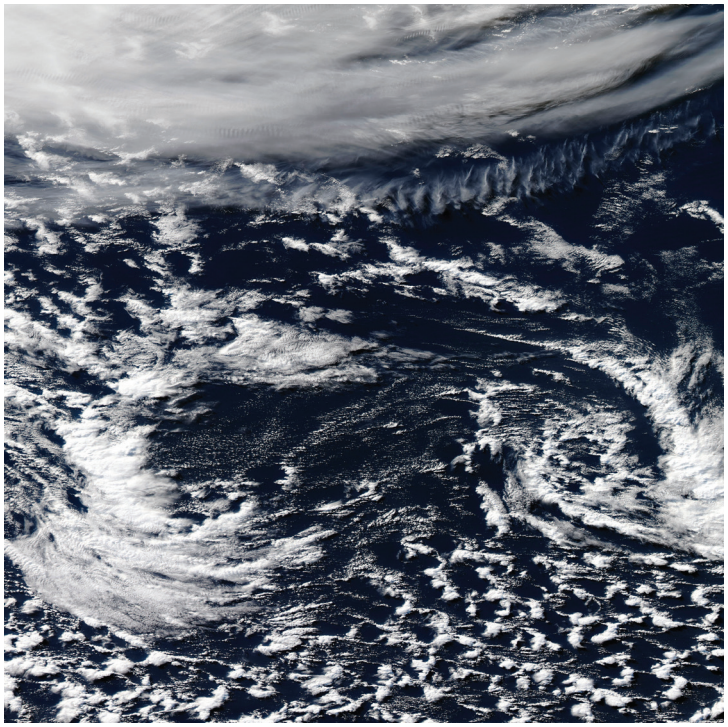
ECMWF Feature article

from Newsletter Number 146 – Winter 2015/16

COMPUTING

ECMWF's new data decoding software ecCodes



NASA Worldview

www.ecmwf.int/en/about/news-centre/media-resources

doi:10.21957/fpnzn5yj

This article appeared in the Computing section of ECMWF Newsletter No. 146 – Winter 2015/16, pp. 35-39.

ECMWF's new data decoding software ecCodes

Enrico Fucile, Sándor Kertész, Sylvie Lamy-Thépaut, Shahram Najm

BUFR and GRIB are binary data formats developed by the World Meteorological Organization (WMO) to enable the worldwide exchange of meteorological data through the WMO's Global Telecommunication System. BUFR (Binary Universal Form for the Representation of meteorological data) is a flexible format mainly used to encode observations, although it can also represent forecast data. GRIB (General Regularly-distributed Information in Binary form) is designed to encode data produced by numerical weather prediction models.

The vast majority of data processed at ECMWF are either in BUFR or GRIB format. Therefore, developing effective software to decode and encode data in these formats is extremely important to facilitate scientific research, build a robust forecasting system and create tools for visualisation and other processing purposes.

ecCodes is a new software package developed at ECMWF to enable decoding and encoding of meteorological data in BUFR or GRIB format using a single, simple interface. It will replace ECMWF's current GRIB decoder/encoder, GRIB-API, and current BUFR decoder/encoder, BUFRDC.

Key/value approach

Since 2011, GRIB-API has been the only software library used to decode and encode GRIB at ECMWF. It is used in several Member States and has been made available as open source software to WMO and the scientific community. GRIB-API has successfully attracted a wide user community because of the simplicity of its programming interface based on a key/value approach. In this approach, any piece of information, such as 'latitude of first grid point', is associated with a key name, such as 'latitudeOfFirstGridPoint'. The association with key names hides the complexity of the underlying GRIB format. Users can access the data using just a few 'get' and 'set' functions and the semantics of the key names. This paradigm has proven to be so effective in facilitating user access to GRIB data that several users have requested a similar software library for BUFR to benefit from the key/value approach.

This is why ECMWF has developed ecCodes, which provides a single programming interface to access both BUFR and GRIB in a consistent manner without any need for detailed knowledge of the binary format. A beta version of ecCodes has been released on the ECMWF website and is available as open source software under the Apache 2.0 license.

ecCodes is based on the key/value approach developed for GRIB-API while extending the key name semantics to cater for the complexity of the BUFR format. The GRIB encoding and decoding features of ecCodes are inherited from GRIB-API and are unchanged. We can think of ecCodes as being GRIB-API with extended decoding/encoding capabilities to deal with BUFR and other formats. ecCodes also inherits its design from GRIB-API: a decoding engine written in C and a set of text definition files. The definition files contain the decoding rules for each data type in a specific language interpreted by the decoding engine.

 doi:10.21957/fpnzn5yj

BUFR format

WMO developed BUFR to replace the Traditional Alphanumeric Codes (TAC) used since the fifties to continuously exchange meteorological data on a worldwide basis. BUFR is a binary format suitable for the exchange of large datasets, while TAC were designed to be readable directly by humans and to be exchanged by telex.

To provide flexibility in the definition of new types of meteorological datasets, BUFR was developed as a Table Driven Code Form (TDCF), with a stable core of encoding rules on the one hand and external tables that are subject to change on the other. WMO provides regular updates of the external tables to describe new datasets. BUFR decoding or encoding software does not require any change to be able to access new datasets defined with new tables available on the WMO website. Using external tables made BUFR flexible, but it has the drawback that users continually have to refer to the tables to interpret the data. The complexity of the rules and the lack of software providing a high-level view of BUFR data make it very difficult for scientific users to make sense of the data without having some knowledge of the underlying binary representation.

Descriptors are the elementary building blocks of BUFR syntax and semantics. They identify the meaning and decoding rule for associated pieces of information (Box A). Each descriptor has a six-digit identifier (such as '012063'). The list of allowed descriptors with their meaning is provided in external tables maintained by WMO, and updates are provided twice a year.

Combining descriptors to build a meaningful data structure requires the knowledge of complex rules. On the other hand, the syntax and semantics of descriptors is flexible enough to allow the representation of any kind of meteorological dataset.

For a full description of the BUFR format, we refer the interested reader to the *WMO Manual on Codes* (2014).

ecCodes key names

The design of ecCodes aims to provide a semantics layer hiding the underlying complexity of BUFR. This is achieved by giving key names to all descriptors (Box A) and providing the software library with 'get' and 'set' functions to access and modify the information associated with each descriptor. For example, the value associated with the descriptor '012063' can be accessed with the key name 'brightnessTemperature' using the following syntax:

```
value=codes_get(bufr, 'brightnessTemperature')
```

In this case, users do not have to know the descriptor code and can transparently access the value using the key name. Moreover, the units of this parameter are available using special syntax applied to the key name:

```
units=codes_get
(bufr, 'brightnessTemperature->units')
```

Units are provided as an attribute of all key names. Other attributes are: 'code', which refers to the BUFR descriptor code value ('012063' in this case); and width, scale and reference, which are related to the binary representation of the data. Many other attributes may be available in a sufficiently complex data structure.

Providing key names has the advantage that the user software clearly references each physical quantity, improving maintainability compared to the use of the obscure descriptor code value, whose semantics is mapped externally in a table. Using key names to retrieve information also helps to avoid potential confusion stemming from the versioning of the external tables. This is because the content of the tables may differ from version to version and the data producer can define their own local tables (although this practice is discouraged by WMO). Therefore, the decoding software should take care of the table version management and provide users with the piece of information required in their specific application. For example, if the user software is retrieving a piece of information associated with the descriptor '012063', this descriptor could have different meanings in different table versions and could be linked to a variable which is different from the required 'brightnessTemperature'. ecCodes sorts out the semantics and provides users with the appropriate piece of information regardless of the BUFR descriptor code value.

doi: 10.21957/fpnzn5yj

In order to provide the necessary support for ecCodes management of BUFR tables, we have reviewed ECMWF local tables and organised the WMO tables into a database. A snapshot of this comprehensive BUFR table database is available from the ECMWF software website (Box B). In the database, a key name is defined for each descriptor, and the content of the database is used to create the text definition files which are used by ecCodes to decode the BUFR data. Tools have been developed to create ecCodes text definition files from the database and to publish the same content on the ECMWF software website, ensuring consistency between ecCodes and the information accessible from the website (Figure 1).

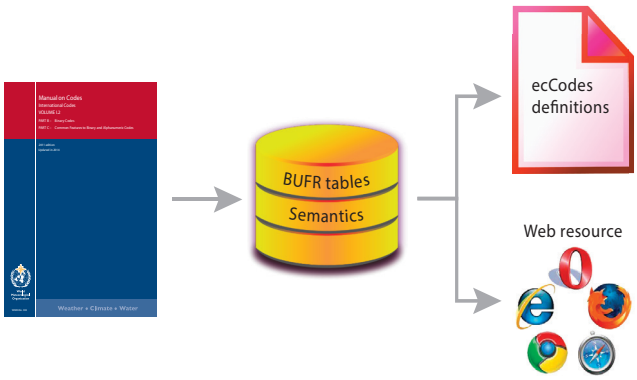


Figure 1 The BUFR tables from the WMO Manual on Codes are inserted into a database and the key names are added. The database is used to create the definitions used by ecCodes for decoding/encoding and to populate web pages for documentation.

BUFR descriptors

A

We can think of BUFR messages as a continuous bit-stream made up of a sequence of octets (1 octet = 8 bits). Each message is composed of five sections. The first three sections contain some headers and the last section marks the end of the message. The information is encoded in sections 3 and 4. Section 3 is a list of descriptors, each of which occupies two octets and is made up of three parts: F (2 bits), X (6 bits) and Y (8 bits). Section 4 contains the data encoded as described in section 3 with the descriptors.

The descriptor F-X-Y uniquely identifies the meaning and decoding rule for the associated piece of information.

There are four types of descriptors:

- F=0 (Element Descriptor). Each element is a single piece of information with a floating point, integer or string value.
- F=1 (Replication Operator). These are used to repeat a sequence of descriptors several times.
- F=2 (Operator Descriptor). These define operations on other descriptors.
- F=3 (Sequence Descriptor). These define sequences of descriptors of any type.

Useful web resources

B

ecCodes home page:
software.ecmwf.int/wiki/display/ECC/ecCodes+Home

ecCodes documentation page:
software.ecmwf.int/wiki/display/ECC/Documentation

A snapshot of ECMWF's BUFR table database:
software.ecmwf.int/wiki/display/ECC/BUFR+tables

New BUFR format validator page:
apps.ecmwf.int/codes/bufr/validator

GRIB-API home page:
software.ecmwf.int/wiki/display/GRIB/Home

WMO International Codes web page:
http://www.wmo.int/pages/prog/www/WMOCodes.html

doi:10.21957/fpnzn5yj

Key name syntax

ecCodes does not only provide a semantic mapping to the BUFR descriptors. It also applies the syntactic rules of the descriptors to provide users with an understandable data structure. To achieve this, a simple syntax for the key names has been developed.

The simple example of using a key name to access a piece of information (an element) is not always applicable because the data structure can be complex and the same key name can appear several times in different branches of the data tree. We may for example have a dataset in which the physical quantity of interest is 'backscatter' and we have several satellite instrument beams in the dataset from which we can get the backscatter. ecCodes provides two options in this situation.

1. Access by rank

The element is accessed using the rank in the data tree structure:

```
value=codes_get(bufr, '#3#backscatter')
```

This returns the backscatter value at rank three in the data structure.

2. Access by condition

The element is accessed by specifying a condition, such as:

```
value=codes_get(bufr, '/beamIdentifier=3/backscatter')
```

This returns the value of backscatter for the data tree branch where 'beamIdentifier' has the value 3.

The attributes for the key name 'backscatter' accessed by rank or condition are still accessible with the attribute syntax, as follows:

```
units=codes_get(bufr, '#3#backscatter->units')
```

or

```
units=codes_get(bufr, '/beamIdentifier=3/backscatter->units')
```

Some of the BUFR features are very complex and difficult to understand for users who are not experts at binary encoding. One of the most complicated features is the bitmap, which links two different elements in the same BUFR dataset. This is useful when a value such as confidence or other quality information needs to be associated with a data item. ecCodes makes the association and exposes the associated values as attributes of the data item. As an example, for a 'temperature' element, we can access the percent confidence as:

```
value=codes_get(bufr, 'temperature->percentConfidence')
```

As for any other key, we can also get the units of percent confidence as:

```
units=codes_get(bufr, 'temperature->percentConfidence->units')
```

Users can access extra information regarding the physical parameter using simple attribute syntax and do not have to know the details of how this information is dealt with at BUFR coding level. They can thus focus on the scientific aspect of data processing without needing to master the details of BUFR coding.

Access by rank or condition and the attribute notation are the only syntax rules currently available in ecCodes, and they are considered sufficient to navigate the data tree and obtain the desired information. They provide easy access to BUFR data and a programming interface hiding the details of the binary code.

Another advantage of defining syntax rules for key names is that they can be used in any supported language, such as Python, Fortran 90 or C, in a consistent way. The result of obtaining information associated with a key name in Python will be the same as in Fortran 90 or C, and the functions in the three languages are very similar.

BUFR encoding

ecCodes can also be used to encode data into BUFR format. This involves two steps: defining the data structure and filling up the structure with values. A BUFR data structure is encoded as a sequence of descriptors. ecCodes provides the key name 'unexpandedDescriptors' to create the desired sequence. After setting this key name, ecCodes creates a skeleton BUFR data structure with all elements of information set to 'missing'. At this point users can set the values using the relevant key names to finalise the BUFR dataset.

If we aim to produce aircraft reports, we can get the descriptor for this type of data from the WMO table, which is '311001', and write the following instruction:

```
codes_set(bufr, 'unexpandedDescriptors', 311001)
```

This instruction will produce a BUFR dataset with a data section ready for data value encoding. We can now set values for all the parameters present in the data structure (e.g. 'airTemperature', 'windDirection', 'windSpeed') by just using the function 'codes_set' as many times as necessary (Box C).

BUFR data encoding

C

bufr_filter is a command-line tool providing a simple language to quickly decode and encode data in BUFR format. The bufr_filter is applying the instructions written in a text file to an input BUFR file, and it can write the encoded BUFR data to an output file.

Below we give the example of the content of a filter instruction file to encode aircraft report data. In this case only the 'set' instruction (equivalent to the 'codes_set' instruction in Fortran or Python) is used to modify the data and produce a new BUFR dataset:

```
# choosing the tables versions
set masterTablesVersionNumber=24;
set localTablesVersionNumber=0;
# data in compressed format
set compressedData=1;
# number of reports to be encoded
set numberOfSubsets=10;
# choosing the aircraft reports template
set unexpandedDescriptors={311001};
# setting data values
set windSpeed = {10,5.3,3.4,4.5,5.6,6,6,32,22,15.2};
set airTemperature = {273.0,300.1,273.0,300.1,273.0,300.1,273.0,300.1,273.0,300.3};
#setting more data
…
# packing
set pack=1;
# writing the data
write;
```

If we write the previous instructions in a file named 'instructions.filter', 'bufr_filter' can be used in the following way to produce the new aircraft report data (output.bufr) from a generic BUFR dataset used as input (input.bufr):

```
bufr_filter –o output.bufr instructions.filter input.bufr
```

The simplicity of the bufr_filter syntax, built on top of the ecCodes key name semantics, provides a powerful tool for BUFR data processing.

doi:10.21957/fpnzn5yj

Programming languages and tools

Python, Fortran 90 and C are the programming languages available in ecCodes. The use of the key name library is very similar in all of them. 'Get' and 'set' functions are used to access information through key names. The key name syntax is the same in all three languages, and this makes it easier to switch from one to the other. The library is designed to be easily extended to other languages. We may add support for more languages in the future if there is enough interest and resources to make the appropriate bindings.

A set of command line tools is also provided to help users perform common tasks in shell scripts. The tools are similar to the GRIB-API tools and have similar behaviour. We suggest starting with the 'bufr_filter' command, which provides a quick way to test key syntax and to experiment with 'setting' and 'getting' information from a dataset. An example of the use of 'bufr_filter' is provided in Box C and more are available from the ecCodes documentation page.

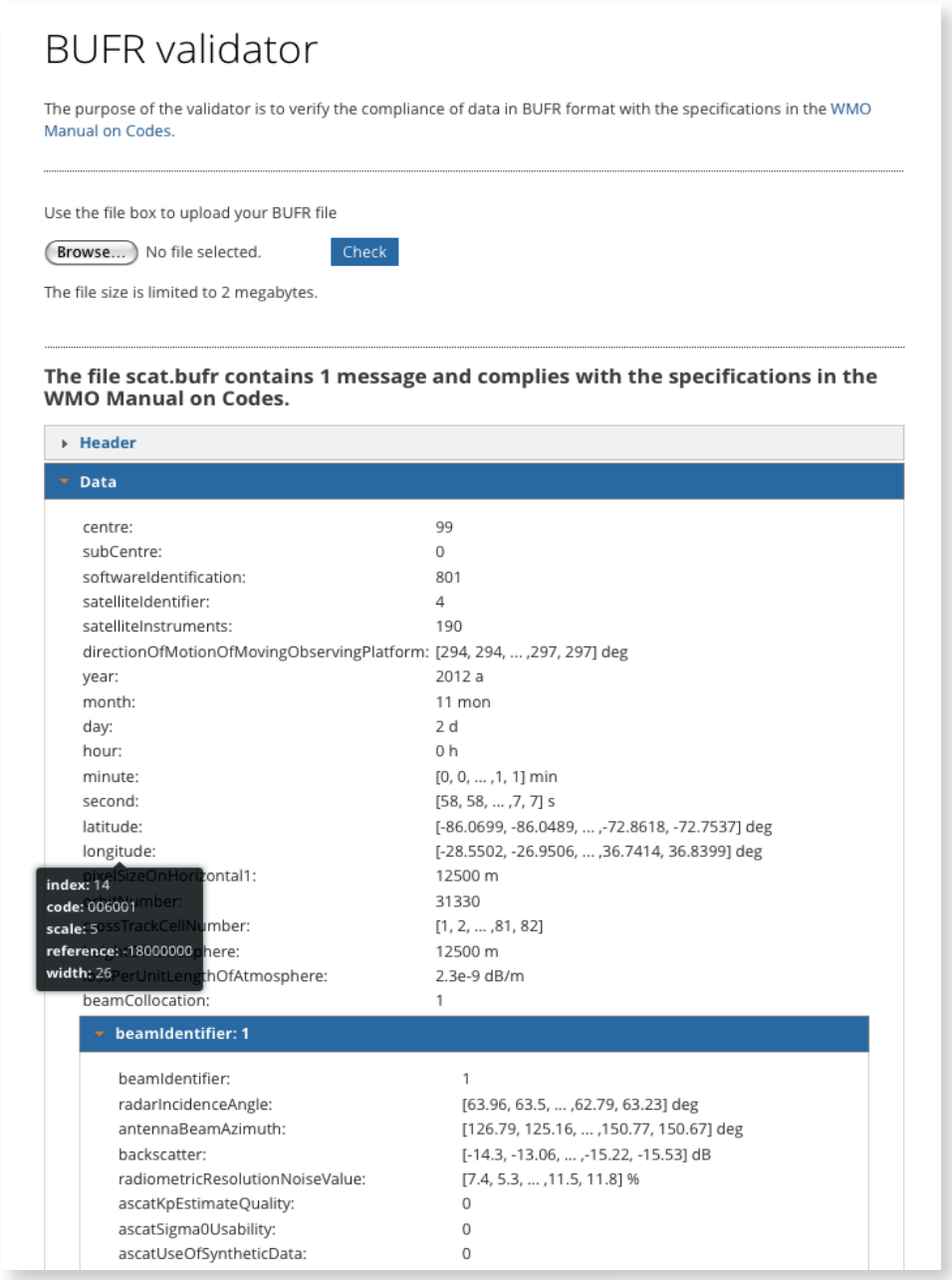


Figure 2 The new BUFR validator, based on the ecCodes decoder, provides a tree view of a BUFR dataset.

Another useful tool is 'bufr_dump', which provides a JSON dump of the content of a BUFR dataset. JSON is the JavaScript Object Notation and is a standard widely used in web applications. Many JSON viewers are also available as web browser plugins. This format is very convenient for inspecting data and allows users to have a view of and navigate the data structure before using ecCodes functions or tools to access the data. The suggested way of working with ecCodes is for users to inspect the data using bufr_dump and a JSON viewer, familiarise themselves with the key names in the dataset using bufr_filter, and eventually write a Python or a Fortran program to make use of the data.

For many years, ECMWF has provided a web page where users can verify whether a BUFR dataset conforms to the WMO standard. This web application was based on the BUFRDC decoder. With the migration to a new website, it has been decided to provide a new version of the validator based on the new ecCodes decoder. This is the first application where we have experimented with creating a JSON dump of the BUFR data. We make use of the output of bufr_dump directly in a JavaScript application to provide a tree view of the dataset (Figure 2). Although the JSON format is excellent for a first look at the data and for interactive navigation, for performance reasons we advise using the library calls in the available languages or the command line tools for data processing purposes.

Release schedule

ecCodes has been built to provide a user-friendly decoder to access data in BUFR and GRIB format. It is based on the key/value approach adopted for the first time in GRIB-API, and it is going to replace GRIB-API and the current BUFR decoder BUFRDC. GRIB-API will be replaced with minimal impact on the user code as most GRIB-API functions and tools will be provided in ecCodes with the same behaviour. BUFRDC will be replaced at a different pace since there are big differences to ecCodes. However, the simplification and improved maintainability provided by ecCodes in the user software should encourage rapid migration to the new decoding library as soon as ecCodes is provided in its full release version. The version of ecCodes that is currently available for download from the ECMWF website is a beta release and cannot be used in an operational environment. The availability of the full release version, which is expected for 2016, will be announced on the ecCodes home page and via the ecCodes mailing list.

Documentation and examples are available on the ecCodes documentation web page, and we suggest interested readers familiarise themselves with ecCodes by going through the examples and contact *software.support@ecmwf.int* if they encounter any issues.

Further reading

WMO, 2014: Manual on Codes. *WMO Publication* **No. 306.**

 doi:10.21957/fpnzn5yj