# BUFR decoding with ecCodes

## Introduction to BUFR decoding and ecCodes

Marijana Crepulja, Roberto Ribas, Adrien Oyono

ECMWF Reading UK

Marijana.crepulja@ecmwf.int / Roberto.ribas@ecmwf.int/ adrien.owono@ecmwf.int

ECMWF

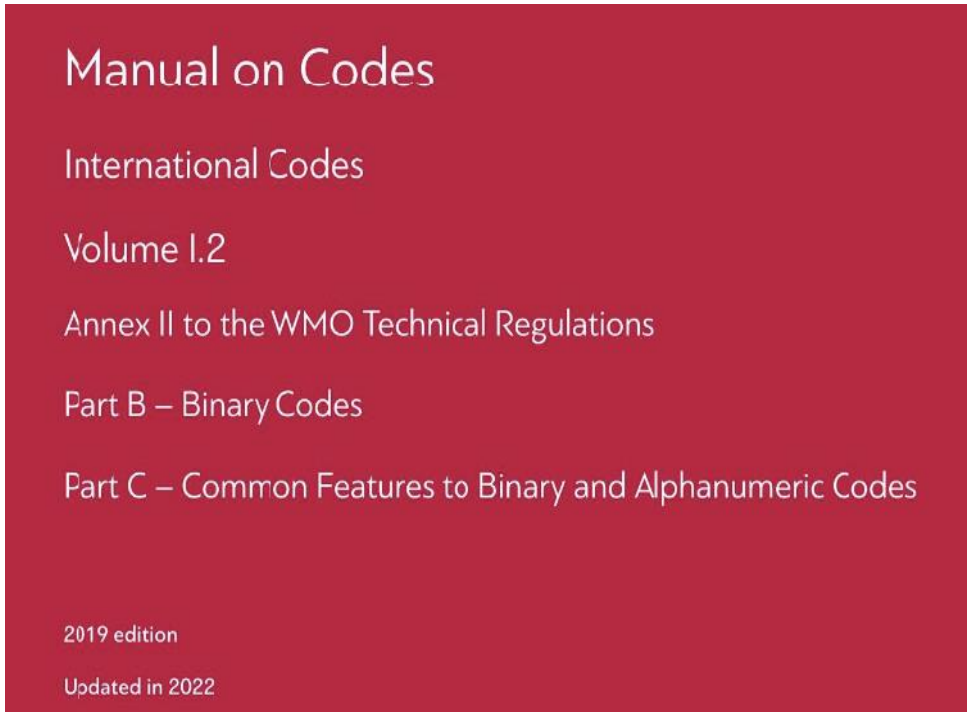
# BUFR format

- **BUFR** is a **B**inary **U**niversal **F**ormat for the **R**epresentation of meteorological data. Supported by the ***World Meteorological Organization*** (WMO) through the **Manual on Codes** https://library.wmo.int/records/item/35625-manual-on-codes-international-codes-volume-i-2 -.X18yfpMza3I
- **BUFR** is a binary format.
- Mainly used for meteorological observations,

(**SYNOP**,**TEMP**, satellite data).

- **BUFR is table driven**( we need the

**BUFR** tables to decode a **BUFR** message).**WMO** provides and updates the **BUFR** tables twice a year.



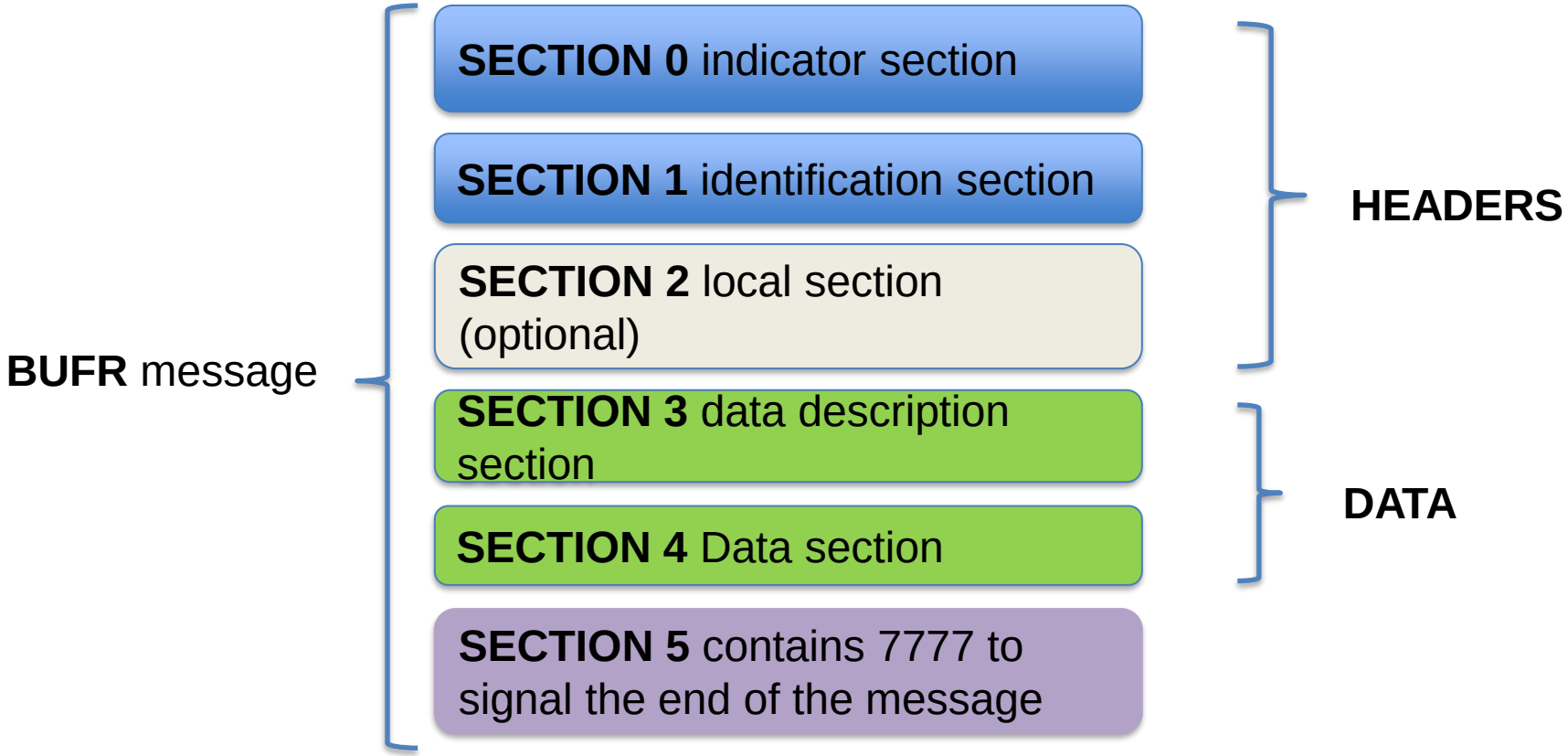
# BUFR tables

- To decode a **BUFR** message, we need to know how to parse the *bit stream* contained in the message. To do so, we need the **WMO BUFR** tables.
- A *bit stream* is a sequence of octets ( 1 octet=8 bits).
- **ecCodes** includes the **WMO BUFR** tables.
- These tables are stored under the **ecCodes** installation directory (you can see where they are by using **codes_info** as we will see later).
- There are 4 tables

| TABLE | CONTENTS |
|---|---|
| **TABLE A** | Data category (surface data/ Radar Data….) |
| **TABLE B** | Contains the BUFR keys ( elements) |
| **TABLE C** | Operators |
| **TABLE D** | BUFR Common Sequences (templates) |

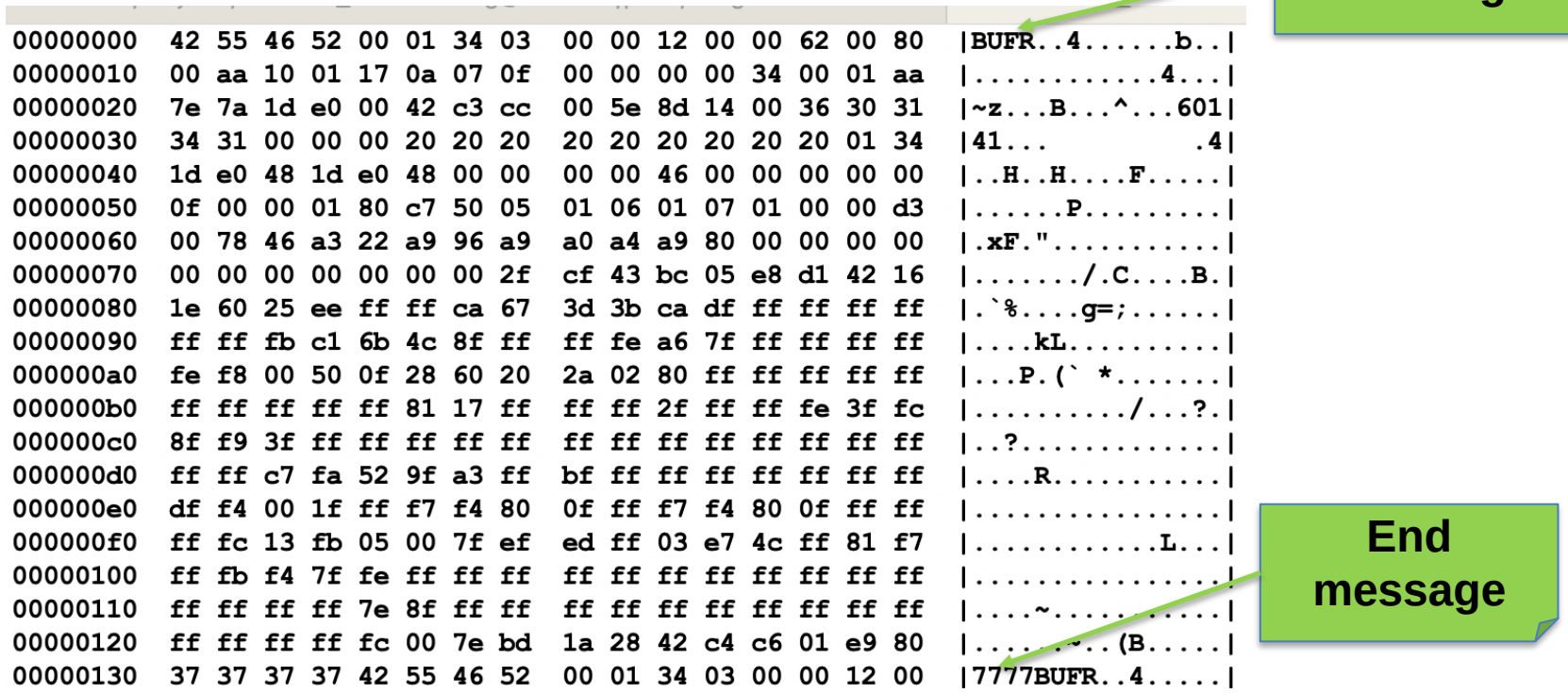
# BUFR files and BUFR messages

- **BUFR** files may contain one or several messages.
- A **BUFR** message contains 6 sections:



# BUFR messages

- Section 0 contains the string **BUFR** the length of the message and the edition number.
- A **BUFR** message is all the data that comes after the string BUFR and before the end of message **7777**.



# BUFR sections 0,1,2 Headers

| | |
|---|---|
| SECTION 0 – Indicator | BUFR<br>Total length of BUFR message<br>BUFR edition number |
| SECTION 1 – Identification | Length of section<br>BUFR master table<br>Identification of originating/generating centre<br>Identification of originating/generating sub-centre<br>Data category<br>International data sub-category<br>Local data sub-category<br>Version number of master tables<br>Version number of local tables<br>Year<br>Month<br>Day<br>Hour<br>Minute<br>Second |
| SECTION 2 – Optional Local | Whatever the originating centre needs for internal purposes (processing, archiving) |

# BUFR Data section (sections 3 and 4)

- Contains the Data description section ( recipe) and the binary data ( ingredients).

**Section 3 : Data Description Section**

```
309007,222000,101000,31002,31031,
1031,1032,101000,31002, 33007
```

**Descriptors**

**Binary data(bit stream)**

**Section 4 : Data section**

```
10010001110001111110010001100010001000
00011000010001000100010000100001000010000
```

BUFR Message

# BUFR descriptors (elements) and its attributes

## BUFR Table B (descriptors starting with 0)

encoding parameters

meaning

Elements descriptors

**Class 12 – BUFR/CREX Temperature**

| TABLE REFERENCE F X Y | ELEMENT NAME | BUFR | | | | CREX | | |
|---|---|---|---|---|---|---|---|---|
| | | UNIT | SCALE | REFERENCE VALUE | DATA WIDTH (Bits) | UNIT | SCALE | DATA WIDTH (Characters) |
| 0 12 001 | Temperature/air temperature | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 002 | Wet-bulb temperature | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 003 | Dewpoint temperature | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 004 | Air temperature at 2 m | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 005 | Wet-bulb temperature at 2 m | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 006 | Dewpoint temperature at 2 m | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 007 | Virtual temperature | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 011 | Maximum temperature, at height and over period specified | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 012 | Minimum temperature, at height and over period specified | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 013 | Ground minimum temperature, past 12 hours | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 014 | Maximum temperature at 2 m, past 12 hours | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 015 | Minimum temperature at 2 m, past 12 hours | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 016 | Maximum temperature at 2 m, past 24 hours | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 017 | Minimum temperature at 2 m, past 24 hours | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 021 | Maximum temperature at 2 m | K | 2 | 0 | 16 | °C | 2 | 4 |
| 0 12 022 | Minimum temperature at 2 m | K | 2 | 0 | 16 | °C | 2 | 4 |
| 0 12 023 | Temperature | °C | 0 | –99 | 8 | °C | 0 | 2 |
| 0 12 024 | Dewpoint temperature | °C | 0 | –99 | 8 | °C | 0 | 2 |
| 0 12 030 | Soil temperature | K | 1 | 0 | 12 | °C | 1 | 3 |
| 0 12 049 | Temperature change over specified period | K | 0 | –30 | 6 | °C | 0 | 2 |
| 0 12 051 | Standard deviation temperature | K | 1 | 0 | 10 | °C | 1 | 3 |
| 0 12 052 | Highest daily mean temperature | K | 1 | 0 | 12 | °C | 1 | 3 |

# BUFR descriptors/elements

Each descriptor has a 6 digits code, referred as F-X-Y

- **Contains a list of six digit descriptors in the form F-X-Y → 0-04-006**
- **Descriptors starting with**
  - **F=0** are elements listed in **Table B**
  - **F=1** denote replication of descriptors
  - **F=2** are operators acting on descriptors (**Table C**)
  - **F=3** are sequences of descriptors listed in **Table D**

# BUFR Table B elements and attributes

Each element/descriptor in table B has:

- a **code**, for example **0 12 001** is an element as starts with 0 (F=0).
- a **meaning/description** for example Temperature/Air Temperature
- and attributes such as ***scale***, ***reference*** and ***data width***. These appear in **Table B** for each element.

To retrieve the actual value, **ecCodes** applies the following recipe.

$$\textbf{Actual Value} = (\textbf{intValue} + \textbf{REFERENCE}) * 10^{-\textbf{SCALE}}$$

**intValue** is the integer equivalent of the binary data stored in the message.

# BUFR min and max value to encode

**Value=(intValue+Ref) $10^{-scale}$**

**Precision=$10^{-scale}$**

When we know the **reference**, **data width** and **scale** for an element (from the Tables), we can calculate the **maximum** and **minimum** values for that can be encoded using that element.

**MINIMUM=(Min_int+REF)*$10^{-scale}$**

**MAXIMUM=(Max_int+REF)*$10^{-scale}$**

The Max_int=$2^{dataWidth}$-2, as two numbers are already taken. All 1s mean MISSING VALUE and all 0s mean 0.

# Data replication

**Replications**.

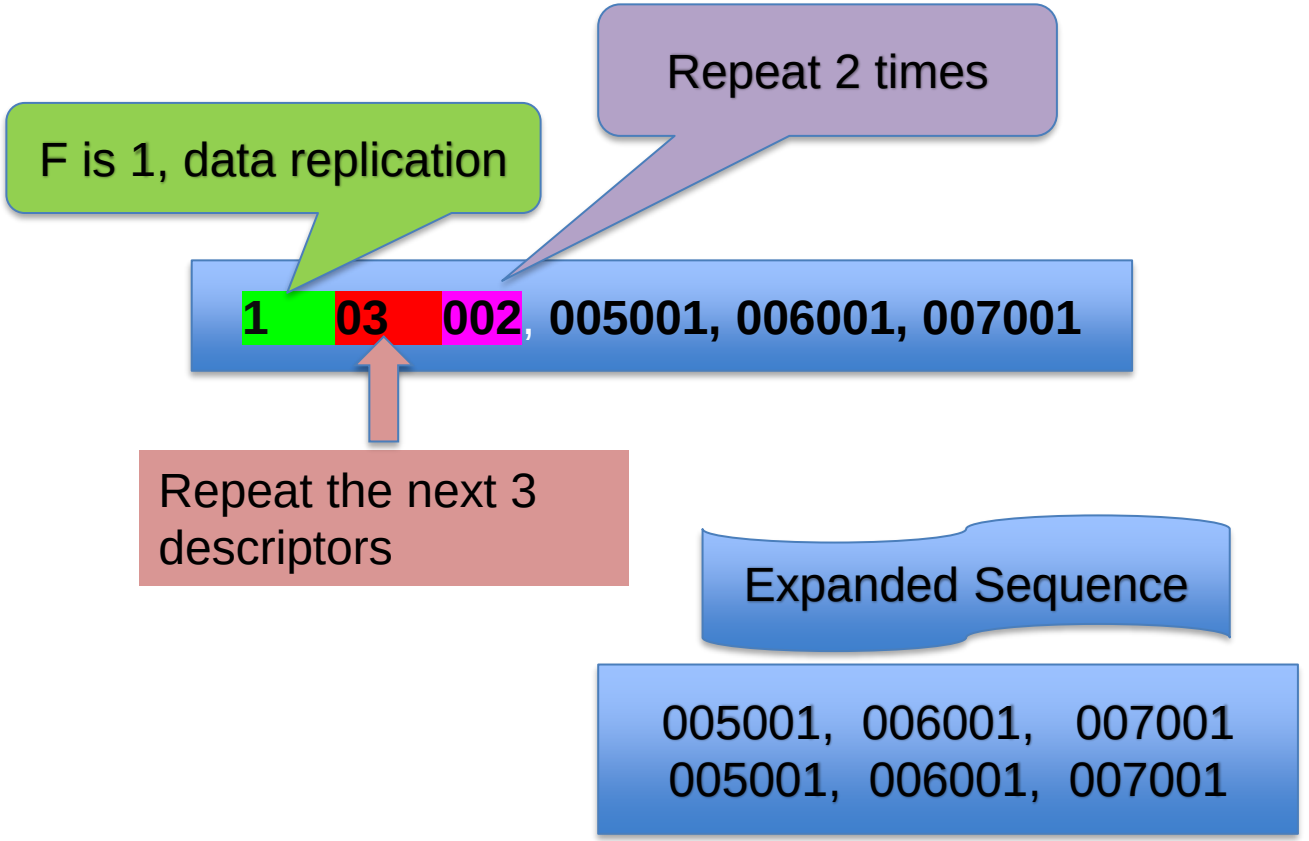
- As mentioned before**, F=1** means data replication.
- Data replication is used to repeat some descriptors. It is very similar to a loop in computer programming languages.
- For instance, a **TEMP** message contains a profile with several values of temperature, wind speed, pressure. Instead of repeating the elements as many times as levels, we use replication.

There are two types:

- **Standard replication**, where we the replication descriptor contains the number of replications.
- **Delayed replication**, where the number of replications is written in the data section.

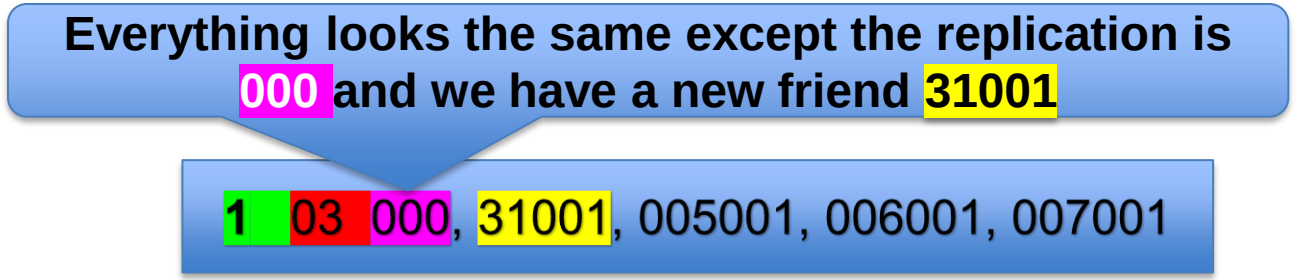
# BUFR descriptors: standard replication

- In **standard replication**, the replication descriptor contains all the information.



# Delayed replication

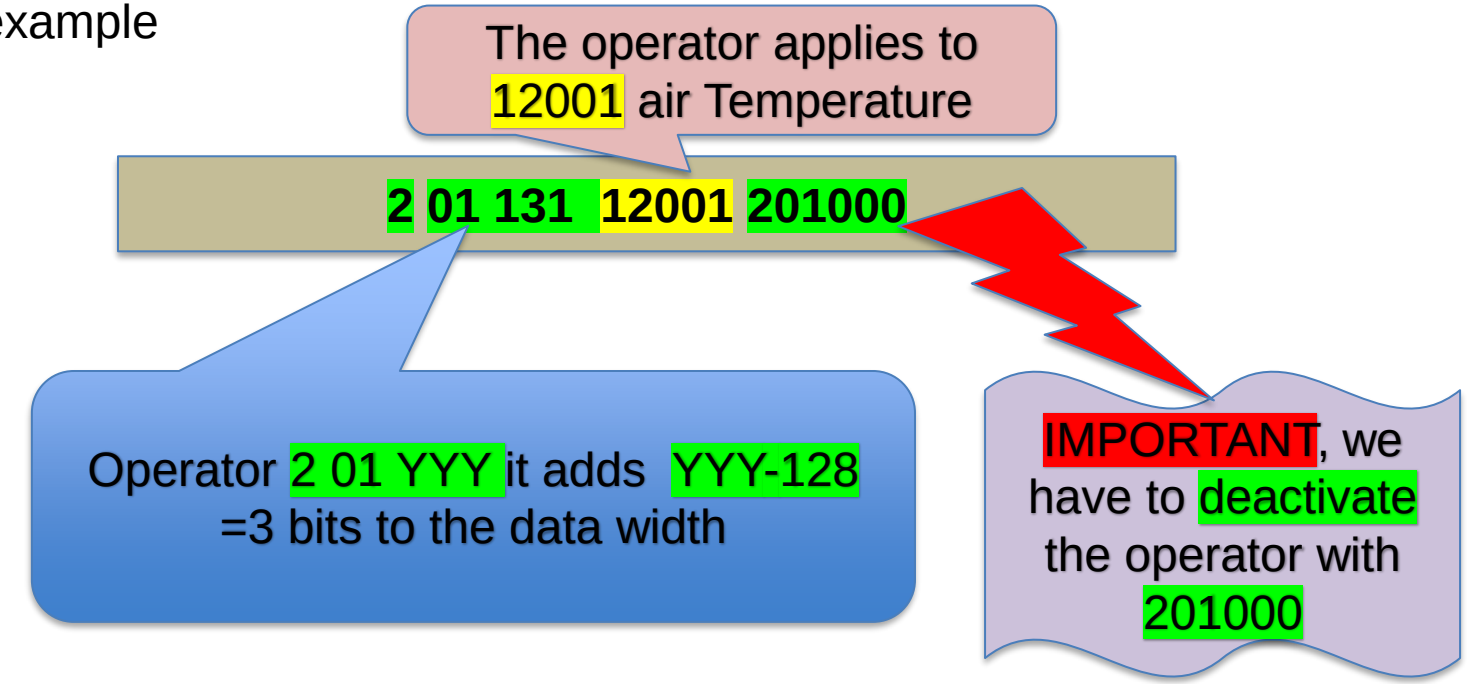
- Sometimes, we don't know the number of replications beforehand. Then, we use delayed replication. In this case, we store the number of replications in the data section.




- This new **31001** descriptor, means we should look in the data section to find out how many replications we have.
- The descriptor **031001** (8 bits) allows 255 replications, the descriptor **031002**( 16 bits) allows 65536 replications.
- The descriptor 31001 is not included in the replication, so in this case, the **03** descriptors to replicate are the ones after 31001( 005001, 006001, 007001)

# Operators

- When **F=2** we have an operator that allows us to modify some descriptor attributes, such as scale, data width etc.
- For example



# BUFR sequences ( templates)

**F=3** indicates sequences. For example:

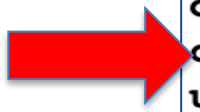
```
"301022" = [  005001, 006001, 007001 ]
```

The sequence **301002** is made of three descriptors 005001, 006001 and 007001. The sequences appear in **Table D**(Common Sequences).

Some sequences expand to other sub sequences and contain different elements inside, including replications. It is important to know the expanded sequence when reading the **BUFR** message.

# Compressed and uncompressed messages

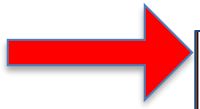
- **compressedData=0**. means uncompressed data,  in general used for single observations.



```
observedData=1
compressedData=0
unexpandedDescriptors={
      307080, 005001, 006001, 007001 }
blockNumber=60
stationNumber=141
stationOrSiteName="FES-SAIS"
stationType=1
year=2023
month=10
day=7
hour=15
minute=0
#1#latitude=33.93
#1#longitude=-4.98
heightOfStationGroundAboveMeanSeaLevel=571
heightOfBarometerAboveMeanSeaLevel=MISSING
nonCoordinatePressure=95230
pressureReducedToMeanSeaLevel=101510
```

# Compressed messages

- **compressedData=1**. Multiple observations in arrays, mainly used for satellite data.



```
compressedData=1
unexpandedDescriptors={
      301072, 030021, 030022, 007024, 005021, 007025, 005022, 010002, 304036, 002152,
      002167, 101010, 304037, 222000, 236000, 101174, 031031, 001031, 001032, 101060,
      033007, 224000, 237000, 001031, 001032, 008023, 101060, 224255 }
satelliteIdentifier=272
#1#centre=160
satelliteClassification=241
segmentSizeAtNadirInXDirection=30000
segmentSizeAtNadirInYDirection=30000
year=2023
month=10
day=7
hour=14
minute=40
second=21
latitude={
      5.22743, 5.23223, 5.23712, 5.24211, 5.24721, 5.25241, 5.25772, 5.26315, 5.2687, 5.27437,
      5.28018, 5.28612, 5.2922, 5.29843, 5.30482, 5.31138, 5.31811, 5.32502, 5.33213, 5.33944,
      5.34697, 5.35474, 5.36275, 5.37104, 5.37961, 5.38849, 5.39772, 5.40731, 5.41731, 5.42776,
```

# ECCODES

# ecCodes

- **ecCodes** is a software package developed by **ECMWF** that simplifies the processing of **GRIB/BUFR** messages(encoding and decoding). It is written in C and has  Fortran, C and Python3 APIs.
- **ecCodes** only works with python3 (python2 support stopped Q1 2023).
- It has also command line utilities for quick access to messages, keys, attributes. It allows also to modify some keys and filter messages according to different criteria.
- **ecCodes** https://confluence.ecmwf.int/display/ECC/ecCodes+Home

# ecCodes

Use **codes_info** to know about **ecCodes** installation ( tables/definitions).

```
(base) [marg@ac6-100 ecCodes_2023]$ codes_info

ecCodes Version 2.28.0

Default definition files path is used: /usr/local/apps/ecmwf-toolbox/2023.01.0.0/GNU/8.4/share/eccode
s/definitions
Definition files path can be changed by setting the ECCODES_DEFINITION_PATH environment variable.

Default SAMPLES path is used: /usr/local/apps/ecmwf-toolbox/2023.01.0.0/GNU/8.4/share/eccodes/samples
SAMPLES path can be changed by setting the ECCODES_SAMPLES_PATH environment variable.
```

Although not recommended, you can add your own local tables
https://confluence.ecmwf.int/display/UDOC/Local+configuration+-+ecCodes+BUFR+FAQ

# Command line tools

- **bufr_count** counts the messages in a **BUFR** file.
- **bufr_ls** to show header information and some data section keys ( need to unpack with option **–s unpack=1** ).
- **bufr_dump** to show the contents of a **BUFR** message.
- **bufr_filte**r allows to apply some actions (set a key, print) for each message.
- **bufr_compare** allows to compare **BUFR** messages.
- **bufr_set** allows to set the value for some keys.
- **bufr_copy** allows copying messages from a **BUFR** file.

https://confluence.ecmwf.int/display/ECC/BUFR+tools

These commands come with the **option –h** for help

# bufr_ls

- **bufr_ls**, used to list some contents(headers) of the BUFR messages inside a file. It can also print some keys in the data section(using **–s unpack=1**).

```
(base) [marg@ac6-100 ecCodes_2023]$ bufr_ls BSSY_1425853031_20231007150035.b
BSSY_1425853031_20231007150035.b
centre                      masterTablesVersionNumber  localTablesVersionNumber   typical
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
ecmf                        16                         1                          202310
```

# bufr_ls

- **bufr_ls** we can select keys with the option **–p**.

```
(base) [marg@ac6-100 ecCodes_2023]$ bufr_ls -p centre,numberOfSubsets BSSY_1425853031_20231007150035.b
BSSY_1425853031_20231007150035.b
centre          numberOfSubsets
98              1
98              1
98              1
98              1
98              1
98              1
```

- We can also see “some” data section keys if we use the option –s unpack=1 although some keys cannot be obtained this way.

```
(base) [marg@ac6-100 ecCodes_2023]$ bufr_ls -s unpack=1 -p blockNumber,stationNumber  msg1.b
msg1.b
blockNumber    stationNumber
60             141
1 of 1 messages in msg1.b
```

# bufr_dump

- **bufr_dump** shows the contents of the **BUFR** message. Useful options :
- -w count=N shows message N.
- -p output in plain text. Default is json.
- -O octet mode.
- -d prints the expanded descriptors.
- -S subset_number. Shows the given subset.

- -D [Fortran|C|python] produces a Fortran or C or python script to decode the **BUFR** message.

More examples https://confluence.ecmwf.int/display/ECC/bufr_dump

# bufr_dump

- **bufr_dump -O msg1.b  |less**

```
***** FILE: msg1.b
#==============   MESSAGE 1 ( length=308 )                ==============
1-4       identifier = BUFR
5-7       totalLength = 308
8         edition = 3
======================   SECTION_1 ( length=18, padding=0 )    ======================
1-3       section1Length = 18
4         masterTableNumber = 0
5         bufrHeaderSubCentre = 0
6         bufrHeaderCentre = 98 [European Centre for Medium-Range Weather Forecasts (commo
7         updateSequenceNumber = 0
8         section1Flags = 128 [10000000 (bufr/section1_flags.table) ]
9         dataCategory = 0
10        dataSubCategory = 170
11        masterTablesVersionNumber = 16
12        localTablesVersionNumber = 1
13        typicalYearOfCentury = 23
14        typicalMonth = 10
15        typicalDay = 7
16        typicalHour = 15
17        typicalMinute = 0
18        section1Padding = 1 {
               00
            } # section_padding section1Padding
======================   SECTION_2 ( length=52, padding=0 )    ======================
1-3       section2Length = 52
4         reservedSection2 = 0
5         rdbType = 1
6         oldSubtype = 170
7-19      keyData = 13 {
                          7e, 7a, 1d, e0, 00, 42, c3, cc, 00, 5e, 8d, 14, 00
```

# Bufr_dump

- Bufr_dump produces a JSON output


```
  {
    "key" : "subsetNumber",
    "value" : 1
  },
  {
    "key" : "blockNumber",
    "value" : 60,
    "units" : "Numeric"
  },
  [

    {
      "key" : "stationNumber",
      "value" : 141,
      "units" : "Numeric"
    },
    [

      {
        "key" : "stationOrSiteName",
        "value" : "FES-SAIS",
        "units" : "CCITT IA5"
      },
      [

        {
          "key" : "stationType",
          "value" : 1,
          "units" : "CODE TABLE"
        },
```

# Bufr_filter

**bufr_filter** allows some simple operations to be done on all the messages of a BUFR file. For example, we could select only some satellite Identifiers from a BUFR file that contains a mix of satellite data. Let's create this filter to write the messages with satelliteIdentifier 64 (Sentinel5P) to an output file.

```
set unpack=1;
if (satelliteIdentifier==64){
set pack=1;
write;
}
```

bufr_filter –o output.bufr **filter.flt** input.bufr

The file **output.bufr** file contains only the Sentinel-5P messages.
More examples https://confluence.ecmwf.int/display/ECC/bufr_filter

# Bufr_copy

- **bufr_copy** allows to copy whole messages from a file.

For example:

```
bufr_copy input.bufr  'out_[bufrHeaderCentre].bufr'
```

- This command will copy the messages in the ***input.bufr*** and will split them according to the **bufrHeaderCentre** key.
- More examples https://confluence.ecmwf.int/display/ECC/bufr_copy

# Bufr_get and bufr_set

- **bufr_get** allows to retrieve some header keys. If the requested key does not exist, unlike **bufr_ls**, **bufr_get** fails with an error that can be trapped in a script.

- **bufr_set** allows to set keys/pairs in a message and write the result in an output file.

More information https://confluence.ecmwf.int/display/ECC/bufr_set and https://confluence.ecmwf.int/display/ECC/bufr_get

# Bufr_compare

- **bufr_compare** is used to compare **BUFR** messages in different **BUFR** files.

For example, we can have a reference **BUFR** file and a test BUFR file. We can compare them by doing:

```
bufr_compare reference.bufr  test.bufr
```

Some handy options:

- **-f force** execution not to fail in error
- **-v verbose**, shows the differences between corresponding messages
- **-d** writes different messages on files.

More information https://confluence.ecmwf.int/display/ECC/bufr_compare

# Eccodes python API

- **Eccodes** has also a Python API that can be used to create decoder/encoder programs which can implement more complex logic than the command line tools.
- To install the python eccodes API use:

```
pip3 install eccodes
```

- https://confluence.ecmwf.int/display/ECC/ecCodes+installation

# Python API

- The structure of a decoder could be:

```
Import eccodes and other python libraries you may need
open BUFR file in binary (mode ‘rb’)
Loop over the messages
    generate the BUFR handle
    unpack the message to access the data section
    read the keys you want using the BUFR handle
    release the BUFR handle
Close BUFR file
```

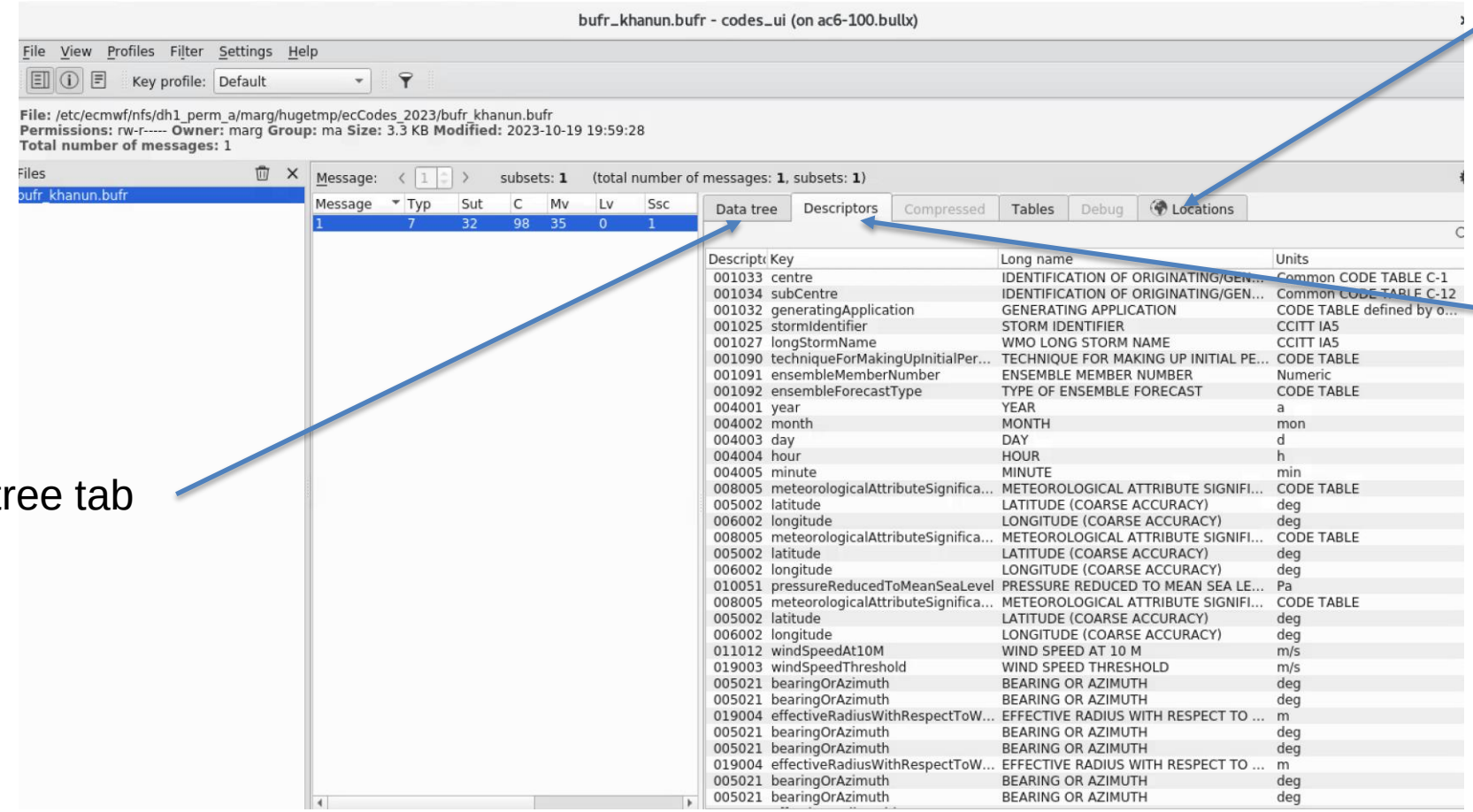
**IMPORTANT**, each **BUFR** handle must be released otherwise memory is exhausted.

# Python example

```
fname="BSSY_1425853031_2023100715003 5.b"
with open(fname,'rb') as f:
  #open BUFR in binary mode for reading 'rb'
    nmsg=codes_count_in_file(f) # count messages
    print(f" file {fname} contains {nmsg}messages")
    for i in range(0,nmsg): # loop over messages
        # get the BUFR handle bid
        bid=codes_bufr_new_from_file(f)
        #unpack to see the data section
        codes_set(bid,'unpack',1)
        # retrieve the keys needed
        lat=codes_get(bid,'latitude')
        print(f" msg {i+1} latitude {lat}")
        codes_release(bid)
      # important to release the BUFR handle before
reading the next message otherwise MEMORY will be exhausted
```

# **codes_ui** the graphical interface

- There is a graphical interface that allows to inspect messages graphically

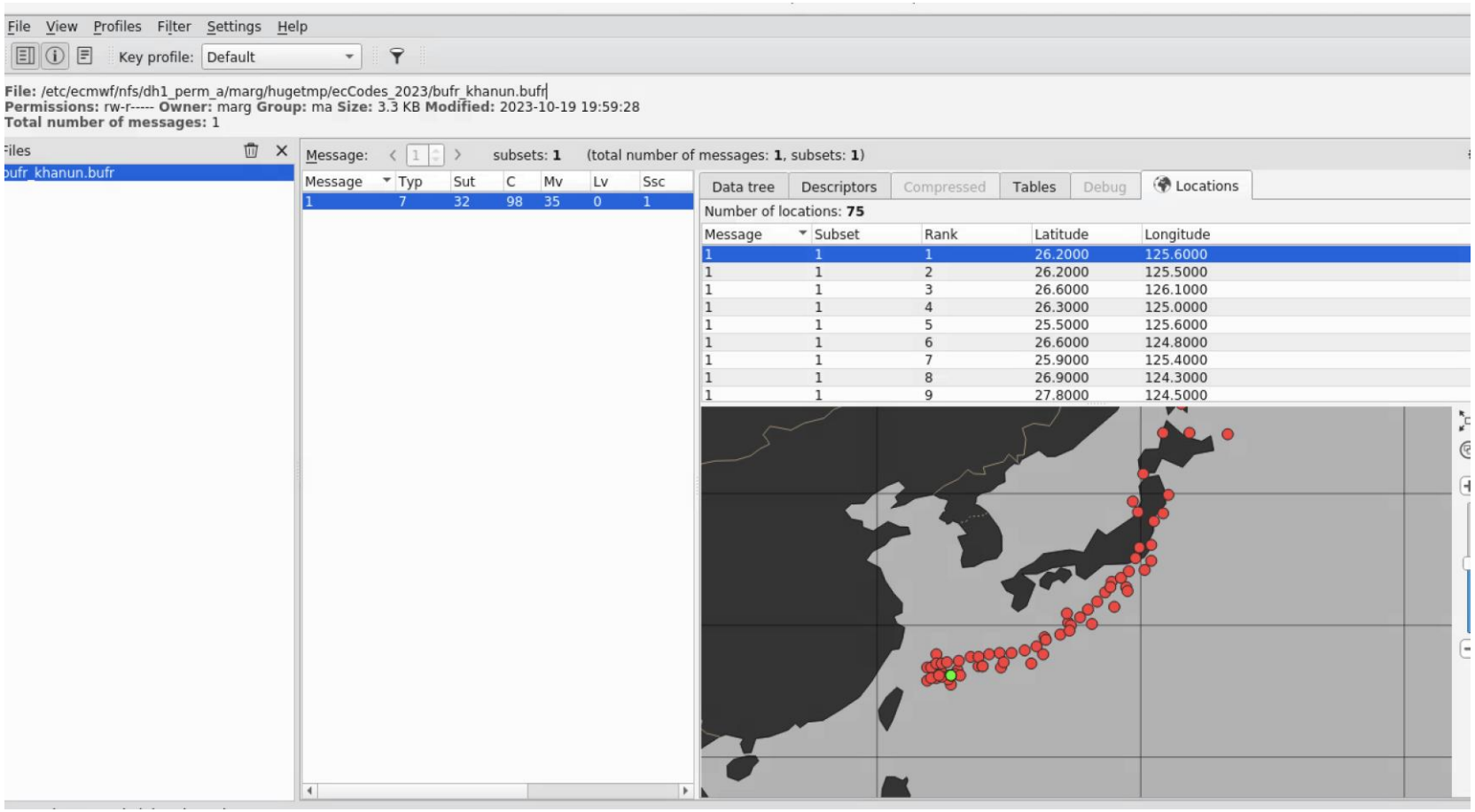



Location tab

Descriptors tab

Data tree tab

# codes_ui allows to see the observations' locations



# Installation

1. Download the **ecCodes** package from: https://confluence.ecmwf.int/display/ECC/Releases
2. Copy to a subdirectory **/tmp** and un compress the ***ecCodes*** package there.
3. Create a subdirectory **/tmp/build**.
4. Use the following command from **/tmp/build** to install ***ecCodes***.

```
cmake -DCMAKE_INSTALL_PREFIX=/path/to/install/eccodes
../eccodes-x.y.z-Source
```

5. Run make and make install to finally install ecCodes.

This command will install the eccodes library under the directory provided in –DCMAKE_INSTALL_PREFIX

# Resources

Videos:

- https://www.youtube.com/watch?v=Xf2YuIzVS7g
- https://www.youtube.com/watch?app=desktop&v=d-iiVT2XxTw
- https://www.youtube.com/watch?v=xQx2214buVU

ECMWF training pages:

- https://learning.ecmwf.int/mod/scorm/player.php?a=176&currentorg=Introduction_to_BUFR_decoding_with_ecCodes_ORG&scoid=452

ecCodes confluence:

- https://confluence.ecmwf.int/display/ECC/ecCodes+Home


thank you
danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
спасибо
gracias
dank je
merci
obrigado
bedankt
dziękuję
go raibh maith agat
sukriya
kop khun krap
terima kasih
dakujem
arigatô
grazie
takk
tapadh leat
mochchakkeram
merci
Баярлалаа
faafetai lava
vinaka
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
chokrane
murakoze
mamnun
дякую
djiere dieuf
tau
kiitos
dankie
dhanyavad
nanni
nandri
hvala
mauruuru
köszönöm
bayarlalaa
grâçie
enkosi
dankon
akun
aciu
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
sobodi
dēkuji
sagolun
didi madloba
mèsi
najis tuke
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
감사합니다
xièxie
ありがとう
tanemirt
rahmet
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merci
мерси
trugarez
spas
welalin
tack
barka
mersi
kia ora