (grib_context *c, FILE *f, int *n)

  Counts the messages contained in a file resource.

- grib_handle * grib_handle_new_from_file (grib_context *c, FILE *f, int *error)

  Create a handle from a file resource.

- grib_handle * grib_handle_new_from_message (grib_context *c, void *data, size_t data_len)

  Create a handle from a user message in memory.

- grib_handle * grib_handle_new_from_multi_message (grib_context *c, void **data, size_t *data_len, int *error)

  Create a handle from a user message in memory.

- grib_handle * grib_handle_new_from_message_copy (grib_context *c, const void *data, size_t data_len)

  Create a handle from a user message.

- grib_handle * grib_handle_new_from_template (grib_context *c, const char *res_name)

  Create a handle from a read_only template resource.

- grib_handle * grib_handle_new_from_samples (grib_context *c, const char *res_name)

  Create a handle from a message contained in a samples directory.

- 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.

- int grib_handle_delete (grib_handle *h)

  Frees a handle, also frees the message if it is not a user message.

- grib_multi_handle * grib_multi_handle_new (grib_context *c)

  Create an empty multi field handle.

- int grib_multi_handle_append (grib_handle *h, int start_section, grib_multi_handle *mh)

  Append the sections starting with start_section of the message pointed by h at the end of the multi field handle mh.

- int grib_multi_handle_delete (grib_multi_handle *mh)

  Delete multi field handle.

- int grib_multi_handle_write (grib_multi_handle *mh, FILE *f)

  Write a multi field handle in a file.

5.3.1 Detailed Description

The grib_handle is the structure giving access to parsed grib values by keys.

5.3.2 Typedef Documentation

5.3.2.1 typedef struct grib_handle grib_handle

Grib handle, structure giving access to parsed grib values by keys

Examples:

get.c, iterator.c, keys_iterator.c, multi.c, multi_write.c, nearest.c, precision.c, print_data.c, and set.c.

5.3.2.2 typedef struct grib_multi_handle grib_multi_handle

Grib multi field handle, structure used to build multi fields messages.

Examples:

multi_write.c.

5.3.3 Function Documentation

5.3.3.1 int grib_count_in_file (grib_context * *c*, FILE * *f*, int * *n*)

Counts the messages contained in a file resource.

Parameters:

c : the context from wich the handle will be created (NULL for default context)
f : the file resource
n : the number of messages in the file

Returns:

0 if OK, integer value on error

Examples:

count_messages.f90.

5.3.3.2 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.

Parameters:

h : The handle to be cloned

Returns:

the new handle, NULL if the message is invalid or a problem is encountered

5.3.3.3 int grib_handle_delete (grib_handle * *h*)

Frees a handle, also frees the message if it is not a user message.

See also:

grib_handle_new_from_message

Parameters:

h : The handle to be deleted

Returns:

0 if OK, integer value on error

Examples:

get.c, iterator.c, multi.c, multi_write.c, nearest.c, precision.c, print_data.c, and set.c.

5.3.3.4 grib_handle* grib_handle_new_from_file (grib_context * *c*, FILE * *f*, int * *error*)

Create a handle from a file resource.

The file is read until a message is found. The message is then copied. Remember always to delete the handle when it is not needed any more to avoid memory leaks.

Parameters:

c : the context from wich the handle will be created (NULL for default context)

f : the file resource

error : error code set if the returned handle is NULL and the end of file is not reached

Returns:

the new handle, NULL if the resource is invalid or a problem is encountered

Examples:

get.c, iterator.c, keys_iterator.c, multi.c, multi_write.c, precision.c, print_data.c, and set.c.

5.3.3.5 grib_handle* grib_handle_new_from_message (grib_context * *c*, void * *data*, size_t *data_len*)

Create a handle from a user message in memory.

The message will not be freed at the end. The message will be copied as soon as a modification is needed.

Parameters:

c : the context from which the handle will be created (NULL for default context)

data : the actual message

data_len : the length of the message in number of bytes

Returns:

the new handle, NULL if the message is invalid or a problem is encountered

5.3.3.6 grib_handle* grib_handle_new_from_message_copy (grib_context * *c*, const void * *data*, size_t *data_len*)

Create a handle from a user message.

The message is copied and will be freed with the handle

Parameters:

c : the context from wich the handle will be created (NULL for default context)

data : the actual message

data_len : the length of the message in number of bytes

Returns:

the new handle, NULL if the message is invalid or a problem is encountered

5.3.3.7 grib_handle* grib_handle_new_from_multi_message (grib_context * *c*, void ** *data*, size_t * *data_len*, int * *error*)

Create a handle from a user message in memory.

The message will not be freed at the end. The message will be copied as soon as a modification is needed. This function works also with multi field messages.

Parameters:

c : the context from which the handle will be created (NULL for default context)

data : the actual message

data_len : the length of the message in number of bytes

error : error code

Returns:

the new handle, NULL if the message is invalid or a problem is encountered

5.3.3.8 grib_handle* grib_handle_new_from_samples (grib_context * *c*, const char * *res_name*)

Create a handle from a message contained in a samples directory.

The message is copied at the creation of the handle

Parameters:

c : the context from wich the handle will be created (NULL for default context)

res_name : the resource name

Returns:

the new handle, NULL if the resource is invalid or a problem is encountered

5.3.3.9 grib_handle* grib_handle_new_from_template (grib_context * *c*, const char * *res_name*)

Create a handle from a read_only template resource.

The message is copied at the creation of the handle

Parameters:

c : the context from wich the handle will be created (NULL for default context)

res_name : the resource name

Returns:

the new handle, NULL if the resource is invalid or a problem is encountered

5.3.3.10 int grib_multi_handle_append (grib_handle * *h*, int *start_section*, grib_multi_handle * *mh*)

Append the sections starting with start_section of the message pointed by h at the end of the multi field handle mh.

Remember always to delete the multi handle when it is not needed any more to avoid memory leaks.

Parameters:

h : The handle from which the sections are copied.

start_section : section number. Starting from this section all the sections to then end of the message will be copied.

mh : The multi field handle on which the sections are appended.

Returns:

0 if OK, integer value on error

Examples:

multi_write.c.

5.3.3.11 int grib_multi_handle_delete (grib_multi_handle * *mh*)

Delete multi field handle.

Parameters:

mh : The multi field handle to be deleted.

Returns:

0 if OK, integer value on error

Examples:

multi_write.c.

5.3.3.12 grib_multi_handle* grib_multi_handle_new (grib_context * *c*)

Create an empty multi field handle.

Remember always to delete the multi handle when it is not needed any more to avoid memory leaks.

Parameters:

c : the context from wich the handle will be created (NULL for default context)

Examples:

multi_write.c.

5.3.3.13 int grib_multi_handle_write (grib_multi_handle * *mh*, FILE * *f*)

Write a multi field handle in a file.

Remember always to delete the multi handle when it is not needed any more to avoid memory leaks.

Parameters:

mh : The multi field handle to be written.

f : File on which the file handle is written.

Returns:

0 if OK, integer value on error

Examples:

multi_write.c.

5.4 Handling coded messages

Functions

- int grib_get_message (grib_handle *h, const void **message, size_t *message_length)
  getting the message attached to a handle
- int grib_get_message_copy (grib_handle *h, void *message, size_t *message_length)
  getting a copy of the message attached to a handle

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 int grib_get_message (grib_handle * *h*, const void ** *message*, size_t * *message_length*)

getting the message attached to a handle

Parameters:

h : the grib handle to wich the buffer should be gathered

message : the pointer to be set to the handle's data

message_length : at exist, the message size in number of bytes

Returns:

0 if OK, integer value on error

Examples:

precision.c, and set.c.

5.4.2.2 int grib_get_message_copy (grib_handle * *h*, void * *message*, size_t * *message_length*)

getting a copy of the message attached to a handle

Parameters:

h : the grib handle to wich the buffer should be returned

message : the pointer to the data buffer to be filled

message_length : at entry, the size in number of bytes of the allocated empty message. At exist, the actual message length in number of bytes

Returns:

0 if OK, integer value on error

5.5 Iterating on latitude/longitude/values

Functions

- grib_iterator * grib_iterator_new (grib_handle *h, unsigned long flags, int *error)

  Create a new iterator from a handle, using current geometry and values.

- int grib_iterator_next (grib_iterator *i, double *lat, double *lon, double *value)

  Get the next value from an iterator.

- int grib_iterator_previous (grib_iterator *i, double *lat, double *lon, double *value)

  Get the previous value from an iterator.

- int grib_iterator_has_next (grib_iterator *i)

  Test procedure for values in an iterator.

- int grib_iterator_reset (grib_iterator *i)

  Test procedure for values in an iterator.

- int grib_iterator_delete (grib_iterator *i)

  Frees an iterator from memory.

- grib_nearest * grib_nearest_new (grib_handle *h, int *error)

  Create a new nearest from a handle, using current geometry .

- int grib_nearest_find (grib_nearest *nearest, grib_handle *h, double inlat, double inlon, unsigned long flags, double *outlats, double *outlons, double *values, double *distances, int *indexes, size_t *len)

  Find the 4 nearest points of a latitude longitude point.

- int grib_nearest_delete (grib_nearest *nearest)

  Frees an nearest from memory.

- int grib_nearest_find_multiple (grib_handle *h, int is_lsm, double *inlats, double *inlons, long npoints, double *outlats, double *outlons, double *values, double *distances, int *indexes)

  Find the nearest point of a set of points whose latitudes and longitudes are given in the inlats, inlons arrays respectively.

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 int grib_iterator_delete (grib_iterator * *i*)

Frees an iterator from memory.

Parameters:

i : the iterator

Returns:

0 if OK, integer value on error

Examples:

iterator.c, and iterator_fortran.F.

5.5.2.2 int grib_iterator_has_next (grib_iterator * *i*)

Test procedure for values in an iterator.

Parameters:

i : the iterator

Returns:

boolean, 1 if the iterator still nave next values, 0 otherwise

5.5.2.3 grib_iterator* grib_iterator_new (grib_handle * *h*, unsigned long *flags*, int * *error*)

Create a new iterator from a handle, using current geometry and values.

Parameters:

h : the handle from which the iterator will be created
flags : flags for future use.
error : error code

Returns:

the new iterator, NULL if no iterator can be created

Examples:

iterator.c, and iterator_fortran.F.

5.5.2.4 int grib_iterator_next (grib_iterator * *i*, double * *lat*, double * *lon*, double * *value*)

Get the next value from an iterator.

Parameters:

i : the iterator
lat : on output latitude in degree
lon : on output longitude in degree
value : on output value of the point

Returns:

positive value if successful, 0 if no more data are available

Examples:

iterator.c, and iterator_fortran.F.

5.5.2.5 int grib_iterator_previous (grib_iterator * *i*, double * *lat*, double * *lon*, double * *value*)

Get the previous value from an iterator.

Parameters:

i : the iterator

lat : on output latitude in degree

lon : on output longitude in degree

value : on output value of the point*

Returns:

positive value if successful, 0 if no more data are available

5.5.2.6 int grib_iterator_reset (grib_iterator * *i*)

Test procedure for values in an iterator.

Parameters:

i : the iterator

Returns:

0 if OK, integer value on error

5.5.2.7 int grib_nearest_delete (grib_nearest * *nearest*)

Frees an nearest from memory.

Parameters:

nearest : the nearest

Returns:

0 if OK, integer value on error

Examples:

nearest.c.

5.5.2.8 int grib_nearest_find (grib_nearest * *nearest*, grib_handle * *h*, double *inlat*, double *inlon*, unsigned long *flags*, double * *outlats*, double * *outlons*, double * *values*, double * *distances*, int * *indexes*, size_t * *len*)

Find the 4 nearest points of a latitude longitude point.

The flags are provided to speed up the process of searching. If you are sure that the point you are asking for is not changing from a call to another you can use GRIB_NEAREST_SAME_POINT. The same is valid for the grid. Flags can be used together duing an or.

Parameters:

nearest : nearest structure

h : handle from which geography and data values are taken

inlat : latitude of the point to search for

inlon : longitude of the point to search for

flags : GRIB_NEAREST_SAME_POINT, GRIB_NEAREST_SAME_GRID

outlats : returned array of latitudes of the nearest points

outlons : returned array of longitudes of the nearest points

values : returned array of data values of the nearest points

distances : returned array of distances from the nearest points

indexes : returned array of indexes of the nearest points

len : size of the arrays

Returns:

0 if OK, integer value on error

Examples:

nearest.c.

5.5.2.9 int grib_nearest_find_multiple (grib_handle * *h*, int *is_lsm*, double * *inlats*, double * *inlons*, long *npoints*, double * *outlats*, double * *outlons*, double * *values*, double * *distances*, int * *indexes*)

Find the nearest point of a set of points whose latitudes and longitudes are given in the inlats, inlons arrays respectively.

If the flag is_lsm is 1 the nearest land point is returned and the grib passed as handle (h) is considered a land sea mask. The land nearest point is the nearest point with land sea mask value>=0.5. If no nearest land points are found the nearest value is returned. If the flag is_lsm is 0 the nearest point is returned. values, distances, indexes (in the "values" array) for the nearest points (ilons,ilats) are returned.

Parameters:

h : handle from which geography and data values are taken

is_lsm : lsm flag (1-> nearest land, 0-> nearest)

inlats : latitudes of the points to search for

inlons : longitudes of the points to search for

npoints : number of points (size of the inlats,inlons,outlats,outlons,values,distances,indexes arrays)

outlats : returned array of latitudes of the nearest points

outlons : returned array of longitudes of the nearest points

values : returned array of data values of the nearest points

distances : returned array of distances from the nearest points

indexes : returned array of indexes of the nearest points

Returns:

0 if OK, integer value on error

5.5.2.10 grib_nearest* grib_nearest_new (grib_handle * *h*, int * *error*)

Create a new nearest from a handle, using current geometry .

Parameters:

h : the handle from which the iterator will be created

error : error code

Returns:

the new nearest, NULL if no nearest can be created

Examples:

nearest.c.

5.6 Accessing header and data values

Functions

- int grib_get_offset (grib_handle *h, const char *key, size_t *offset)

  Get the number offset of a key, in a message if several keys of the same name are present, the offset of the last one is returned.

- int grib_get_size (grib_handle *h, const char *key, size_t *size)

  Get the number of coded value from a key, if several keys of the same name are present, the total sum is returned.

- int grib_get_long (grib_handle *h, const char *key, long *value)

  Get a long value from a key, if several keys of the same name are present, the last one is returned.

- int grib_get_double (grib_handle *h, const char *key, double *value)

  Get a double value from a key, if several keys of the same name are present, the last one is returned.

- int grib_get_double_element (grib_handle *h, const char *key, int i, double *value)

  Get as double the i-th element of the "key" array.

- int grib_get_double_elements (grib_handle *h, const char *key, int *i, long size, double *value)

  Get as double array the elements of the "key" array whose indexes are listed in the input array i.

- int grib_get_string (grib_handle *h, const char *key, char *mesg, size_t *length)

  Get a string value from a key, if several keys of the same name are present, the last one is returned.

- int grib_get_bytes (grib_handle *h, const char *key, unsigned char *bytes, size_t *length)

  Get raw bytes values from a key.

- int grib_get_double_array (grib_handle *h, const char *key, double *vals, size_t *length)

  Get double array values from a key.

- int grib_get_long_array (grib_handle *h, const char *key, long *vals, size_t *length)

  Get long array values from a key.

- int grib_copy_namespace (grib_handle *dest, const char *name, grib_handle *src)

  Copy the keys belonging to a given namespace from a source handle to a destination handle.

- int grib_set_long (grib_handle *h, const char *key, long val)

  Set a long value from a key.

- int grib_set_double (grib_handle *h, const char *key, double val)

  Set a double value from a key.

- int grib_set_string (grib_handle *h, const char *key, const char *mesg, size_t *length)

  Set a string value from a key.

- int grib_set_bytes (grib_handle *h, const char *key, const unsigned char *bytes, size_t *length)

  Set a bytes array from a key.

- int grib_set_double_array (grib_handle *h, const char *key, const double *vals, size_t length)

  Set a double array from a key.

- int grib_set_long_array (grib_handle *h, const char *key, const long *vals, size_t length)

  Set a long array from a key.

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 int grib_copy_namespace (grib_handle * *dest*, const char * *name*, grib_handle * *src*)

Copy the keys belonging to a given namespace from a source handle to a destination handle.

Parameters:

dest : destination handle

name : namespace

src : source handle

Returns:

0 if OK, integer value on error

5.6.2.2 int grib_get_bytes (grib_handle * *h*, const char * *key*, unsigned char * *bytes*, size_t * *length*)

Get raw bytes values from a key.

If several keys of the same name are present, the last one is returned

See also:

grib_set_bytes

Parameters:

h : the handle to get the data from

key : the key to be searched

bytes : the address of a byte array where the data will be retreived

length : the address of a size_t that contains allocated length of the byte array on input, and that contains the actual length of the byte array on output

Returns:

0 if OK, integer value on error

5.6.2.3 int grib_get_double (grib_handle * *h*, const char * *key*, double * *value*)

Get a double value from a key, if several keys of the same name are present, the last one is returned.

See also:

grib_set_double

Parameters:

h : the handle to get the data from

key : the key to be searched

value : the address of a double where the data will be retreived

Returns:

0 if OK, integer value on error

Examples:

get.c, iterator.c, and print_data.c.

5.6.2.4 int grib_get_double_array (grib_handle * *h*, const char * *key*, double * *vals*, size_t * *length*)

Get double array values from a key.

If several keys of the same name are present, the last one is returned

See also:

grib_set_double_array

Parameters:

h : the handle to get the data from

key : the key to be searched

vals : the address of a double array where the data will be retreived

length : the address of a size_t that contains allocated length of the double array on input, and that contains the actual length of the double array on output

Returns:

0 if OK, integer value on error

Examples:

get.c, precision.c, and print_data.c.

5.6.2.5 int grib_get_double_element (grib_handle * *h*, const char * *key*, int *i*, double * *value*)

Get as double the i-th element of the "key" array.

Parameters:

h : the handle to get the data from

key : the key to be searched

i : zero based index

value : the address of a double where the data will be retreived

Returns:

0 if OK, integer value on error

5.6.2.6 int grib_get_double_elements (grib_handle * *h*, const char * *key*, int * *i*, long *size*, double * *value*)

Get as double array the elements of the "key" array whose indexes are listed in the input array i.

Parameters:

h : the handle to get the data from

key : the key to be searched

i : zero based array of indexes

size : size of the i and value arrays

value : the address of a double where the data will be retreived

Returns:

0 if OK, integer value on error

5.6.2.7 int grib_get_long (grib_handle * *h*, const char * *key*, long * *value*)

Get a long value from a key, if several keys of the same name are present, the last one is returned.

See also:

grib_set_long

Parameters:

h : the handle to get the data from

key : the key to be searched

value : the address of a long where the data will be retreived

Returns:

0 if OK, integer value on error

Examples:

get.c, multi.c, nearest.c, precision.c, and set.c.

5.6.2.8 int grib_get_long_array (grib_handle * *h*, const char * *key*, long * *vals*, size_t * *length*)

Get long array values from a key.

If several keys of the same name are present, the last one is returned

See also:

grib_set_long_array

Parameters:

h : the handle to get the data from

key : the key to be searched

vals : the address of a long array where the data will be retreived

length : the address of a size_t that contains allocated length of the long array on input, and that contains the actual length of the long array on output

Returns:

0 if OK, integer value on error

5.6.2.9 int grib_get_offset (grib_handle * *h*, const char * *key*, size_t * *offset*)

Get the number offset of a key, in a message if several keys of the same name are present, the offset of the last one is returned.

Parameters:

h : the handle to get the offset from

key : the key to be searched

offset : the address of a size_t where the offset will be set

Returns:

0 if OK, integer value on error

5.6.2.10 int grib_get_size (grib_handle * *h*, const char * *key*, size_t * *size*)

Get the number of coded value from a key, if several keys of the same name are present, the total sum is returned.

Parameters:

h : the handle to get the offset from

key : the key to be searched

size : the address of a size_t where the size will be set

Returns:

0 if OK, integer value on error

Examples:

count_messages.f90, get.c, get.f90, get_fortran.F, get_pl.f90, get_pv.f90, precision.c, precision.f90, precision_fortran.F, print_data.c, print_data.f90, print_data_fortran.F, samples.f90, and set_bitmap.f90.

5.6.2.11 int grib_get_string (grib_handle * *h*, const char * *key*, char * *mesg*, size_t * *length*)

Get a string value from a key, if several keys of the same name are present, the last one is returned.

See also:

grib_set_string

Parameters:

h : the handle to get the data from

key : the key to be searched

mesg : the address of a string where the data will be retreived

length : the address of a size_t that contains allocated length of the string on input, and that contains the actual length of the string on output

Returns:

0 if OK, integer value on error

Examples:

keys_iterator.c, keys_iterator_fortran.F, nearest.c, set.c, and set_fortran.F.

5.6.2.12 int grib_set_bytes (grib_handle * *h*, const char * *key*, const unsigned char * *bytes*, size_t * *length*)

Set a bytes array from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_bytes

Parameters:

h : the handle to set the data to

key : the key to be searched

bytes : the address of a byte array where the data will be read

length : the address of a size_t that contains the length of the byte array on input, and that contains the actual packed length of the byte array on output

Returns:

0 if OK, integer value on error

5.6.2.13 int grib_set_double (grib_handle * *h*, const char * *key*, double *val*)

Set a double value from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_double

Parameters:

h : the handle to set the data to

key : the key to be searched

val : a double where the data will be read

Returns:

0 if OK, integer value on error

5.6.2.14 int grib_set_double_array (grib_handle * *h*, const char * *key*, const double * *vals*, size_t *length*)

Set a double array from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_double_array

Parameters:

h : the handle to set the data to

key : the key to be searched

vals : the address of a double array where the data will be read

length : a size_t that contains the length of the byte array on input

Returns:

0 if OK, integer value on error

5.6.2.15 int grib_set_long (grib_handle * *h*, const char * *key*, long *val*)

Set a long value from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_long

Parameters:

h : the handle to set the data to

key : the key to be searched

val : a long where the data will be read

Returns:

0 if OK, integer value on error

Examples:

multi_write.c, precision.c, and set.c.

5.6.2.16 int grib_set_long_array (grib_handle * *h*, const char * *key*, const long * *vals*, size_t *length*)

Set a long array from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_long_array

Parameters:

h : the handle to set the data to

key : the key to be searched

vals : the address of a long array where the data will be read

length : a size_t that contains the length of the long array on input

Returns:

0 if OK, integer value on error

5.6.2.17 int grib_set_string (grib_handle * *h*, const char * *key*, const char * *mesg*, size_t * *length*)

Set a string value from a key.

If several keys of the same name are present, the last one is set

See also:

grib_get_string

Parameters:

h : the handle to set the data to

key : the key to be searched

mesg : the address of a string where the data will be read

length : the address of a size_t that contains the length of the string on input, and that contains the actual packed length of the string on output

Returns:

0 if OK, integer value on error

5.7 The context object

Typedefs

- typedef void(∗ grib_free_proc)(const grib_context ∗c, void ∗data)

  Grib free procedure, format of a procedure referenced in the context that is used to free memory.

- typedef void ∗(∗ grib_malloc_proc)(const grib_context ∗c, size_t length)

  Grib malloc procedure, format of a procedure referenced in the context that is used to allocate memory.

- typedef void ∗(∗ grib_realloc_proc)(const grib_context ∗c, void ∗data, size_t length)

  Grib realloc procedure, format of a procedure referenced in the context that is used to reallocate memory.

- typedef void(∗ grib_log_proc)(const grib_context ∗c, int level, const char ∗mesg)

  Grib loc proc, format of a procedure referenced in the context that is used to log internal messages.

- typedef void(∗ grib_print_proc)(const grib_context ∗c, void ∗descriptor, const char ∗mesg)

  Grib print proc, format of a procedure referenced in the context that is used to print external messages.

- typedef size_t(∗ grib_data_read_proc)(const grib_context ∗c, void ∗ptr, size_t size, void ∗stream)

  Grib data read proc, format of a procedure referenced in the context that is used to read from a stream in a resource.

- typedef size_t(∗ grib_data_write_proc)(const grib_context ∗c, const void ∗ptr, size_t size, void ∗stream)

  Grib data read write, format of a procedure referenced in the context that is used to write to a stream from a resource.

- typedef off_t(∗ grib_data_tell_proc)(const grib_context ∗c, void ∗stream)

  Grib data tell, format of a procedure referenced in the context that is used to tell the current position in a stream.

- typedef off_t(∗ grib_data_seek_proc)(const grib_context ∗c, off_t offset, int whence, void ∗stream)

  Grib data seek, format of a procedure referenced in the context that is used to seek the current position in a stream.

- typedef int(∗ grib_data_eof_proc)(const grib_context ∗c, void ∗stream)

  Grib data eof, format of a procedure referenced in the context that is used to test end of file.

Functions

- grib_context ∗ grib_get_context (grib_handle ∗h)

  Retreive the context from a handle.

- grib_context ∗ grib_context_get_default (void)

  Get the static default context.

- grib_context ∗ grib_context_new (grib_context ∗c)

Create and allocate a new context from a parent context.

- void grib_context_delete (grib_context *c)

  Frees the cached definition files of the context.

- void grib_gts_header_on (grib_context *c)

  Set the gts header mode on.

- void grib_gts_header_off (grib_context *c)

  Set the gts header mode off.

- void grib_gribex_mode_on (grib_context *c)

  Set the gribex mode on.

- void grib_gribex_mode_off (grib_context *c)

  Set the gribex mode off.

- void grib_context_set_user_data (grib_context *c, void *udata)

  Sets user data in a context.

- void * grib_context_get_user_data (grib_context *c)

  get userData from a context

- void grib_context_set_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree, grib_realloc_proc gribrealloc)

  Sets memory procedures of the context.

- void grib_context_set_persistent_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree)

  Sets memory procedures of the context for persistent data.

- void grib_context_set_buffer_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree, grib_realloc_proc gribrealloc)

  Sets memory procedures of the context for large buffers.

- void grib_context_set_path (grib_context *c, const char *path)

  Sets the context search path for definition files.

- void grib_context_set_dump_mode (grib_context *c, int mode)

  Sets context dump mode.

- void grib_context_set_print_proc (grib_context *c, grib_print_proc printp)

  Sets the context printing procedure used for user interaction.

- void grib_context_set_logging_proc (grib_context *c, grib_log_proc logp)

  Sets the context logging procedure used for system (warning, errors, infos .

- void grib_multi_support_on (grib_context *c)

  Turn on support for multiple fields in single grib messages.

- void grib_multi_support_off (grib_context *c)

  Turn off support for multiple fields in single grib messages.

5.7.1 Detailed Description

The context is a long life configuration object of the grib_api. It is used to define special allocation and free routines or to set special grib_api behaviours and variables.

5.7.2 Typedef Documentation

5.7.2.1 typedef int(* grib_data_eof_proc)(const grib_context *c, void *stream)

Grib data eof, format of a procedure referenced in the context that is used to test end of file.

Parameters:

c : the context where the tell will apply

****stream*** : the stream

Returns:

the position in the stream

5.7.2.2 typedef size_t(* grib_data_read_proc)(const grib_context *c, void *ptr, size_t size, void *stream)

Grib data read proc, format of a procedure referenced in the context that is used to read from a stream in a resource.

Parameters:

c : the context where the read will apply

****ptr*** : the resource

size : size to read

****stream*** : the stream

Returns:

size read

5.7.2.3 typedef off_t(* grib_data_seek_proc)(const grib_context *c, off_t offset, int whence, void *stream)

Grib data seek, format of a procedure referenced in the context that is used to seek the current position in a stream.

Parameters:

c : the context where the tell will apply

offset : the offset to seek to

whence : If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

∗***stream*** : the stream

Returns:

0 if OK, integer value on error

5.7.2.4 typedef off_t(∗ grib_data_tell_proc)(const grib_context ∗c, void ∗stream)

Grib data tell, format of a procedure referenced in the context that is used to tell the current position in a stream.

Parameters:

c : the context where the tell will apply

∗***stream*** : the stream

Returns:

the position in the stream

5.7.2.5 typedef size_t(∗ grib_data_write_proc)(const grib_context ∗c, const void ∗ptr, size_t size, void ∗stream)

Grib data read write, format of a procedure referenced in the context that is used to write to a stream from a resource.

Parameters:

c : the context where the write will apply

∗***ptr*** : the resource

size : size to read

∗***stream*** : the stream

Returns:

size written

5.7.2.6 typedef void(∗ grib_free_proc)(const grib_context ∗c, void ∗data)

Grib free procedure, format of a procedure referenced in the context that is used to free memory.

Parameters:

c : the context where the memory freeing will apply

data : pointer to the data to be freed must match

See also:

grib_malloc_proc

5.7.2.7 typedef void(∗ grib_log_proc)(const grib_context ∗c, int level, const char ∗mesg)

Grib loc proc, format of a procedure referenced in the context that is used to log internal messages.

Parameters:

c : the context where the logging will apply
level : the log level, as defined in log modes
mesg : the message to be logged

5.7.2.8 typedef void∗(∗ grib_malloc_proc)(const grib_context ∗c, size_t length)

Grib malloc procedure, format of a procedure referenced in the context that is used to allocate memory.

Parameters:

c : the context where the memory allocation will apply
length : length to be allocated in number of bytes

Returns:

a pointer to the alocated memory, NULL if no memory can be allocated must match

See also:

grib_free_proc

5.7.2.9 typedef void(∗ grib_print_proc)(const grib_context ∗c, void ∗descriptor, const char ∗mesg)

Grib print proc, format of a procedure referenced in the context that is used to print external messages.

Parameters:

c : the context where the logging will apply
descriptor : the structure to be printed on, must match the implementation
mesg : the message to be printed

5.7.2.10 typedef void∗(∗ grib_realloc_proc)(const grib_context ∗c, void ∗data, size_t length)

Grib realloc procedure, format of a procedure referenced in the context that is used to reallocate memory.

Parameters:

c : the context where the memory allocation will apply
data : pointer to the data to be reallocated
length : length to be allocated in number of bytes

Returns:

a pointer to the alocated memory

5.7.3 Function Documentation

5.7.3.1 void grib_context_delete (grib_context * *c*)

Frees the cached definition files of the context.

Parameters:

c : the context to be deleted

5.7.3.2 grib_context* grib_context_get_default (void)

Get the static default context.

Returns:

the default context, NULL it the context is not available

5.7.3.3 void* grib_context_get_user_data (grib_context * *c*)

get userData from a context

Parameters:

c : the context from which the user data will be retreived

Returns:

the user data referenced in the context

5.7.3.4 grib_context* grib_context_new (grib_context * *c*)

Create and allocate a new context from a parent context.

Parameters:

c : the context to be cloned, NULL for default context

Returns:

the new and empty context, NULL if error

5.7.3.5 void grib_context_set_buffer_memory_proc (grib_context * *c*, grib_malloc_proc *griballoc*, grib_free_proc *gribfree*, grib_realloc_proc *gribrealloc*)

Sets memory procedures of the context for large buffers.

Parameters:

c : the context to be modified

griballoc : the memory allocation procedure to be set

See also:

grib_malloc_proc

Parameters:

gribfree : the memory freeing procedure to be set

See also:

grib_free_proc

5.7.3.6 void grib_context_set_dump_mode (grib_context * *c*, int *mode*)

Sets context dump mode.

Parameters:

c : the context to be modified
mode : the log mode to be set

5.7.3.7 void grib_context_set_logging_proc (grib_context * *c*, grib_log_proc *logp*)

Sets the context logging procedure used for system (warning, errors, infos .

..) messages

Parameters:

c : the context to be modified
logp : the logging procedure to be set

See also:

grib_log_proc

5.7.3.8 void grib_context_set_memory_proc (grib_context * *c*, grib_malloc_proc *griballoc*, grib_free_proc *gribfree*, grib_realloc_proc *gribrealloc*)

Sets memory procedures of the context.

Parameters:

c : the context to be modified
griballoc : the memory allocation procedure to be set

See also:

grib_malloc_proc

Parameters:

gribfree : the memory freeing procedure to be set

See also:

grib_free_proc

5.7.3.9 void grib_context_set_path (grib_context * *c*, const char * *path*)

Sets the context search path for definition files.

Parameters:

c : the context to be modified

path : the search path to be set

5.7.3.10 void grib_context_set_persistent_memory_proc (grib_context * *c*, grib_malloc_proc *griballoc*, grib_free_proc *gribfree*)

Sets memory procedures of the context for persistent data.

Parameters:

c : the context to be modified

griballoc : the memory allocation procedure to be set

See also:

grib_malloc_proc

Parameters:

gribfree : the memory freeing procedure to be set

See also:

grib_free_proc

5.7.3.11 void grib_context_set_print_proc (grib_context * *c*, grib_print_proc *printp*)

Sets the context printing procedure used for user interaction.

Parameters:

c : the context to be modified

printp : the printing procedure to be set

See also:

grib_print_proc

5.7.3.12 void grib_context_set_user_data (grib_context * *c*, void * *udata*)

Sets user data in a context.

Parameters:

c : the context to be modified

udata : the user data to set

5.7.3.13 grib_context* grib_get_context (grib_handle * *h*)

Retreive the context from a handle.

Parameters:

h : the handle used to retreive the context from

Returns:

The handle's context, NULL it the handle is invalid

5.7.3.14 void grib_gribex_mode_off (grib_context * *c*)

Set the gribex mode off.

Grib files won't be always compatible with gribex.

Parameters:

c : the context to be deleted

5.7.3.15 void grib_gribex_mode_on (grib_context * *c*)

Set the gribex mode on.

Grib files will be compatible with gribex.

Parameters:

c : the context to be deleted

5.7.3.16 void grib_gts_header_off (grib_context * *c*)

Set the gts header mode off.

The GTS headers will be deleted.

Parameters:

c : the context to be deleted

5.7.3.17 void grib_gts_header_on (grib_context * *c*)

Set the gts header mode on.

The GTS headers will be preserved.

Parameters:

c : the context to be deleted

5.7.3.18 void grib_multi_support_off (grib_context * c)

Turn off support for multiple fields in single grib messages.

Parameters:

c : the context to be modified

Examples:

multi.f90, and multi_fortran.F.

5.7.3.19 void grib_multi_support_on (grib_context * c)

Turn on support for multiple fields in single grib messages.

Parameters:

c : the context to be modified

Examples:

multi.c, multi.f90, and multi_fortran.F.

5.8 Iterating on keys names

Defines

- #define GRIB_KEYS_ITERATOR_ALL_KEYS 0
- #define GRIB_KEYS_ITERATOR_SKIP_READ_ONLY (1<<0)
- #define GRIB_KEYS_ITERATOR_SKIP_OPTIONAL (1<<1)
- #define GRIB_KEYS_ITERATOR_SKIP_EDITION_SPECIFIC (1<<2)
- #define GRIB_KEYS_ITERATOR_SKIP_CODED (1<<3)
- #define GRIB_KEYS_ITERATOR_SKIP_COMPUTED (1<<4)
- #define GRIB_KEYS_ITERATOR_SKIP_DUPLICATES (1<<5)
- #define GRIB_KEYS_ITERATOR_SKIP_FUNCTION (1<<6)

Typedefs

- typedef struct grib_keys_iterator grib_keys_iterator

Functions

- grib_keys_iterator * grib_keys_iterator_new (grib_handle *h, unsigned long filter_flags, char *name_space)
- int grib_keys_iterator_next (grib_keys_iterator *kiter)
- const char * grib_keys_iterator_get_name (grib_keys_iterator *kiter)
- int grib_keys_iterator_delete (grib_keys_iterator *kiter)
- int grib_keys_iterator_rewind (grib_keys_iterator *kiter)
- int **grib_keys_iterator_set_flags** (grib_keys_iterator *kiter, unsigned long flags)

5.8.1 Detailed Description

The keys iterator is designed to get the key names defined in a message. Key names on which the iteration is carried out can be filtered through their attributes or by the namespace they belong to.

5.8.2 Define Documentation

5.8.2.1 #define GRIB_KEYS_ITERATOR_ALL_KEYS 0

Iteration is carried out on all the keys available in the message

See also:

grib_keys_iterator_new

Examples:

keys_iterator.c.

5.8.2.2 #define GRIB_KEYS_ITERATOR_SKIP_CODED (1<<3)

coded keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.3 #define GRIB_KEYS_ITERATOR_SKIP_COMPUTED (1<<4)

computed keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.4 #define GRIB_KEYS_ITERATOR_SKIP_DUPLICATES (1<<5)

duplicates of a key are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.5 #define GRIB_KEYS_ITERATOR_SKIP_EDITION_SPECIFIC (1<<2)

edition specific keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.6 #define GRIB_KEYS_ITERATOR_SKIP_FUNCTION (1<<6)

function keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.7 #define GRIB_KEYS_ITERATOR_SKIP_OPTIONAL (1<<1)

optional keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.2.8 #define GRIB_KEYS_ITERATOR_SKIP_READ_ONLY (1<<0)

read only keys are skipped by keys iterator.

See also:

grib_keys_iterator_new

5.8.3 Typedef Documentation

5.8.3.1 typedef struct grib_keys_iterator grib_keys_iterator

Grib keys iterator. Iterator over keys.

Examples:

keys_iterator.c, and keys_iterator_fortran.F.

5.8.4 Function Documentation

5.8.4.1 int grib_keys_iterator_delete (grib_keys_iterator * *kiter*)

Delete the iterator.

Parameters:

kiter : valid grib_keys_iterator

Returns:

0 if OK, integer value on error

Examples:

keys_iterator.c, keys_iterator.f90, and keys_iterator_fortran.F.

5.8.4.2 const char* grib_keys_iterator_get_name (grib_keys_iterator * *kiter*)

get the key name from the iterator

Parameters:

kiter : valid grib_keys_iterator

Returns:

key name

Examples:

keys_iterator.c, keys_iterator.f90, and keys_iterator_fortran.F.

5.8.4.3 grib_keys_iterator* grib_keys_iterator_new (grib_handle * *h*, unsigned long *filter_flags*, char * *name_space*)

Create a new iterator from a valid and initialized handle.

Parameters:

h : the handle whose keys you want to iterate

filter_flags : flags to filter out some of the keys through their attributes

name_space : if not null the iteration is carried out only on keys belongin to the namespace passed. (NULL for all the keys)

Returns:

keys iterator ready to iterate through keys according to filter_flags and namespace

Examples:

keys_iterator.c, and keys_iterator.f90.

5.8.4.4 int grib_keys_iterator_next (grib_keys_iterator * *kiter*)

Step to the next iterator.

Parameters:

kiter : valid grib_keys_iterator

Returns:

1 if next iterator exitsts, 0 if no more elements to iterate on

Examples:

keys_iterator.c, keys_iterator.f90, and keys_iterator_fortran.F.

5.8.4.5 int grib_keys_iterator_rewind (grib_keys_iterator * *kiter*)

Rewind the iterator.

Parameters:

kiter : valid grib_keys_iterator

Returns:

0 if OK, integer value on error

Chapter 6

grib_api.h File Documentation

6.1 grib_api.h File Reference

Defines

- #define **GRIB_API_VERSION** (GRIB_API_MAJOR_VERSION∗10000+GRIB_API_MINOR_-VERSION∗100+GRIB_API_REVISION_VERSION)
- #define **GRIB_LOG_INFO** 0
- #define **GRIB_LOG_WARNING** 1
- #define **GRIB_LOG_ERROR** 2
- #define **GRIB_LOG_FATAL** 3
- #define **GRIB_LOG_DEBUG** 4
- #define **GRIB_TYPE_UNDEFINED** 0
- #define **GRIB_TYPE_LONG** 1
- #define **GRIB_TYPE_DOUBLE** 2
- #define **GRIB_TYPE_STRING** 3
- #define **GRIB_TYPE_BYTES** 4
- #define **GRIB_TYPE_SECTION** 5
- #define **GRIB_TYPE_LABEL** 6
- #define **GRIB_TYPE_MISSING** 7
- #define **GRIB_MISSING_LONG** 0xffffffff
- #define **GRIB_MISSING_DOUBLE** -1e+100
- #define **GRIB_DUMP_FLAG_READ_ONLY** (1<<0)
- #define **GRIB_DUMP_FLAG_DUMP_OK** (1<<1)
- #define **GRIB_DUMP_FLAG_VALUES** (1<<2)
- #define **GRIB_DUMP_FLAG_CODED** (1<<3)
- #define **GRIB_DUMP_FLAG_OCTECT** (1<<4)
- #define **GRIB_DUMP_FLAG_ALIASES** (1<<5)
- #define **GRIB_DUMP_FLAG_TYPE** (1<<6)
- #define **GRIB_DUMP_FLAG_HEXADECIMAL** (1<<7)
- #define **GRIB_DUMP_FLAG_NO_DATA** (1<<8)
- #define **GRIB_DUMP_FLAG_ALL_DATA** (1<<9)
- #define **GRIB_NEAREST_SAME_GRID** (1<<0)

- #define **GRIB_NEAREST_SAME_DATA** (1<<1)
- #define **GRIB_NEAREST_SAME_POINT** (1<<2)
- #define GRIB_KEYS_ITERATOR_ALL_KEYS 0
- #define GRIB_KEYS_ITERATOR_SKIP_READ_ONLY (1<<0)
- #define GRIB_KEYS_ITERATOR_SKIP_OPTIONAL (1<<1)
- #define GRIB_KEYS_ITERATOR_SKIP_EDITION_SPECIFIC (1<<2)
- #define GRIB_KEYS_ITERATOR_SKIP_CODED (1<<3)
- #define GRIB_KEYS_ITERATOR_SKIP_COMPUTED (1<<4)
- #define GRIB_KEYS_ITERATOR_SKIP_DUPLICATES (1<<5)
- #define GRIB_KEYS_ITERATOR_SKIP_FUNCTION (1<<6)
- #define **GRIB_CHECK**(a, msg) grib_check(#a,__FILE__,__LINE__,a,msg)
- #define GRIB_SUCCESS 0

  No error.

- #define GRIB_END_OF_FILE -1

  End of ressource reached.

- #define GRIB_INTERNAL_ERROR -2

  Internal error.

- #define GRIB_BUFFER_TOO_SMALL -3

  Passed buffer is too small.

- #define GRIB_NOT_IMPLEMENTED -4

  Function not yet implemented.

- #define GRIB_7777_NOT_FOUND -5

  Missing 7777 at end of message.

- #define GRIB_ARRAY_TOO_SMALL -6

  Passed array is too small.

- #define GRIB_FILE_NOT_FOUND -7

  File not found.

- #define GRIB_CODE_NOT_FOUND_IN_TABLE -8

  Code not found in code table.

- #define GRIB_STRING_TOO_SMALL_FOR_CODE_NAME -9

  Code cannot unpack because of string too small.

- #define GRIB_WRONG_ARRAY_SIZE -10

  Array size mismatch.

- #define GRIB_NOT_FOUND -11

  Key/value not found.

- #define GRIB_IO_PROBLEM -12

  Input output problem.

- #define GRIB_INVALID_MESSAGE -13
  Message invalid.
- #define GRIB_DECODING_ERROR -14
  Decoding invalid.
- #define GRIB_ENCODING_ERROR -15
  Encoding invalid.
- #define GRIB_NO_MORE_IN_SET -16
  Code cannot unpack because of string too small.
- #define GRIB_GEOCALCULUS_PROBLEM -17
  Problem with calculation of geographic attributes.
- #define GRIB_OUT_OF_MEMORY -18
  Out of memory.
- #define GRIB_READ_ONLY -19
  Value is read only.
- #define GRIB_INVALID_ARGUMENT -20
  Invalid argument.
- #define GRIB_NULL_HANDLE -21
  Null handle.
- #define GRIB_INVALID_SECTION_NUMBER -22
  Invalid section number.
- #define GRIB_VALUE_CANNOT_BE_MISSING -23
  Value cannot be missing.
- #define GRIB_WRONG_LENGTH -24
  Wrong message length.
- #define GRIB_INVALID_TYPE -25
  Invalid key type.
- #define GRIB_WRONG_STEP -26
  Unable to set step.
- #define GRIB_WRONG_STEP_UNIT -27
  Wrong units for step (step must be integer).
- #define GRIB_INVALID_FILE -28
  Invalid file id.
- #define GRIB_INVALID_GRIB -29
  Invalid grib id.

- #define GRIB_INVALID_INDEX -30
  Invalid index id.
- #define GRIB_INVALID_ITERATOR -31
  Invalid iterator id.
- #define GRIB_INVALID_KEYS_ITERATOR -32
  Invalid keys iterator id.
- #define GRIB_INVALID_NEAREST -33
  Invalid nearest id.
- #define GRIB_INVALID_ORDERBY -34
  Invalid order by.
- #define GRIB_MISSING_KEY -35
  Missing a key from the fieldset.
- #define GRIB_OUT_OF_AREA -36
  The point is out of the grid area.
- #define GRIB_CONCEPT_NO_MATCH -37
  Concept no match.
- #define GRIB_NO_DEFINITIONS -38
  Definitions files not found.
- #define GRIB_WRONG_TYPE -39
  Wrong type while packing.
- #define GRIB_END -40
  End of resource.
- #define GRIB_NO_VALUES -41
  Unable to code a field without values.
- #define GRIB_WRONG_GRID -42
  Grid description is wrong or inconsistent.
- #define GRIB_END_OF_INDEX -43
  End of index reached.
- #define GRIB_NULL_INDEX -44
  Null index.
- #define GRIB_PREMATURE_END_OF_FILE -45
  End of ressource reached when reading message.
- #define GRIB_INTERNAL_ARRAY_TOO_SMALL -46

An internal array is too small.

- #define GRIB_MESSAGE_TOO_LARGE -47

  Message is too large for the current architecture.

- #define GRIB_CONSTANT_FIELD -48

  Constant field.

- #define GRIB_SWITCH_NO_MATCH -49

  Switch unable to find a matching case.

Typedefs

- typedef struct grib_key_value_list **grib_key_value_list**
- typedef struct grib_values **grib_values**
- typedef struct grib_handle grib_handle
- typedef struct grib_multi_handle grib_multi_handle
- typedef struct grib_context grib_context
- typedef struct grib_iterator grib_iterator
- typedef struct grib_nearest grib_nearest
- typedef struct grib_keys_iterator grib_keys_iterator
- typedef struct grib_fieldset **grib_fieldset**
- typedef struct grib_order_by **grib_order_by**
- typedef struct grib_where **grib_where**
- typedef struct grib_darray **grib_darray**
- typedef struct grib_iarray **grib_iarray**
- typedef struct grib_index grib_index
- typedef void(∗ grib_free_proc)(const grib_context ∗c, void ∗data)

  Grib free procedure, format of a procedure referenced in the context that is used to free memory.

- typedef void ∗(∗ grib_malloc_proc)(const grib_context ∗c, size_t length)

  Grib malloc procedure, format of a procedure referenced in the context that is used to allocate memory.

- typedef void ∗(∗ grib_realloc_proc)(const grib_context ∗c, void ∗data, size_t length)

  Grib realloc procedure, format of a procedure referenced in the context that is used to reallocate memory.

- typedef void(∗ grib_log_proc)(const grib_context ∗c, int level, const char ∗mesg)

  Grib loc proc, format of a procedure referenced in the context that is used to log internal messages.

- typedef void(∗ grib_print_proc)(const grib_context ∗c, void ∗descriptor, const char ∗mesg)

  Grib print proc, format of a procedure referenced in the context that is used to print external messages.

- typedef size_t(∗ grib_data_read_proc)(const grib_context ∗c, void ∗ptr, size_t size, void ∗stream)

  Grib data read proc, format of a procedure referenced in the context that is used to read from a stream in a resource.

- typedef size_t(∗ grib_data_write_proc)(const grib_context ∗c, const void ∗ptr, size_t size, void ∗stream)

Grib data read write, format of a procedure referenced in the context that is used to write to a stream from a resource.

- typedef off_t(∗ grib_data_tell_proc)(const grib_context ∗c, void ∗stream)

  Grib data tell, format of a procedure referenced in the context that is used to tell the current position in a stream.

- typedef off_t(∗ grib_data_seek_proc)(const grib_context ∗c, off_t offset, int whence, void ∗stream)

  Grib data seek, format of a procedure referenced in the context that is used to seek the current position in a stream.

- typedef int(∗ grib_data_eof_proc)(const grib_context ∗c, void ∗stream)

  Grib data eof, format of a procedure referenced in the context that is used to test end of file.

Functions

- grib_fieldset ∗ **grib_fieldset_new_from_files** (grib_context ∗c, char ∗filenames[ ], int nfiles, char ∗∗keys, int nkeys, char ∗where_string, char ∗order_by_string, int ∗err)
- void **grib_fieldset_delete** (grib_fieldset ∗set)
- void **grib_fieldset_rewind** (grib_fieldset ∗set)
- int **grib_fieldset_apply_order_by** (grib_fieldset ∗set, const char ∗order_by_string)
- grib_handle ∗ **grib_fieldset_next_handle** (grib_fieldset ∗set, int ∗err)
- int **grib_fieldset_count** (grib_fieldset ∗set)
- grib_index ∗ grib_index_new_from_file (grib_context ∗c, char ∗filename, const char ∗keys, int ∗err)

  Create a new index form a file.

- 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_long (grib_index ∗index, const char ∗key, long ∗values, size_t ∗size)

  Get the 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.

- int grib_index_get_string (grib_index ∗index, const char ∗key, char ∗∗values, size_t ∗size)

  Get the distinct values of the key in argument contained in the index.

- int grib_index_select_long (grib_index ∗index, const char ∗key, long value)

  Select the message subset with key==value.

- int grib_index_select_double (grib_index ∗index, const char ∗key, double value)

  Select the message subset with key==value.

- int grib_index_select_string (grib_index ∗index, const char ∗key, char ∗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.

- void grib_index_delete (grib_index *index)

  Delete the index.

- int grib_count_in_file (grib_context *c, FILE *f, int *n)

  Counts the messages contained in a file resource.

- grib_handle * grib_handle_new_from_file (grib_context *c, FILE *f, int *error)

  Create a handle from a file resource.

- grib_handle * grib_handle_new_from_message (grib_context *c, void *data, size_t data_len)

  Create a handle from a user message in memory.

- grib_handle * grib_handle_new_from_multi_message (grib_context *c, void **data, size_t *data_len, int *error)

  Create a handle from a user message in memory.

- grib_handle * grib_handle_new_from_message_copy (grib_context *c, const void *data, size_t data_len)

  Create a handle from a user message.

- grib_handle * grib_handle_new_from_template (grib_context *c, const char *res_name)

  Create a handle from a read_only template resource.

- grib_handle * grib_handle_new_from_samples (grib_context *c, const char *res_name)

  Create a handle from a message contained in a samples directory.

- 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.

- int grib_handle_delete (grib_handle *h)

  Frees a handle, also frees the message if it is not a user message.

- grib_multi_handle * grib_multi_handle_new (grib_context *c)

  Create an empty multi field handle.

- int grib_multi_handle_append (grib_handle *h, int start_section, grib_multi_handle *mh)

  Append the sections starting with start_section of the message pointed by h at the end of the multi field handle mh.

- int grib_multi_handle_delete (grib_multi_handle *mh)

  Delete multi field handle.

- int grib_multi_handle_write (grib_multi_handle *mh, FILE *f)

  Write a multi field handle in a file.

- int grib_get_message (grib_handle *h, const void **message, size_t *message_length)

  getting the message attached to a handle

- int grib_get_message_copy (grib_handle *h, void *message, size_t *message_length)

  getting a copy of the message attached to a handle

- grib_iterator * grib_iterator_new (grib_handle *h, unsigned long flags, int *error)

  Create a new iterator from a handle, using current geometry and values.

- int grib_iterator_next (grib_iterator *i, double *lat, double *lon, double *value)

  Get the next value from an iterator.

- int grib_iterator_previous (grib_iterator *i, double *lat, double *lon, double *value)

  Get the previous value from an iterator.

- int grib_iterator_has_next (grib_iterator *i)

  Test procedure for values in an iterator.

- int grib_iterator_reset (grib_iterator *i)

  Test procedure for values in an iterator.

- int grib_iterator_delete (grib_iterator *i)

  Frees an iterator from memory.

- grib_nearest * grib_nearest_new (grib_handle *h, int *error)

  Create a new nearest from a handle, using current geometry .

- int grib_nearest_find (grib_nearest *nearest, grib_handle *h, double inlat, double inlon, unsigned long flags, double *outlats, double *outlons, double *values, double *distances, int *indexes, size_t *len)

  Find the 4 nearest points of a latitude longitude point.

- int grib_nearest_delete (grib_nearest *nearest)

  Frees an nearest from memory.

- int grib_nearest_find_multiple (grib_handle *h, int is_lsm, double *inlats, double *inlons, long npoints, double *outlats, double *outlons, double *values, double *distances, int *indexes)

  Find the nearest point of a set of points whose latitudes and longitudes are given in the inlats, inlons arrays respectively.

- int grib_get_offset (grib_handle *h, const char *key, size_t *offset)

  Get the number offset of a key, in a message if several keys of the same name are present, the offset of the last one is returned.

- int grib_get_size (grib_handle *h, const char *key, size_t *size)

  Get the number of coded value from a key, if several keys of the same name are present, the total sum is returned.

- int grib_get_long (grib_handle *h, const char *key, long *value)

  Get a long value from a key, if several keys of the same name are present, the last one is returned.

- int grib_get_double (grib_handle *h, const char *key, double *value)

Get a double value from a key, if several keys of the same name are present, the last one is returned.

- int grib_get_double_element (grib_handle *h, const char *key, int i, double *value)
  Get as double the i-th element of the "key" array.
- int grib_get_double_elements (grib_handle *h, const char *key, int *i, long size, double *value)
  Get as double array the elements of the "key" array whose indexes are listed in the input array i.
- int grib_get_string (grib_handle *h, const char *key, char *mesg, size_t *length)
  Get a string value from a key, if several keys of the same name are present, the last one is returned.
- int grib_get_bytes (grib_handle *h, const char *key, unsigned char *bytes, size_t *length)
  Get raw bytes values from a key.
- int grib_get_double_array (grib_handle *h, const char *key, double *vals, size_t *length)
  Get double array values from a key.
- int grib_get_long_array (grib_handle *h, const char *key, long *vals, size_t *length)
  Get long array values from a key.
- int grib_copy_namespace (grib_handle *dest, const char *name, grib_handle *src)
  Copy the keys belonging to a given namespace from a source handle to a destination handle.
- int grib_set_long (grib_handle *h, const char *key, long val)
  Set a long value from a key.
- int grib_set_double (grib_handle *h, const char *key, double val)
  Set a double value from a key.
- int grib_set_string (grib_handle *h, const char *key, const char *mesg, size_t *length)
  Set a string value from a key.
- int grib_set_bytes (grib_handle *h, const char *key, const unsigned char *bytes, size_t *length)
  Set a bytes array from a key.
- int grib_set_double_array (grib_handle *h, const char *key, const double *vals, size_t length)
  Set a double array from a key.
- int grib_set_long_array (grib_handle *h, const char *key, const long *vals, size_t length)
  Set a long array from a key.
- void grib_dump_content (grib_handle *h, FILE *out, const char *mode, unsigned long option_flags, void *arg)
  Print all keys, with the context print procedure and dump mode to a resource.
- void grib_get_all_names (grib_handle *h, char *names)
  Gather all names available in a handle to a string, using a space as separator.
- void grib_dump_action_tree (grib_context *c, FILE *f)
  Print all keys from the parsed definition files available in a context.

- grib_context * grib_get_context (grib_handle *h)
  Retreive the context from a handle.
- grib_context * grib_context_get_default (void)
  Get the static default context.
- grib_context * grib_context_new (grib_context *c)
  Create and allocate a new context from a parent context.
- void grib_context_delete (grib_context *c)
  Frees the cached definition files of the context.
- void grib_gts_header_on (grib_context *c)
  Set the gts header mode on.
- void grib_gts_header_off (grib_context *c)
  Set the gts header mode off.
- void grib_gribex_mode_on (grib_context *c)
  Set the gribex mode on.
- void grib_gribex_mode_off (grib_context *c)
  Set the gribex mode off.
- void grib_context_set_user_data (grib_context *c, void *udata)
  Sets user data in a context.
- void * grib_context_get_user_data (grib_context *c)
  get userData from a context
- void grib_context_set_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree, grib_realloc_proc gribrealloc)
  Sets memory procedures of the context.
- void grib_context_set_persistent_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree)
  Sets memory procedures of the context for persistent data.
- void grib_context_set_buffer_memory_proc (grib_context *c, grib_malloc_proc griballoc, grib_free_proc gribfree, grib_realloc_proc gribrealloc)
  Sets memory procedures of the context for large buffers.
- void grib_context_set_path (grib_context *c, const char *path)
  Sets the context search path for definition files.
- void grib_context_set_dump_mode (grib_context *c, int mode)
  Sets context dump mode.
- void grib_context_set_print_proc (grib_context *c, grib_print_proc printp)

Sets the context printing procedure used for user interaction.

- void grib_context_set_logging_proc (grib_context *c, grib_log_proc logp)

  Sets the context logging procedure used for system (warning, errors, infos .

- void grib_multi_support_on (grib_context *c)

  Turn on support for multiple fields in single grib messages.

- void grib_multi_support_off (grib_context *c)

  Turn off support for multiple fields in single grib messages.

- long grib_get_api_version (void)

  Get the api version.

- void grib_print_api_version (FILE *out)

  Prints the api version.

- grib_keys_iterator * grib_keys_iterator_new (grib_handle *h, unsigned long filter_flags, char *name_space)
- int grib_keys_iterator_next (grib_keys_iterator *kiter)
- const char * grib_keys_iterator_get_name (grib_keys_iterator *kiter)
- int grib_keys_iterator_delete (grib_keys_iterator *kiter)
- int grib_keys_iterator_rewind (grib_keys_iterator *kiter)
- int **grib_keys_iterator_set_flags** (grib_keys_iterator *kiter, unsigned long flags)
- void **grib_update_sections_lengths** (grib_handle *h)
- const char * grib_get_error_message (int code)

  Convert an error code into a string.

- const char * **grib_get_type_name** (int type)
- int **grib_get_native_type** (grib_handle *h, const char *name, int *type)
- void **grib_check** (const char *call, const char *file, int line, int e, const char *msg)
- int **grib_set_values** (grib_handle *h, grib_values *grib_values, size_t arg_count)
- grib_handle * **grib_handle_new_from_partial_message_copy** (grib_context *c, const void *data, size_t size)
- grib_handle * **grib_handle_new_from_partial_message** (grib_context *c, void *data, size_t buflen)
- int **grib_is_missing** (grib_handle *h, const char *key, int *err)
- int **grib_set_missing** (grib_handle *h, const char *key)
- int **grib_get_gaussian_latitudes** (long truncation, double *latitudes)
- int **grib_julian_to_datetime** (double jd, long *year, long *month, long *day, long *hour, long *minute, long *second)
- int **grib_datetime_to_julian** (long year, long month, long day, long hour, long minute, long second, double *jd)
- long **grib_julian_to_date** (long jdate)
- long **grib_date_to_julian** (long ddate)
- int **wmo_read_any_from_file** (FILE *f, void *buffer, size_t *len)
- int **wmo_read_any_from_stream** (void *stream_data, long(*stream_proc)(void *, void *buffer, long len), void *buffer, size_t *len)
- void * **wmo_read_any_from_file_malloc** (FILE *f, int *err)

6.1.1 Detailed Description

grib_api C header file

This is the only file that must be included to use the grib_api library from C.

6.1.2 Define Documentation

6.1.2.1 #define GRIB_7777_NOT_FOUND -5

Missing 7777 at end of message.

6.1.2.2 #define GRIB_ARRAY_TOO_SMALL -6

Passed array is too small.

6.1.2.3 #define GRIB_BUFFER_TOO_SMALL -3

Passed buffer is too small.

6.1.2.4 #define GRIB_CODE_NOT_FOUND_IN_TABLE -8

Code not found in code table.

6.1.2.5 #define GRIB_CONCEPT_NO_MATCH -37

Concept no match.

6.1.2.6 #define GRIB_CONSTANT_FIELD -48

Constant field.

6.1.2.7 #define GRIB_DECODING_ERROR -14

Decoding invalid.

6.1.2.8 #define GRIB_ENCODING_ERROR -15

Encoding invalid.

6.1.2.9 #define GRIB_END -40

End of resource.

6.1.2.10 #define GRIB_END_OF_FILE -1

End of ressource reached.

Examples:

get.f90, get_data.f90, keys_iterator.f90, multi.f90, and samples.f90.

6.1.2.11 #define GRIB_END_OF_INDEX -43

End of index reached.

Examples:

index.f90.

6.1.2.12 #define GRIB_FILE_NOT_FOUND -7

File not found.

6.1.2.13 #define GRIB_GEOCALCULUS_PROBLEM -17

Problem with calculation of geographic attributes.

6.1.2.14 #define GRIB_INTERNAL_ARRAY_TOO_SMALL -46

An internal array is too small.

6.1.2.15 #define GRIB_INTERNAL_ERROR -2

Internal error.

6.1.2.16 #define GRIB_INVALID_ARGUMENT -20

Invalid argument.

6.1.2.17 #define GRIB_INVALID_FILE -28

Invalid file id.

6.1.2.18 #define GRIB_INVALID_GRIB -29

Invalid grib id.

6.1.2.19 #define GRIB_INVALID_INDEX -30

Invalid index id.

6.1.2.20 #define GRIB_INVALID_ITERATOR -31

Invalid iterator id.

6.1.2.21 #define GRIB_INVALID_KEYS_ITERATOR -32

Invalid keys iterator id.

6.1.2.22 #define GRIB_INVALID_MESSAGE -13

Message invalid.

6.1.2.23 #define GRIB_INVALID_NEAREST -33

Invalid nearest id.

6.1.2.24 #define GRIB_INVALID_ORDERBY -34

Invalid order by.

6.1.2.25 #define GRIB_INVALID_SECTION_NUMBER -22

Invalid section number.

6.1.2.26 #define GRIB_INVALID_TYPE -25

Invalid key type.

6.1.2.27 #define GRIB_IO_PROBLEM -12

Input output problem.

6.1.2.28 #define GRIB_MESSAGE_TOO_LARGE -47

Message is too large for the current architecture.

6.1.2.29 #define GRIB_MISSING_KEY -35

Missing a key from the fieldset.

6.1.2.30 #define GRIB_NO_DEFINITIONS -38

Definitions files not found.

6.1.2.31 #define GRIB_NO_MORE_IN_SET -16

Code cannot unpack because of string too small.

6.1.2.32 #define GRIB_NO_VALUES -41

Unable to code a field without values.

6.1.2.33 #define GRIB_NOT_FOUND -11

Key/value not found.

6.1.2.34 #define GRIB_NOT_IMPLEMENTED -4

Function not yet implemented.

6.1.2.35 #define GRIB_NULL_HANDLE -21

Null handle.

6.1.2.36 #define GRIB_NULL_INDEX -44

Null index.

6.1.2.37 #define GRIB_OUT_OF_AREA -36

The point is out of the grid area.

6.1.2.38 #define GRIB_OUT_OF_MEMORY -18

Out of memory.

6.1.2.39 #define GRIB_PREMATURE_END_OF_FILE -45

End of ressource reached when reading message.

6.1.2.40 #define GRIB_READ_ONLY -19

Value is read only.

6.1.2.41 #define GRIB_STRING_TOO_SMALL_FOR_CODE_NAME -9

Code cannot unpack because of string too small.

6.1.2.42 #define GRIB_SUCCESS 0

No error.

Examples:

iterator.c.

6.1.2.43 #define GRIB_SWITCH_NO_MATCH -49

Switch unable to find a matching case.

6.1.2.44 #define GRIB_VALUE_CANNOT_BE_MISSING -23

Value cannot be missing.

6.1.2.45 #define GRIB_WRONG_ARRAY_SIZE -10

Array size mismatch.

6.1.2.46 #define GRIB_WRONG_GRID -42

Grid description is wrong or inconsistent.

6.1.2.47 #define GRIB_WRONG_LENGTH -24

Wrong message length.

6.1.2.48 #define GRIB_WRONG_STEP -26

Unable to set step.

6.1.2.49 #define GRIB_WRONG_STEP_UNIT -27

Wrong units for step (step must be integer).

6.1.2.50 #define GRIB_WRONG_TYPE -39

Wrong type while packing.

6.1.3 Typedef Documentation

6.1.3.1 typedef struct grib_context grib_context

Grib context, structure containing the memory methods, the parsers and the formats.

6.1.3.2 typedef struct grib_iterator grib_iterator

Grib iterator, structure supporting a geographic iteration of values on a grib message.

Examples:

iterator.c.

6.1.3.3 typedef struct grib_nearest grib_nearest

Grib nearest, structure used to find the nearest points of a latitude longitude point.

Examples:

nearest.c.

6.1.4 Function Documentation

6.1.4.1 void grib_dump_action_tree (grib_context * *c*, FILE * *f*)

Print all keys from the parsed definition files available in a context.

Parameters:

f : the File used to print the keys on

c : the context that containd the cached definition files to be printed

6.1.4.2 void grib_dump_content (grib_handle * *h*, FILE * *out*, const char * *mode*, unsigned long *option_flags*, void * *arg*)

Print all keys, with the context print procedure and dump mode to a resource.

Parameters:

h : the handle to be printed

out : output file handle

mode : available dump modes are: debug wmo c_code

option_flags : all the GRIB_DUMP_FLAG_x flags can be used

arg : used to provide a format to output data (experimental)

6.1.4.3 void grib_get_all_names (grib_handle * *h*, char * *names*)

Gather all names available in a handle to a string, using a space as separator.

Parameters:

h : the handle used to gather the keys

names : the sting to be filled with the names

6.1.4.4 long grib_get_api_version (void)

Get the api version.

Returns:

api version

6.1.4.5 const char* grib_get_error_message (int *code*)

Convert an error code into a string.

Parameters:

code : the error code

Returns:

the error message

6.1.4.6 void grib_print_api_version (FILE * *out*)

Prints the api version.

Index