ECMWF GRIB_API

Sinisa Curic, Batti Filippi, Manuel Fuentes, Baudouin Raoult

Data and Services Section

GRIB_API Overview

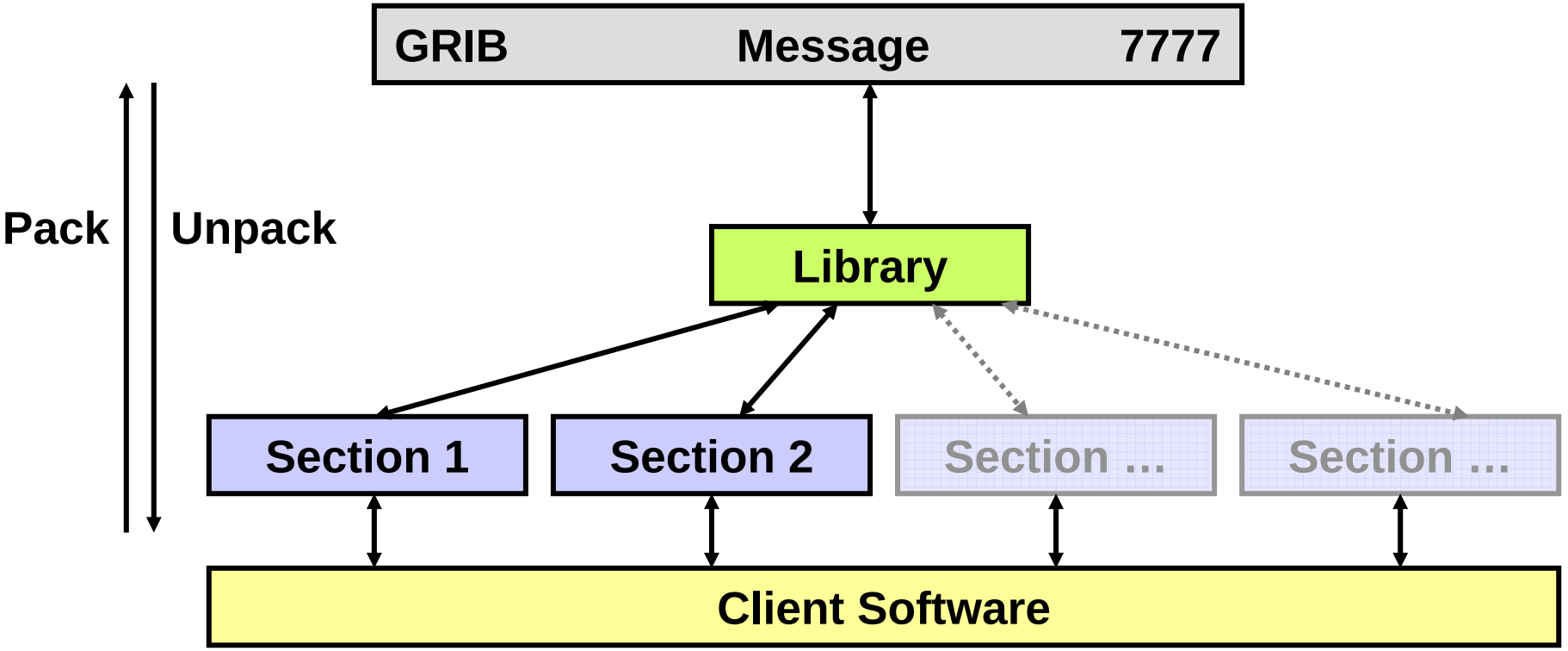
- **Existing libraries**
- **Existing Usage – implications**
- **Presentation of the API**
- **Reading GRIB**
- **Writing GRIB**
- **Associated tools**
- **Status - Conclusion**

GRIB Existing Libraries

- **GRIB 1 :**
  - ➔ **WGRIB – DEGRIB (unpack only) – EMOSLIB (GRIBEX) – GRIB_API**
- **GRIB2 :**
  - ➔ **WGRIB2 – DEGRIB (unpack only) - GRIB_API**

GRIB Existing Libraries

- **- Current Data Library Usage -**



GRIB Existing Libraries

- **Limitation with current usage / library design.**
  - **Programs access data via unpacked “sections”, represented as arrays of values.**
  - **A change or upgrade in the format definition (new local definition, new template..) implies a complicated process, a typical library version lifecycle is :**

    1. Implementation of the change
    2. Compilation
    3. Testing
    4. Installation / distribution
    5. Re-link all the library dependant programs !

GRIB Existing Libraries

- **Implications of current usage / library design.**

  - **Very short lifecycle for libraries versions (average 5 month in ECMWF, 40 different releases installed).**

  - **Release/implementation process is very time consuming, and heavy for users**

  - **Programs using the library are very tight with the data file format used.**

  - **GRIB edition 1 limitations (message size, expressiveness)**

GRIB Implications for GRIB2

- **GRIB2 is an answer to the increase in complexity in data design, based on template to be easily extended.**

- **Expect many extensions during the lifetime of the format, lifecycles will be shorter.**

- **There will be more and different “sections”, client software will require a major rewriting because they are tight to the GRIB1 format.**

- **Coexistence of two edition**
  - **Different units**
  - **Different data layout**

GRIB Implications for GRIB2

- **Main problems with current usage :**

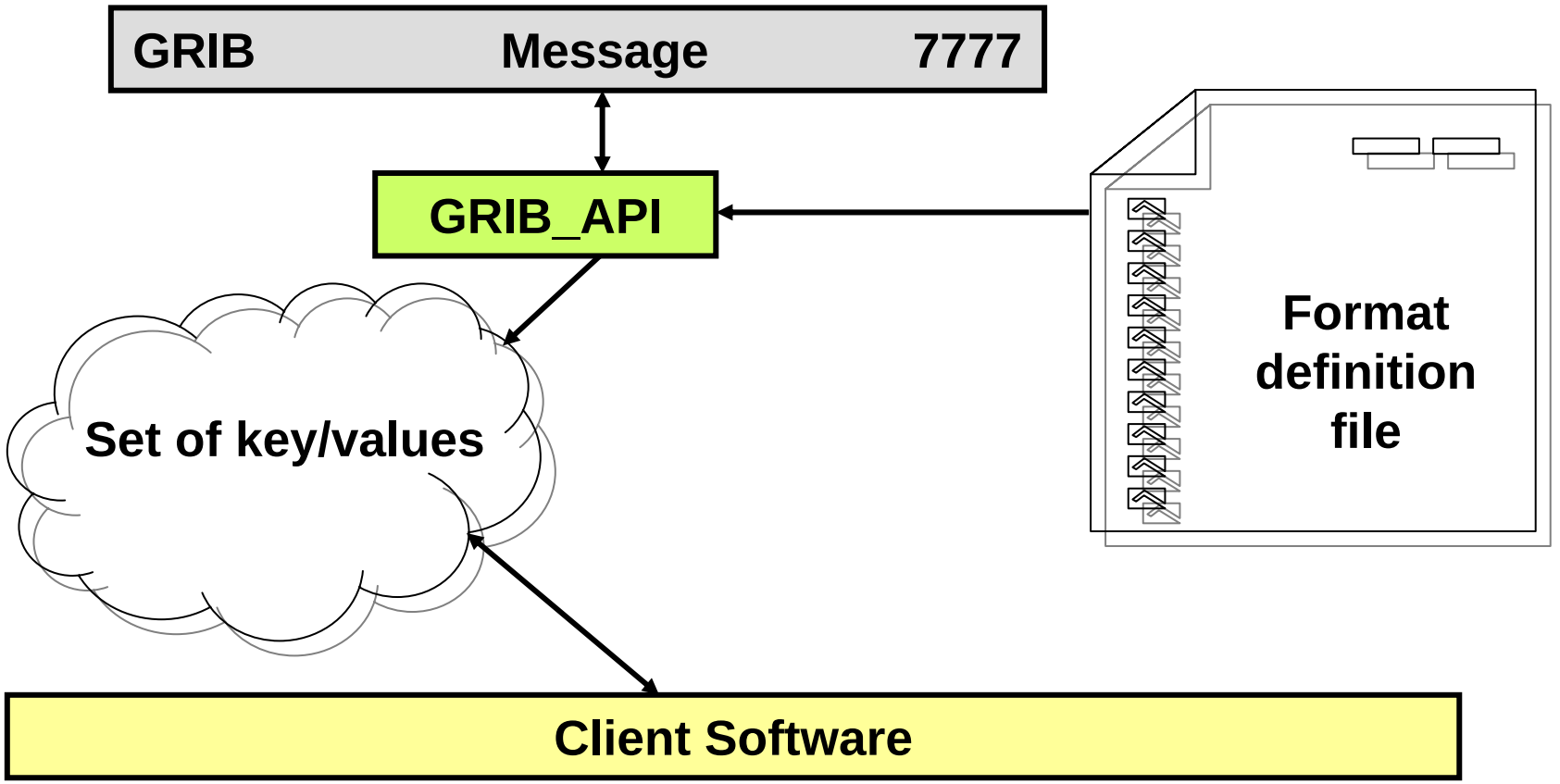
  - ➔ **Current interfaces are too tight to the GRIB format.**

  - ➔ **Libraries lifecycles are too heavy to handle and too short.**

  - ➔ **Coexistence of two editions :**
    - In the archives
    - In the applications

GRIB GRIB_API

- **- Proposed Data Library Usage -**



GRIB GRIB_API

- **Genericity of the interface :**

  - **All messages referred via a "grib_handle" ex :**

    - FILE* f = fopen("sample.grib","r");
    - grib_handle* g = grib_new_from_file(NULL,f);
    - grib_handle* g = grib_new_from_message(NULL,message,size);

  - **Nothing is ever expanded internally, the binary message is the only reference.**
  - **It is possible to open as many handle as memory permits.**

GRIB GRIB_API

- **Genericity of the interface :**

  - **All values referred via a "key" like in a database: ex :**
    - **long x;**
    - **grib_get_long(g,"step",&x);**
  - **The "atomic" element is not the section but a value referred with a character "key".**
  - **When a GRIB message is parsed, a set of keys is available to the client to retrieve the data.**
  - **Handling GRIB 1 and GRIB 2 can be transparent, providing the fact that the keys are the same.**

GRIB GRIB_API

- **Lighter changes in implementation :**

  - **Knowledge of the file format is external to the API, contained in ASCII "Definition Files"**
  - **The library contains all functions to create a set of "key/values"**
  - **There is no need to "re-link" or to recompile when a new template or local definition is available. A typical change in file format implies :**

    1. Writing the definition file
    2. Testing the definition file
    3. Install the definition file (copy in the right directory)

GRIB GRIB_API

- **Sample Definition File :**

```
# SECTION 7, DATA SECTION
# Octets 1-4 : Length of section in octets
# (nn)


length[4] sectionLength ;

# Octet 5 : Number of section

unsigned[1] sectionNumber;

# Octets 6-nn : Data in a format described by Data Template 7.x,
# where x is the Data Representation Template number given in
# octets 10-11 of Section 5


template dataValues
    "grib2/template.7.[dataRepresentationTemplateNumber].def";
```

GRIB GRIB_API

- **Sample Definition File : GRIB 1**

```
codetable[1] indicatorOfTypeOfLevel 'grib1/3.table';
if(indicatorOfTypeOfLevel == 100 or
indicatorOfTypeOfLevel == 160)
{
    unsigned[2] level;
    export levelist(level, mars);
}
else
{
    unsigned[1] topLevel;
    unsigned[1] bottomLevel;
    export levelist(topLevel, mars);
}
```

- **Sample codetable File**

```
1 od Operational archive
2 rd Research department
3 er REANALYSE
4 cs ECSN
5 e4 REANALYSE40
6 dm DEMETER
```

GRIB GRIB_API

- **Sample Definition File : date in GRIB 1**

```
unsigned[1] yearOfCentury ;
unsigned[1] month ;
unsigned[1] day;
…
unsigned[1] century ;

meta  date g1date(century,yearOfCentury,month,day);
```

- **Sample Definition File : date in GRIB 2**

```
unsigned[2] year ;

unsigned[1] month ;

unsigned[1] day ;

meta    date    g2date(year,month,day);
```

GRIB 3 sort of keys

- **Names defined as in the WMO documentation**
  - **Shape of the earth = `shapeOfTheEarth`**
  - **Edition dependent, returned as coded (GRIB unit)**
- **Names that represents higher level concepts**
  - **`date`**
  - **Should exist in both editions**
  - **Returned in edition independent units (e.g. latitude in degrees)**
- **Names specific to ECMWF work practices**
  - **`step`**
  - **`class`**
  - **…**

Example – getting the values in GRIB 1

```
include <grib_api.h>

main(){
  FILE*        f             =  fopen("sample.grib1","r");
  grib_handle* g             =  grib_new_from_file(NULL,f);
  double       values*       = NULL;
  int          ret           = 0;
  size_t       values_length = 0;
  grib_get_size(g,"values",&values_length );
  values = malloc(values_length*sizeof(values));

  ret = grib_get_double_array(g,"values", values,values_length );

  if( ret == GRIB_SUCCESS){
   printf("got the values !!!!")
  }
  fclose(f);
  free(values);
  grib_handle_delete(g);
}
```

Example – getting the values in GRIB 2

```
include <grib_api.h>

main(){
  FILE*        f             =  fopen("sample.grib2","r");
  grib_handle* g             =  grib_new_from_file(NULL,f);
  double       values*       = NULL;
  int          ret           = 0;
  size_t       values_length = 0;
  grib_get_size(g,"values",&values_length );
  values = malloc(values_length*sizeof(values));

  ret = grib_get_double_array(g,"values", values,values_length );

  if( ret == GRIB_SUCCESS){
   printf("got the values !!!!")
  }
  fclose(f);
  free(values);
  grib_handle_delete(g);
}
```

Example – getting the date in GRIB 1

```
include <grib_api.h>

main(){
  FILE*        f             =  fopen("sample.grib1","r");
  grib_handle* g             =  grib_new_from_file(NULL,f);
  long         date          = 0;
  grib_get_long(g,"date",&date);
  printf("The date is %ld", date);
  fclose(f);
  grib_handle_delete(g);
}
```

Example – getting the date in GRIB 2

```
include <grib_api.h>

main(){
  FILE*        f             =  fopen("sample.grib2","r");
  grib_handle* g             =  grib_new_from_file(NULL,f);
  long         date          = 0;
  grib_get_long(g,"date",&date);
  printf("The date is %ld", date);
  fclose(f);
  grib_handle_delete(g);
}
```

Example – getting the mars class in GRIB

```
include <grib_api.h>

main(){
  FILE*        f             =  fopen("sample.grib2","r");
  grib_handle* g             =  grib_new_from_file(NULL,f);
  char         class[1000];
  grib_get_string(g,"mars.class",class,1000);
  printf("The class is %s", class);
  fclose(f);
  grib_handle_delete(g);
}
```

Example – Iterate gridded data

```
include <grib_api.h>

   double lat;
   double lon;
   double val;
   int i = 0;
   grib_iterator* iter = NULL;

    iter = grib_iterator_new(g);
    if(iter)
    {

      while (grib_iterator_next(iter,&lat,&lon,&val))
        printf("lat %g,lon %g,val %g\n",lat,lon,val);

       grib_iterator_delete(iter);
    }
```

Example – Dumping a GRIB in ASCII

```
include <grib_api.h>

main(){
   FILE*           f  =  fopen("sample.grib","r");
   grib_handle* g =  grib_new_from_file(NULL,f);
  while(g){
    grib_dump_content(g,stdout);
    grib_handle_delete(g);
    g =  grib_new_from_file(NULL,f);
  }
```

```
=====> section GRIB
  0,0 constant grib1divider 1000
  0-4 ascii identifier GRIB
  4-7 section_length totalLength = 91537
  7-8 unsigned editionNumber = 1
======> section section1
     ----> label Grib_Section_1
     8-11 section_length sectionLength = 52
     11-12 unsigned gribTablesVersionNo = 128
     12-13 codetable identificationOfOriginatingGeneratingCenter = 98 [ecmf - ECMWF]
     13-14 unsigned generatingProcessIdentificationNumber = 121
     14-15 unsigned gridDefinition = 255
     15-16  codeflag flag = 10000000 [(1=1) 0  Section 2 omited,(1=1) 1  Section 2 included]
     16-17 codetable indicatorOfParameter = 130 [130 - T       Temperature     K       -]
     17-17 sprintf parameter 130.128
```

Coding GRIB

- **New GRIB from “sample”**
  - **Collection of user defined “prototype messages”, without values**
  - **All samples stored on a single directory**
  - **grib_new_from_sample(NULL,”ModelLevel_T511”);**

- **New GRIB from “dump”**
  - **Create a message from ASCII output of a “dump”**
  - **Yet to be implemented….**

Example usage – writing GRIB from template

```
grib_handle* g =  grib_new_from_sample(NULL,”ModelLevel_T511”);

unsigned char* message  = NULL;
size_t             size = 0;

grib_set_string(g,”param” ,”2t”);
grib_set_long  (g,”step”  ,240 );
grib_set_long  (g,”level” ,10 );
grib_set_long  (g,”date”  ,20051116 );
grib_set_long  (g,”time”  ,1545);

grib_set_double_array(g,”values”, values,&values_length );

message =   grib_get_message(g,&size);
```

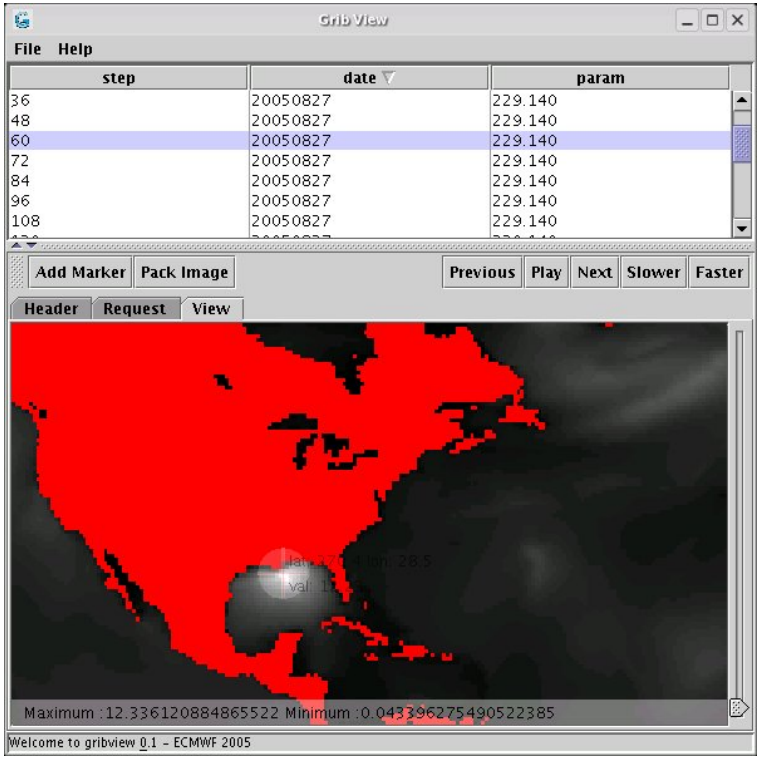
Example usage – writing GRIB from GRIB

```
unsigned char* message  = NULL;
size_t values_length    = 0;
size_t             size = 0;
size_t             i    = 0;

grib_get_size(g,“values”,&values_length );
values      = malloc(values_length*sizeof(double));
ret = grib_get_double_array(g,”values”, values,values_length );

for(i = 0;i< values_length;i++)
       values[i] = log(values[i]);

grib_set_double_array(g,”values”, values,&values_length );
message =   grib_get_message(g,&size);
fwrite(f,size,1,message);
```

GRIB Associated Tools - Experimental

- **Java** **API using JNI**
- **C/C++** **API**
- **Python** **API**
- **Perl** **API**

- **Quick Java viewer for the gribs : “gribview”**



- **Command line interface : “gribshell“**

GRIB Conclusion / Issues

- **Status :**
  - **Part of the operational plot productions**
  - **Magics ++, MagML interpreter**
  - **Mars Client (Alpha)**
  - **Interpolation (Alpha)**
  - **Official release candidate 1.0 for the end of this month.**

- **Issues**
  - **Normalize key names between GRIB1 and GRIB2.**
  - **Normalize code tables between GRIB1 and GRIB2.**
  - **Normalize key meanings between GRIB1 and GRIB2.**