BUFR/PrepBUFR User's Guide

Version 1.0 Draft

Developmental Testbed Center

December, 2011

Foreword

BUFR (**B**inary **U**niversal **F**orm for the **R**epresentation of meteorological data) is Table Driven Data Representation Forms approved by the World Meteorological Organization (WMO) for operational use since 1988. Since then, it has been used for the representation and exchange of observational data, as well as for archiving of all types of observational data in operation centers, including National Center for Environmental Prediction (NCEP).

BUFR is a self-descriptive table driven code form that offers great advantages of flexibility and expandability compared with the traditional alphanumeric code form as well as packing to reduce message sizes.

As one of the operation centers, NCEP converts and archives all observational data received into a BUFR tank and provides several kinds of BUFR files for its global and regional numerical weather forecast systems. These BUFR files are used by the NCEP operational data analysis system, Gridpoint Statistical Interpolation (GSI), as the standard data sources. Therefore, it is one of DTC's GSI user support tasks to provide suitable documentation for community GSI users to acquire basic knowledge and skills to use BUFR form.

This User's Guide is designed with many simplified examples to help readers understand important concepts in BUFR form and to master basic BUFR file processing skills. There are five chapters to serve this purpose in different layers:

Chapter 1: Using two simple examples to discuss the steps of BUFR file processing and the basic concepts of BUFR form.
Chapter 2: Using three examples to explain the structure and the functions of BUFR file processing code, followed by seven examples to illustrate the processing of BUFR/PrepBUFR files for GSI.
Chapter 3: A detailed explanation and illustration of how to understand a BUFR table.
Chapter 4: An introduction to the interface of GSI for ingesting observational data from various BUFR files.
Chapter 5: An introduction to NCEP observation data processing procedures and NCEP BUFR files.

For more information on BUFR/PrepBUFR, please visit the BUFR User's Website at:
http://www.dtcenter.org/com-GSI/BUFR/index.php

Please send questions and comments to gsi_help@ucar.edu.

Contributors to this guide: Ruifang Li, Ming Hu, Joe Olson

Acknowledgments:
We thank the HFIP project and the National Oceanic and Atmospheric Administration for their support of this work.

Table of Contents

Table of Contents 3

Chapter 1 Introduction to BUFR form and its processing 4
1.1 Basic BUFR encode and decode with a temperature observation 4
1.2 BUFR message structure and basic concepts 7
1.2.1 BUFR message layouts 7
1.2.2 "self-descriptive" table driven code 8
1.2.3 BUFR table descriptor 9
1.2.4 BUFR tables 10
1.2.5 Summary of Decoding Process with BUFR tables 12
1.2.6 Further exploration 13

Chapter 2 Process BUFR/PrepBUFR Files 14
2.1 Encode, Decode, Append a simple BUFR file 16
2.1.1 Decoding/reading data from a simple BUFR file 16
2.1.2 Encoding/writing data into a simple BUFR file 25
2.1.3 Appending data to a simple BUFR file 28
2.2 Encode, Decode, Append the PrepBUFR file 30
2.2.1 Decoding/reading data from a PrepBUFR file 30
2.2.2 Encoding/Writing surface data into a PrepBUFR file 33
2.2.3 Encoding/Writing upper air data into a PrepBUFR file 35
2.2.4 Appending surface data into a PrepBUFR file 38
2.2.5 Appending upper air data into a PrepBUFR file 39
2.2.6 Appending retrieve data into a PrepBUFR file 41
2.3 Decoding/reading radiance data 43

Chapter 3 DX BUFR table 46
3.1 Description of DX BUFR tables 46
3.1.1 WMO BUFR tables and DX BUFR table sections 46
3.1.2 WMO BURF table A, B, D in DX BUFR Table 50
3.1.3 DX BUFR table sections 52
3.2 Examples of the DX BUFR table's application 56

Chapter 4 GSI BUFR interface 61
4.1 GSI observation data ingest and process procedure 61
4.2 The BUFR decoding in GSI read files 66

Chapter 5 NCEP Observation Process and BUFR files 68
5.1 Observation Data Processing at NCEP 68
5.2 NCEP BUFR/PrepBUFR 70
5.3 BUFR/PrepBUFR Data Resources for Community Users 75

Reference and Useful Links: 77

Appendix 78

Chapter 1: Introduction to BUFR form and its processing

BUFR file processing is no harder than any other popular data form. But the World Meteorological Organization (WMO) only provides very detailed documentation on the structure and definition of the BUFR and BUFR tables and leaves the operational centers to write code to implement BUFR form in operation. NCEP NCO provides a library called BUFRLIB to support the application of BUFR file processing. NCO also has very good documentation explaining functions in the BUFRLIB, but lacks training instructions and examples to help beginners to start processing BUFR files smoothly. In this chapter, we will start from two simple examples with step-by-step instructions on how to use BUFR files. After some practice with using BUFR files, we will introduce the structure and several basic concepts of BUFR form to help users better understand BUFR.

As a data form, BUFR files are used to save and exchange meteorological data. There are mainly three actions involved in processing BUFR files:

1) Write the observations into a new BUFR file, which is also called 'encode', because we do need to follow WMO defined BUFR format to save the data;
2) Read the observations from a BUFR file, which is also called 'decode';
3) Append the observations to an existing BUFR file.

Now let's start from the most commonly used example for computer language learning:

"Hello Temperature"

1.1 Basic BUFR encode and decode with a temperature observation

This is an example to write one temperature observation value into a new BUFR file. The 15-line Fortran code to achieve this purpose is given below:

```
Line 01:    program bufr_encode_temperature

Line 02:     implicit none
Line 03:     real(8) :: obs
Line 04:     integer :: iret

Line 05:     obs=10.15

Line 06:! encode
Line 07:     open(20,file='bufrtable.txt')
Line 08:     open(10,file='t.bufr',action='write',form='unformatted')
Line 09:     call openbf(10,'OUT',20)
Line 10:       call openmb(10, 'ADPUPA', 08120100)
Line 11:         call ufbint(10,obs,1,1,iret,'TOB')
Line 12:         call writsb(10)
Line 13:       call closmg(10)
Line 14:     call closbf(10)

Line 15:    end program
```

Based on our knowledge of Fortran code, we can see these 15 lines are to open a binary file called "*t.bufr*" and write a temperature observation value 10.15 into it. But here we didn't see standard Fortran write command. Instead, a set of functions are called to fulfill the write function to a BUFR file: *openbf, openmb, ufbint, writsb, closemg*, and *closbf*. We will explain the usage of these functions in detail in Chapter 2. Here, we only need to know that these functions are from the BUFRLIB library, to "*write/encode*" some information into a BUFR file.

After typing (or copying) the code into a file (let's call it *encode_temperature.f90*), the compile and link steps are the same as any other Fortran code:

```
$ ifort -c encode_temperature.f90
$ ifort -o encodeT.exe encode_temperature.o -L../lib -lbufr
```

Please note that the link step needs a BUFR library. If you have successfully compiled a release version of GSI, this BUFR library exists under subdirectory ./lib. For release version 3, the BUFR library is named as "libbufr_i4r8.a", which has integer and real variable accuracy tags. Basically, the only difference between compiling a BUFR Fortran code and normal Fortran code is that you have to link to the NCEP BUFRLIB library that has been compiled with the sample compiler.

Before running the executable *encodeT.exe*, we need to prepare a BUFR table and put it in the same directory as the *encodeT.exe* file. BUFR is a table driven code that requires a table outside the code (and data file) to help define the values inside the BUFR data file. We will explain what "table driven" is later in this Chapter and will explain how to read NCEP BUFR tables in greater details in Chapter 3 because understanding BUFR tables is the key to processing a BUFR file. Here is a simplified NCEP BUFR table for the purpose of running our first simple example:

```
.-------------------------------------------------------------------------------.
| ------------   USER DEFINITIONS FOR TABLE-A TABLE-B TABLE D   -------------- |
|-------------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                               |
|----------|--------|-----------------------------------------------------------|
| ADPUPA   | A48102 | UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS             |
| TOB      | 012245 | TEMPERATURE OBSERVATION
|          |        |                                                           |
|-------------------------------------------------------------------------------|
| MNEMONIC | SEQUENCE                                                               |
|----------|------------------------------------------------------------------------|
| ADPUPA   | TOB                                                                    |
|          |                                                                        |
|-------------------------------------------------------------------------------|
| MNEMONIC | SCAL | REFERENCE   | BIT | UNITS                    |-------------|
|----------|------|-------------|-----|--------------------------|-------------|
| TOB      |    1 |       -2732 |  14 | DEG C                    |-------------|
`-------------------------------------------------------------------------------'
```

The BUFR table is a text file. Please copy these content into a text file called "*bufrtable.txt*" or download this BUFR table from our BUFR user's page. When you have the encode executable and the BUFR table in the same directory, you can run the executable as:

```
$ encodeT.exe
```

and you will find a new file called "*t.bufr*" in the same directory. The file "*t.bufr*" is a BUFR file that includes only one temperature observation, but the steps involved in creating this file are the same for all BUFR encoding.

So far, we have learned how to encode data into a BUFR file. The following simple example shows how to decode/read data from a BUFR file.

```
Line 01:    program bufr_decode_temperature
Line 02:     implicit none
Line 03:     real(8) :: obs
Line 04:     character(8) subset
Line 05:     integer :: idate,iret

Line 06:! decode
Line 07:     open(10,file=' t.bufr',action='read',form='unformatted')
Line 08:     call openbf(10,'IN',10)
Line 09:       call readmg(10,subset,idate,iret)
Line 10:         call ireadsb(10,iret)
Line 11:         call ufbint(10,obs,1,10,iret,'TOB')
Line 12:         write(*,*) obs
Line 13:     call closbf(10)

Line 14:    end program
```

After typing or copying this code into a file called "*decode_temperature.f90*", we can use the same method as the encode example before to compile and link the code to get an executable:

```
$ ifort -c decode_temperature.f90
$ ifort -o decodeT.exe decode_temperature.o -L../lib -lbufr
```

Unlike the encode process that requires a BUFR table to run, the decode process can get the BUFR table from the BUFR file and read the data from BUFR file without an existing BUFR table. So we can run the executable:

```
$ decodeT.exe
```

and the temperature observation in the BUFR file "*t.bufr*" will show up on screen:

```
10.1000000000000
```

With a first taste of BUFR file encoding and decoding from these examples, we hope you have an idea of the general steps involved in dealing with BUFR files and the computer environmental requirements for working with BUFR files. Successful completion of the above examples means you are ready to learn more details about BUFR basic structure and concepts (if you have a day or two to learn about BUFR more deeply) or to learn how to use the functions in BUFRLIB to solve BUFR file processing problems in real work.

1.2 BUFR message structure and basic concepts

In this section, we will introduce BUFR message structure and some basic concepts related to BUFR. The content of this section is designed for users who want to learn more about the theory of BUFR form. This section will certainly help users to better understand BUFR processing methods and BUFR tables but skipping this section will not impact readers who want to directly use NCEP BUFRLIB functions in Chapter 2 and NCEP BUFR tables (Chapter 3) to process BUFR files from NCEP.

1.2.1 BUFR message layouts

A BUFR file consists of one or many messages. The term "message" refers to BUFR being used as a data transmission format. Each BUFR message consists of a continuous binary stream comprising six sections.

C O N T I N U O U S B I N A R Y S T R E A M					
Section 0	Section 1	Section 2	Section 3	Section 4	Section 5

Section Number	Name	Contents
0	Indicator Section	"BUFR", length of message, BUFR edition number
1	Identification Section	Length of section, identification of the message
2	Optional Section	Length of section and any additional items for local use by data processing centers
3	Data Description Section	Length of section, number of data subsets, data category flag, data compression flag, and a collection of data descriptors which define the form and content of individual data elements
4	Data Section	Length of section and binary data
5	End Section	"7777"

As a general user, we do not need to know the detailed contents in each section of one BUFR message. We do not even need to remember the names of these six sections. But a little knowledge of the message layout can help us understand the concept of 'self-description'. Right now, we can see that a BUFR message starts with the 4 characters "BUFR" and ends with the 4 characters "7777". When the NCEP BUFRLIB function reads in a BUFR message, it will check if the first 4 characters in the message are "BUFR". If the first 4 characters are not "BUFR", the code will stop and give you an error message. In most cases, this error happens when we use a big endian BUFR file in a little endian machine and forget to convert byte-order.

1.2.2 “self-descriptive” table driven code

The most important sections in a message are: section 3, data description section, and section 4, data section, which reflects the concept of “self-descriptive” table driven code. In a BUFR message, section 3 (Data Description Section) contains a sequence of data descriptors, which describe the type of data contained in section 4 (Data Section) and the order in which data appears in section 4. To make the data description section efficient, the descriptors in this section are like a set of “pointers” towards elements in predefined and international agreed tables (stored in the official WMO Manual on Codes). So, the term "self- descriptive" means that the form and content of the data contained within a BUFR message are described within the BUFR message itself.

From the example in this Chapter, the BUFR table is a predefined table:

```
.-------------------------------------------------------------------------------.
| ------------   USER DEFINITIONS FOR TABLE-A TABLE-B TABLE D   -------------- |
|-------------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                               |
|----------|--------|-----------------------------------------------------------|
| ADPUPA   | A48102 | UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS             |
| TOB      | 012245 | TEMPERATURE OBSERVATION
|          |        |                                                           |
|-------------------------------------------------------------------------------|
| MNEMONIC | SEQUENCE                                                           |
|----------|--------------------------------------------------------------------|
| ADPUPA   | TOB                                                                |
|          |                                                                    |
|-------------------------------------------------------------------------------|
| MNEMONIC | SCAL | REFERENCE   | BIT | UNITS                    |-------------|
|----------|------|-------------|-----|--------------------------|-------------|
| TOB      |    1 |       -2732 |  14 | DEG C                    |-------------|
`-------------------------------------------------------------------------------'
```

In this table, we can see number 012245 is a descriptor that points to a set of information about temperature observation, such as MNEMONIC, SCALE, REFRENCE, BIT and UNITS. When encoding this temperature value into the BUFR file “t.bufr”, the data description information (012245) is written into section 3 and observation value (10.1) is written into section 4:

Section 3 (Data Description): `012245`
Section 4 (Data): 101

When decoding the BUFR file “t.bufr”, both data descriptor (012245) and data (101) are read in memory and the BUFR function will find the REFRENCE, BIT and UNITS associated with descriptor 012245 in the predefined table and then decode the data value 101 based on the information in this table as temperature observation 10.1 °C.

Please note, the BUFR table we used here as an example is an NCEP BUFR table. The NCEP BUFR is built upon standard WOM BUFR tables but includes a new column named MNEMONIC, which is an easy mnemonic for WMO descriptor. From the above example, we can see one of the WMO descriptors for temperature observation is 012245, which is very hard to remember, but we should know that it is actually a pointer to a

temperature observation definition in the BUFR table. While in the NCEP BUFR table, 012245 is given a MNEMONIC name "TOB", which is easy to remember. Understanding the NCEP BUFR table is the key to mastering NCEP BUFR file processing because the functions in the NCEP BUFR library are using MNEMONIC as a pointer to link both the WMO descriptor and the table definition (see bold lines in example BUFR table above). Before discuss NCEP BUFR tables in detail in Chapter 3, we will briefly introduce WMO BUFR tables and their descriptors to help users better understand NCEP BUFR tables later.

1.2.3 BUFR table descriptor

BUFR Descriptors

A BUFR descriptor is a set of 16 bits, or two octets. The 16 bits are not to be treated as a 16 bit numeric value, but rather as 16 bits divided into 3 parts F, X, and Y, where the parts (F, X and Y) themselves are 2, 6 and 8 bits, respectively. It is the F X Y descriptors in BUFR Section 3 that refer to data represented in Section 4.

Schematically, a BUFR descriptor can be visualized as follows:

F denotes the type of descriptor. With 2 bits, there are 4 possible values for F: 0, 1, 2 and 3. The four values have the following meanings:

F = 0 ➔ Element descriptor (Table B entry)
F = 1 ➔ Replication operator
F = 2 ➔ Operator descriptor (Table C entry)
F = 3 ➔ Sequence descriptor (Table D entry)

X (6 bits: 00-63) indicates the class or category of descriptor.
Y (8 bits: range from 00-255) indicates the entry within a class X.

In the example before, 012245 is a temperature descriptor, which can be treated as:

F	X	Y
0	12	245

This tells us that it is 245 entry in class 12 (temperature class) of BUFR table B

1.2.4 BUFR tables

BUFR employs 3 types of tables: content definition tables, code tables and flag tables. The BUFR content definition tables contain information to describe, classify and define the contents of a BUFR message. There are 4 such tables defined: Tables A, B, C and D.

All these tables are available on-line from WMO website:

http://www.wmo.int/pages/prog/www/WMOCodes/TDCFtables.html#TDCFtables

Here we will introduce these tables briefly:

Table A subdivides data into a number of discrete categories [e.g. Surface data – land, Surface data - sea, Vertical soundings (other than satellite), Vertical soundings (satellite), etc.]. While not technically essential for BUFR encoding/decoding systems, the data categories in Table A are useful for telecommunications purposes and for storage of data in and retrieval of data from a data base.

Table B describes how individual parameters, or elements, are to be encoded and decoded in BUFR. For each element, the table lists the reference number (or element descriptor number, which is used in the description section of the code like a "pointer", as explained earlier), the element name, and the information needed to encode or decode the element. The data items transmitted in a report will have their descriptor numbers listed in the Data Description Section. As an example, extracts of BUFR Table B for Temperature are given below.

Class 12 - Temperature

TABLE REFERENCE			TABLE ELEMENT NAME	BUFR			
				UNIT	SCALE	REFERENCE VALUE	DATA WIDTH (Bits)
F	X	Y					
0	12	001	Temperature/dry-bulb temperature	K	1	0	12
0	12	002	Wet-bulb temperature	K	1	0	12
0	12	003	Dew-point temperature	K	1	0	12
0	12	004	Dry-bulb temperature at 2 m	K	1	0	12
0	12	005	Wet-bulb temperature at 2 m	K	1	0	12
0	12	006	Dew-point temperature at 2 m	K	1	0	12
0	12	007	Virtual temperature	K	1	0	12
0	12	011	Maximum temperature, at height and over period specified	K	1	0	12
0	12	012	Minimum temperature, at height and over period specified	K	1	0	12

Table B is fundamental to encoding and decoding in BUFR.

The relation between the coded value and the actual value is given by the formula:

coded_value = original_value * 10^scale - reference_value

Scale is used to multiply decimal numbers into integer values (scale > 0) or to reduce precision of large values (scale < 0). Thus, in a way, scale tells the length of the decimal fraction used.

Reference value is used to ensure that the encoded value is always positive.

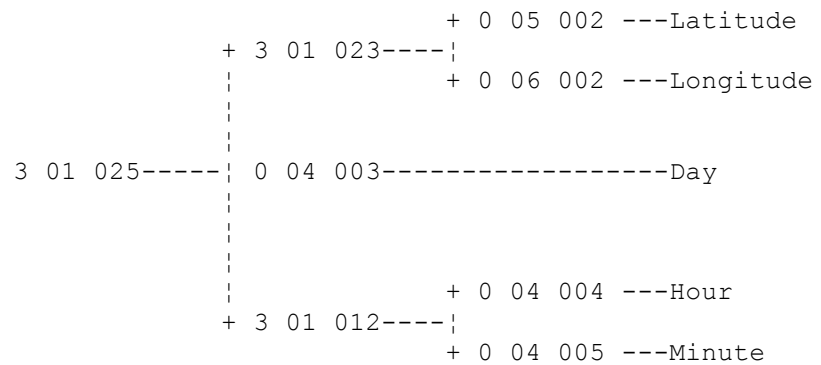
Reference value (together with scale) defines the smallest possible value for the parameter, while data width (together with scale and reference value) defines the largest possible value.

TABLE C defines a number of operations that can be applied to the elements. Each such operation is assigned an operator descriptor.

TABLE D defines groups of elements that are always transmitted together (like a regular SYNOP or TEMP report) in what is called a common sequence. By using a common sequence descriptor, the individual element descriptors will not need to be listed each time in the data description section. This will reduce the amount of space required for a BUFR message. An example of BUFR Table D is shown below.

Sequence descriptors, although not essential for BUFR encoding and decoding, are useful in decreasing the space requirements for BUFR messages.

Table D example: the descriptor 3 01 025 expands to 3 01 023, 0 04 003 and 3 01 012. However, 3 01 023 itself expands to 0 05 002 and 0 06 002, and 3 01 012 expands to 0 04 004 and 0 04 005. Thus, the single Table D descriptor 3 01 025 expands to a total of 5 separate Table B entries.



The order of the data in Section 4 would then be according to the following sequence of Table B entries: 0 05 002, 0 06 002, 0 04 003, 0 04 004, and 0 04 005

Code and Flag Tables: An element based on a code (e.g., Cloud Type) or a set of conditions defined by flags (bits set to 0 or 1) will have an associated Code Table or Flag Table. In this case, "Code Table" or "Flag Table" will appear in the Unit column of Table B. An example of a Code Table and a Flag Table is listed below:

BUFR table B:

Class 20 - Observed phenomena

TABLE REFERENCE			TABLE ELEMENT NAME	BUFR			
				UNIT	SCALE	REFERENCE VALUE	DATA WIDTH (Bits)
F	X	Y					
0	20	024	Intensity of phenomena	Code table	0	0	3

Code table: **0 20 024** ***Intensity of phenomena***

Code figure	
0	No phenomena
1	Light
2	Moderate
3	Heavy
4	Violent
5-6	Reserved
7	Missing value

1.2.5 Summary of Decoding Process with BUFR tables

BUFR decoding software needs to keep the Tables in memory. The decoding process is depicted in figure 1 and summarized below:

- The decoder identifies the successive descriptors in the Data Description Section. If a descriptor is an element descriptor, the decoder looks up the characteristics of the element (units, scale, reference value, data width) in Table B. If a descriptor is a sequence descriptor, the decoder looks up the sequence in Table D. If the sequence in Table D contains only element descriptors, the decoder looks up the characteristics of the elements in Table B and proceeds on to the next descriptor in the Data Description Section. However, if the sequence in Table D contains other sequence descriptors, it looks these up in Table D, repeating this process until only element descriptors remain. The decoder then looks up the characteristics of these elements in Table B and proceeds on to the next descriptor in the Data Description Section. Once the decoder has found the characteristics of all the elements referred to in the Data Description Section, it can decode the values from the Data Section.

- If in Table B, the unit column of the element descriptor contains "Code table" or "Flag table", the interpreter of the decoded data will have to examine the corresponding code table or flag table to understand the meaning of the coded value. The interpreter could be a human or, in some cases, an automatic process that acts depending on the value of the code form or the flags.

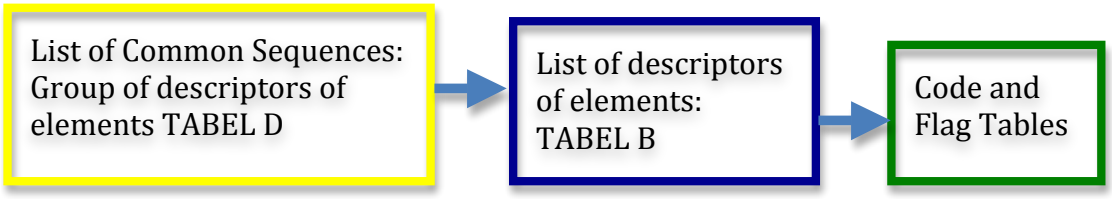


Figure 1. BUFR tables

1.2.6 Further exploration

For more details of BUFR code, users can read the following documents in WMO website:

http://www.wmo.int/pages/prog/www/WMOCodes/Guides/BUFRCREXPreface_en.html

Chapter 2 Process BUFR/PrepBUFR Files

In this Chapter, a set of simple example programs is employed to explain how to process BUFR/PrepBUFR files. The PrepBUFR is the NCEP term for "prepared" or QC'd data in BUFR format (NCEP convention/standard). These examples are Fortran codes and are available in the community GSI release version 3 package under directory *./util/bufr_tools/.* Through these examples, users can easily understand the usage of several commonly used BUFRLIB subroutines, and how these subroutines, together with DX BUFR table, are worked together to encode, decode, append BUFR/PrepBUFR files. These examples can also serve as a starting point for users to solve their specific BUFR file processing problems.

The examples used in this Chapter include:

- bufr_encode_sample.f90
  Write one temperature observation with location and time into a BUFR file.
- bufr_decode_sample.f90
  Read one temperature observation with location and time out from the BUFR file.
- bufr_append_sample.f90
  Append one temperature observation with location and time into an existing BUFR file.
- prepbufr_encode_surface.f90
  Write a surface observation into a PrepBUFR file.
- prepbufr_encode_upperair.f90
  Write an upper air observation into the PrepBUFR file.
- prepbufr_decode_all.f90
  Read all observations and BUFR table out from a PrepBUFR file.
- prepbufr_append_surface.f90
  Append a surface observation into an existing PrepBUFR file.
- prepbufr_append_upperair.f90
  Append an upper air observation into an existing repBUFR file.
- prepbufr_append_retrieve.f90
  Append a retrieved data into an existing PrepBUFR file.
- bufr_decode_radiance.f90
  Read TOVS 1B radiance observations and BUFR table out from the radiance BUFR file.

Please note that all these examples are based on the NCEP BUFRLIB. The BUFRLIB software is a library containing close to 250 different FORTRAN and C subroutines

and functions; however, a typical user will never directly call more than 10-20 of them. The rest are lower-level routines that the software uses to accomplish various underlying tasks and which can therefore be safely treated as "black box" from a user perspective. For more information about BUFRLIB, please refer to

http://www.nco.ncep.noaa.gov/sib/decoders/BUFRLIB/toc/.

The DX BUFR table defines the report structure for each observation type and is embedded at the top of BUFR/PrepBUFR files. We will introduce the DX BUFR table in Chapter 3.

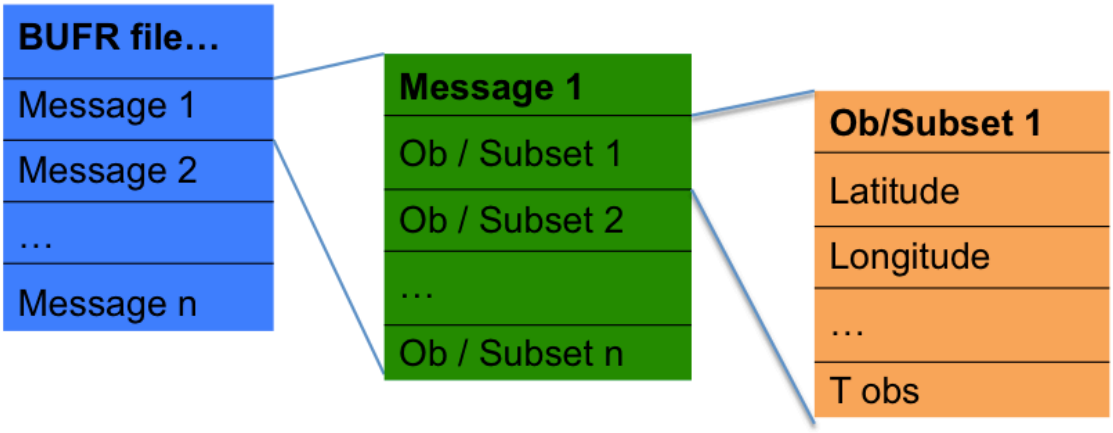
In this Chapter, we will use examples to introduce commonly used BUFRLIB subroutines and functions and the code structure of BUFR processing.

BUFR/PrepBUFR file structure

We introduced BUFR code structure in Chapter 1 but from a practical view, BUFR file structure should be described as: "A BUFR message contains one or more BUFR data subsets. Each data subset contains the data for a single report from a particular observing site at a particular time and location, in addition to time and location information. Typically each data subset contains data values such as pressure, temperature, wind direction and speed, humidity, etc. for that particular observation. Finally, BUFR messages themselves are typically stored in files containing many other BUFR messages of similar content." Therefore, if we summarize in a top-down fashion, we would say:

"A BUFR file contains one or more BUFR messages,
each message containing one or more BUFR data subsets,
each subset containing one or more BUFR data values. "

We can also represent the BUFR/PrepBUFR file structure using the following figure.



2.1 Encode, Decode, Append a simple BUFR file

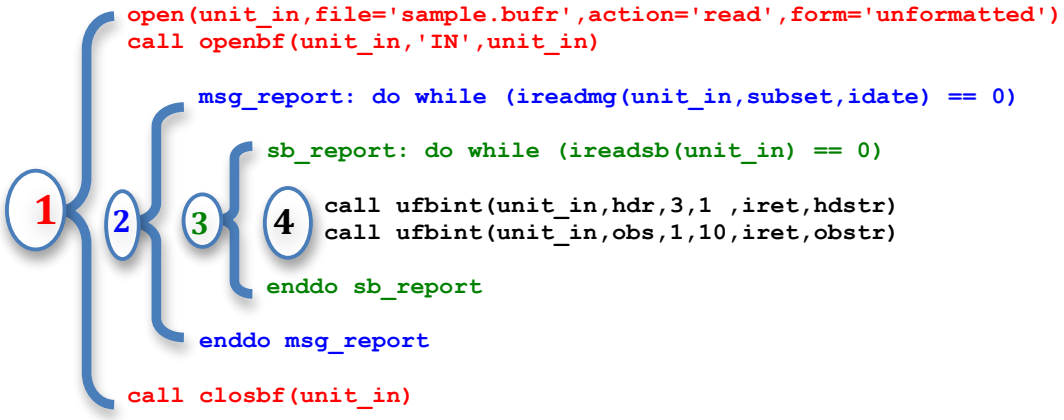
2.1.1 Decoding/reading data from a simple BUFR file

The following is from the code *bufr_decode_sample.f90*, which shows how to read specific observation values (among a large variety) out from a BUFR file.

```
program bufr_decode_sample
!
! example of reading observations from bufr
!
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,10)

 integer :: ireadmg,ireadsb
 character(8) subset
 integer :: unit_in=10
 integer :: idate,iret,num_message,num_subset

! decode
 open(unit_in,file='sample.bufr',action='read',form='unformatted')
 call openbf(unit_in,'IN',unit_in)
 call datelen(10)
   num_message=0
   msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
      num_message=num_message+1
      num_subset = 0
      write(*,'(I10,I4,a10)') idate,num_message,subset
      sb_report: do while (ireadsb(unit_in) == 0)
      num_subset = num_subset+1
      call ufbint(unit_in,hdr,3,1 ,iret,hdstr)
      call ufbint(unit_in,obs,1,10,iret,obstr)
      write(*,'(2I5,4f8.1)') num_subset,iret,hdr,obs(1,1)
      enddo sb_report
   enddo msg_report
 call closbf(unit_in)

end program
```

Specifically, this example will read all temperature observation values with observation location and time from a BUFR file named *sample.bufr*.

The structure of the above FORTRAN BUFR decoding code matches the top-down hierarchy of a BUFR file. To better illustrate this structure, the code is divided into four different levels:



- The 1st Level: the three RED lines are the first level (file level) statements, which open/close a **BUFR file** for decoding.
- The 2nd Level: the two BLUE lines are the second level (message level) statements, which read in **BUFR messages** from the BUFR file. Each loop reads in one message until the last message in the file is reached.
- The 3rd Level: the two GREEN lines are the third level (subset level) statements, which read in **BUFR data subsets** from a BUFR message. Each loop reads in one subset until the last subset in the message is reached.
- The 4th Level: The BLACK lines are the fourth level (data level) statements, which read in user picked **data values** into user defined arrays from each BUFR subset.

All BUFR encode, decode, and append programs have the same structure as listed here. The message loop (`msg_report`) and subset loop (`sb_report`) are needed only if there are multiple messages in a file and multiple subsets in a message, which is the case for most types of observations.

There are several commonly used BUFRLIB subroutines/functions in the code. We will explain the usage of each of them in detail based on the NCO BUFRLIB document. Users are encouraged to read the explanations carefully in parallel to the example code to understand the usage of each function. Understanding the usage of these functions and BUFR file structure are key to successfully processing all NCEP BUFR files.

1st level (file level): open a BUFR file

```
open(unit_in,file='sample.bufr',action='read',form='unformatted')
 call openbf(unit_in,'IN',unit_in)
      …
 call closbf(unit_in)
```

- The ***open*** command: Fortran command to link a BUFR file with a logical unit. Here the action is '*read*' because we want to decode (read) only. The form is always "unformatted" because the BUFR file is a binary stream.

- ***openbf***:

```
CALL OPENBF  ( LUBFR, CIO, LUNDX )

Input arguments:
    LUBFR        INTEGER      Logical unit for BUFR file
    CIO          CHAR*(*)     'IN' or 'OUT' or 'APX' (or NUL', 'NODX',
                                          'SEC3' or 'QUIET')
    LUNDX        INTEGER      Logical unit for BUFR tables
```

This subroutine identifies to the BUFRLIB software a BUFR file that is connected to logical unit *LUBFR*. The argument *CIO* is a character string describing how the file will be used, e.g. 'IN' is used to access an existing file of BUFR messages for reading/decoding BUFR, and 'OUT' is used to access a new file for writing/encoding BUFR. An option 'APX' behaves like 'OUT', except that output is then appended to an existing BUFR file rather than creating a new one from scratch, and there are also some additional options 'NUL', 'NODX', 'SEC3', 'QUIET'. It will be sufficient to further consider only the 'IN', 'OUT', 'APX' cases for the purposes of this discussion. The third argument *LUNDX* identifies the logical unit of DX BUFR table. Except when *CIO*='SEC3', every BUFR file that is presented to the BUFRLIB software must have a DX BUFR tables file associated with it, and these tables may be defined within a separate ASCII text file or, in the case of an existing BUFR file, may be embedded within the first few BUFR messages of the file itself, and in which case the user needs to set *LUNDX* to the same value as *LUBFR*. In any case, note that *LUBFR* and *LUNDX* are logical unit numbers; therefore, the user must have already associated these logical unit numbers with actual filenames on the local system, typically via a FORTRAN "OPEN" statement. Currently, as many as 32 BUFR files can be simultaneously connected to the BUFRLIB software for processing. Of course, each one must have a unique *LUBFR* number and be defined to the software via a separate call to subroutine *OPENBF*.

In this example, LUBFR=LUNDX= *unit_in* since BUFR table is already embedded within the BUFR messages of the file itself. CIO uses 'IN' for reading BUFR file.

- ***closbf***:
  Since *OPENBF* is used to initiate access to a BUFR file, *CLOSBF* would be used to terminate this access:

```
CALL CLOSBF  ( LUBFR )

 Input argument:
    LUBFR     INTEGER      Logical unit for BUFR file
```

  This subroutine severs the connection between logical unit *LUBFR* and the BUFRLIB software. It is always good to call *CLOSBF* for every *LUBFR* that was identified via *OPENBF*; *CLOSBF* will actually execute a FORTRAN "CLOSE" on logical unit *LUBFR* before returning, whereas *OPENBF* did not itself handle the FORTRAN "OPEN" of the same *LUBFR*.

Now that we have covered the library subroutines that operate on the BUFR file level, and recalling the BUFR file structure that was previously discussed, it is now time to continue on to the BUFR message level:

2nd level (message level): read in messages

```
msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
          …
enddo msg_report
```

- Function ***ireadmg***:

```
IRET = IREADMG  (LUBFR, CSUBSET, IDATE)

Input argument:
   LUBFR      INTEGER      Logical unit for BUFR file

Output arguments:
   CSUBSET    CHAR*(*)     Table A mnemonic (name/type) for BUFR message
   IDATE      INTEGER      Section 1 date-time for BUFR message
   IRET       INTEGER      Return code:
                             0 = normal return
                            -1 = no more BUFR messages in LUBFR
```

  Subroutine *IREADMG* reads the next BUFR message from the given BUFR file pointed to by *LUBFR,* returns *IRET* as its function value. It reads the next BUFR message into internal arrays within the BUFRLIB software (from where it can be easily manipulated or further parsed) rather than passed back to the application program directly. If the return code *IRET* contains the value -1, then there are no more BUFR messages within the given BUFR file, and the file will be automatically disconnected from the BUFRLIB software via an internal

call to subroutine *CLOSBF*. Otherwise, if *IRET* returns with the value 0, then the character argument *CSUBSET* will contain the Table A mnemonic which describes a type of data subset, and the integer argument *IDATE* will contain the date-time in format of YYMMDDHH or YYYYMMDDHH determined by subroutine *DATELEN*.

In this example, the loop meg_report will use ireadmg function to read all message in from the BUFR file until getting a none-zero return value (IRET=-1).

After *IREADMG* reads a BUFR message into the internal arrays, we can get into the 3rd level of the code to read a data subset from that internal message:

3rd level (subset level): read in data subsets

```
sb_report: do while (ireadsb(unit_in) == 0)
      …
enddo sb_report
```

- Function ***ireadsb***:

```
IRET = IREADSB  ( LUBFR )

Input argument:
   LUBFR     INTEGER      Logical unit for BUFR file
Output arguments:
   IRET      INTEGER      Return code:
                            0 = normal return
                           -1 = no more BUFR data subsets in
                                current BUFR message
```

Function I*READSB* reads a data subset from the internal arrays. A return code value of -1 within *IRET* indicates that there are no more data subsets within the given BUFR message.

Again, in this example, the loop sb_report will use ireadsb function to read all subset in from the internal array until getting a none-zero return value (IRET=-1).

Once a subset has been successfully read with IRET=0, then we are ready to call the data-level subroutines in order to retrieve actual data values from this subset:

4th level (data level): read in picked data values

This is the level where observation values are read into user-defined arrays. To understand how to read in observations from a BUFR subset, the following two questions need to be addressed:

1) How do I know what kind of data are included in the subset (or a BUFR file)?

This question can be answered by checking the content of a BUFR table and mnemonics. The BUFR table and mnemonics will be discussed in detail by Chapter 3. Here we illustrate how to use the BUFR table to solve the problem directly. As an example, an excerpt from the BUFR table in `sample.bufr` for the message type `ADPUPA` is shown below. We will use this table information to illustrate how to track observation variables in `ADPUPA` (the upper level data type):

```
|-------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| ADPUPA   | A48102 | UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS       |
| AIRCAR   | A48103 | MDCRS ACARS AIRCRAFT REPORTS                        |
```

```
| MNEMONIC | SEQUENCE                                                     |
|----------|--------------------------------------------------------------|
| ADPUPA   | HEADR  SIRC  {PRSLEVEL}  <SST_INFO>  <PREWXSEQ>  {CLOUDSEQ}  |
| ADPUPA   | <CLOU2SEQ>  <SWINDSEQ>  <AFIC_SEQ>  <TURB3SEQ>               |

| HEADR    | SID  XOB  YOB  DHR  ELV  TYP  T29  TSB  ITP  SQN  PROCN  RPT |
| HEADR    | TCOR  <RSRD_SEQ>                                             |
```

```
|-------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| SID      | 001194 | STATION IDENTIFICATI
| XOB      | 006240 | LONGITUDE                                           |
| YOB      | 005002 | LATITUDE                                            |
| DHR      | 004215 | OBSERVATION TIME MINUS CYCLE TI                     |
| ELV      | 010199 | STATION ELEVATION                                   |
| TYP      | 055007 | PREPBUFR REPORT TYP                                 |
```

```
|-------------------------------------------------------------------------|
| MNEMONIC | SCAL | REFERENCE   | BIT | UNITS                |-------------|
|----------|------|-------------|-----|----------------------|-------------|
|          |      |             |     |                      |-------------|
| SID      |    0 |           0 |  64 | CCITT IA5            |-------------|
| XOB      |    2 |      -18000 |  16 | DEG E                |-------------|
| YOB      |    2 |       -9000 |  15 | DEG N                |-------------|
| DHR      |    3 |      -24000 |  16 | HOURS                |-------------|
| ELV      |    0 |       -1000 |  17 | METER                |-------------|
| TYP      |    0 |           0 |   9 | CODE TABL            |-------------|
```

The four color boxes here are used to separate the different parts of the BUFR table, which can also be marked as Part 1 (red), Part 2 (blue), Part 3 (yellow), and Part 4 (green) in the order they are listed above.

As discussed before, *IREADMG* reads in a message with three output arguments. The first output argument is:

```
CSUBSET      Table A mnemonic for BUFR message
```

It returns the message type (also called data type). This message type is the starting point to learn what types of observations are included in this message. The description of message types can be found in the first section of a BUFR table, for example, Part 1 (red) in the sample BUFR table.

Here, if CSUBSET has the value of ADPUPA, the contents of this message or all subsets (third level) are upper air reports (like rawinsonde). A search of ADPUPA in the BUFR table returns the first two lines of Part 2 (blue), in which ADPUPA is followed by a sequence of items like: HEADR SIRC {PRSLEVEL}…. If we then search for HEADR in the same file, we can find the last two lines in Part 2 (blue), in which HEADR leads the sequence containing SID XOB YOB DHR ELV TYP ….

If we then search for SID XOB YOB DHR ELV TYP in the same file, we can find the definition of these items in Part 3 (yellow). Clearly, the message type ADPUPA includes variables like station ID, observation location (longitude, latitude), observation time, etc. These are important variables to describe an observation. If we keep searching for other items under ADPUPA, we can also find lots of observation variables are included in ADPUPA. Please note that a complete list of all variables in a message type could be very long and complex, but we don't need to learn about all of them - we only need to know what we need for our specific application.

The last part of the BUFR table (Part 4, green) includes useful unit information for a variable; for example, the unit of XOB is DEG (degree) and the unit of DHR is HOURS (hours). Users will not likely need to make use of the scale, reference, and bit information.

There are lots of other details on BUFR tables, but the above information should be sufficient for now to learn about BUFR file processing applications using the NCEP BUFRLIB software with the examples in this Chapter.

2). How do I tell BUFRLIB to only read in specific data information?

From the BUFR table discussion above, we can see a message or a subset could include lots of information. In this example, we only wants to read in temperature observation, along with its longitude, latitude, and observation time. Here we will use this example to illustrate how to solve this question. From the BUFR table, for the message type ADPUPA, the name of longitude, latitude, and time in the BUFR table are 'XOB YOB DHR' within the sequence HEADER. Similarly, the name of the temperature observation can be found as 'TOB' in the sequence {PRSLEVEL} (not shown in the example BUFR table). Actually, most conventional message types contain such observation information.

In the example code, the first several lines define the information we want to read:

```
character(80):: hdstr='XOB YOB DHR'
character(80):: obstr='TOB'
real(8) :: hdr(3),obs(1,10)
```

`hdstr` is a string of blank-separated names (mnemonics) associated with array `hdr`, while `obstr` is another string associated with array `obs`. Please note that arrays (`hdr and obs`) have to be defined as REAL*8 arrays. Now let's first learn the usage of subroutine *ufbint* which is called in the following two lines.

```
call ufbint(unit_in,hdr,3,1 ,iret,hdstr)
call ufbint(unit_in,obs,1,10,iret,obstr)
```

- ufbint

```
CALL UFBINT  ( LUBFR, R8ARR, MXMN, MXLV, NLV, CMNSTR )

Input arguments:
    LUBFR     INTEGER      Logical unit for BUFR file
    CMNSTR    CHAR*(*)     String of blank-separated mnemonics
                           associated with R8ARR
    MXMN      INTEGER      Size of first dimension of R8ARR
    MXLV      INTEGER      Size of second dimension of R8ARR
                           OR number of levels of data values
                           to be written to data subset

Input or output argument (depending on context of LUBFR):
    R8ARR(*,*) REAL*8      Data values written/read to/from
                           data subset

Output argument:
    NLV       INTEGER      Number of levels of data values
                           written/read to/from data subset
```

Subroutine UFBINT writes or reads specified values to or from the current BUFR data subset within the internal arrays, with the direction of the data transfer being determined by the context of *LUBFR*, if *LUBFR* points to a BUFR file that is open for input (i.e. reading/decoding BUFR), then data values are read from the internal data subset; otherwise, data values are written to the internal data subset. The actual data transfer occurs through the use of the two-dimensional REAL*8 array *R8ARR* whose actual first dimension *MXMN* must always be passed in. The call argument *MXLV*, on the other hand, contains the actual second dimension of *R8ARR* only when *LUBFR* points to a BUFR file that is open for input (i.e. reading/decoding BUFR); otherwise, whenever *LUBFR* points to a BUFR file that is open for output (i.e. writing/encoding BUFR), *MXLV* instead contains the actual number of levels of data values that are to be written to the data subset (and where this number must be less than or equal to the actual second dimension of *R8ARR*). In either case, the input character string *CMNSTR* always contains a blank-separated list of "mnemonics" which correspond to the REAL*8 values contained within the first dimension of *R8ARR*, and the output argument *NLV* always denotes the actual number of levels of those values that were written/read to/from the second dimension of *R8ARR*, where each such level represents a repetition of the mnemonics within *CMNSTR*. Note that, when *LUBFR* points to a BUFR file that is open for output (i.e.

writing/encoding BUFR), we would certainly expect that the output value *NLV* is equal to the value of *MXLV* that was input, and indeed this is the case unless some type of error occurred in storing one or more of the data levels.

In this case, after we run the two BUFRLIB subroutines, longitude (XOB), latitude (YOB), and observation time (DHR) will be read into array `hdr` and temperature observations (TOB) is read into array `obs`. The array contents should be:

- hdr(1) - longitude
- hdr(2) - latitude
- hdr(3) - time
- obs(1,1) - temperature observation in 1st level (single level)
- obs(1,2) - temperature observation in 2nd level for multi-level observation
- obs(1,3) - temperature observation in 3rd level for multi-level observation
- ...

Because these two lines are inside the message and subset loops, we can get temperature observation with location and time from all observations in the BUFR file. If data subsets contain some missing data, the data values in the array are assigned as 10.0E10.

Now, only one BUFRLIB subroutine *datelen* left in the code needs to be explained:

- *datelen*:

```
CALL DATELEN  ( LEN )
Input argument:
    LEN       INTEGER      Length of Section 1 date-time values to
                           be output by message-reading subroutines
                           such as READMG, READERME, etc.
                             8 =   YYMMDDHH (i.e. 2-digit year)
                            10 = YYYYMMDDHH (i.e. 4-digit year)
```

This subroutine allows the user to specify the format for the IDATE output argument that is returned by READMG.

2.1.2 Encoding/writing data into a simple BUFR file

The following is from the program *bufr_encode_sample.f90*, which shows how to write a few observation variables into a new BUFR file.

```
program bufr_encode_sample
!
!  example of writing one value into a bufr file
!
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,1)

 character(8) subset
 integer :: unit_out=10,unit_table=20
 integer :: idate,iret

! set data values
 hdr(1)=75.;hdr(2)=30.;hdr(3)=-0.1
 obs(1,1)=287.15
 idate=2008120100  ! YYYYMMDDHH
 subset='ADPUPA'   ! upper-air reports

! encode
 open(unit_table,file='table_prepbufr.txt')
 open(unit_out,file='sample.bufr',action='write' &
           ,form='unformatted')
 call datelen(10)
 call openbf(unit_out,'OUT',unit_table)
   call openmb(unit_out,subset,idate)
      call ufbint(unit_out,hdr,3,1,iret,hdstr)
      call ufbint(unit_out,obs,1,1,iret,obstr)
      call writsb(unit_out)
   call closmg(unit_out)
 call closbf(unit_out)

end program
```

Specifically, this example will write one temperature observation value with observation location and time to a BUFR file named as *sample.bufr*.

Here, we can see the BUFR encode procedure has the same structure as the decode procedure: file level, message level, subset level, which are marked in the same color

as the decode example in Section 2.1.1. The major difference between encode and decode are highlighted in bold in the code and explained below:

- **open(unit_table,file='table_prepbufr.txt')**
  To encode some observation values into a new BUFR file, a pre-existing BUFR table file is necessary and needs to be opened.

- open(unit_out,file='sample.bufr',action='**write**',form='unformatted')
  The action in Fortran open command has to be "**write**".

- call openbf(unit_out,**'OUT',unit_table**)
  The second input parameter is set to "OUT" to access a new file for writing. The third parameter is the logical unit of BUFR table file so that BUFR table will be written into BUFR file. Please check the detailed explanation for *openbf* in section 2.1.1.

- **call openmb(unit_out,subset,idate)**

```
CALL OPENMB  ( LUBFR, CSUBSET, IDATE )

Input arguments:
    LUBFR     INTEGER      Logical unit for BUFR file
    CSUBSET   CHAR*(*)     Table A mnemonic for type of BUFR
                           message to be opened
    IDATE     INTEGER      Date-time to be stored within
                           Section 1 of BUFR message
```

  This function opens and initializes a new BUFR message for eventual output to *LUBFR*, using the arguments *CSUBSET* and *IDATE* to indicate the type and time of message to be encoded. It only opens a new message if either *CSUBSET* or *IDATE* has changed, and otherwise will simply return while leaving the existing internal message unchanged, so that subsequent data subsets can be stored within the same internal message. For this reason, *OPENMB* allows for the storage of an increased number of data subsets within each BUFR message and therefore improves overall encoding efficiency. Regardless, whenever a new BUFR message is opened and initialized, the existing internal BUFR message (if any) will be automatically closed and written to output via an internal call to the following subroutine:

- call closmg(unit_out)

```
CALL CLOSEMG ( LUBFR )
Input arguments:
    LUBFR     INTEGER      Logical unit for BUFR file
```

  Furthermore, since, in the case of a BUFR file that was opened for input, each subsequent call to subroutine I*READMG* will likewise automatically clear an existing message from the internal arrays before reading in the new one, for

this reason, it is rare to ever see subroutine *CLOSMG* called directly from within an application program!

- **call writsb(unit_out)**

```
      CALL WRITSB  ( LUBFR )

      Input argument:
          LUBFR    INTEGER      Logical unit for BUFR file
```

This subroutine is called to indicate to the BUFRLIB software that all necessary data values for this subset have been stored and thus that the subset is ready to be encoded and packed into the current message for the BUFR file associated with logical unit *LUBFR*. However, we should note that the BUFRLIB software will not allow any single BUFR message to grow larger than a certain size (usually 10000 bytes, although this can be increased via a call to subroutine *MAXOUT*);

Before this subroutine, we can see two consecutive calls to the subroutine `ufbint`, which is the same as in the decode example. However, this time, the strings `hdstr` tells the BUFR subroutine `ufbint` that the array `hdr` holds longitude, latitude and observation time, the string `obstr` tells `ufbint` that the array `obs` holds temperature observations. The data subset is ready and written into the BUFR file via call **`writsb`**.

2.1.3 Appending data to a simple BUFR file

The following is from the program *bufr_append_sample.f90*, which shows how to append a new observation variable into an existing BUFR file.

```
program
! sample of appending one observation into bufr file
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,1)

 character(8) subset
 integer :: unit_out=10,unit_table=20
 integer :: idate,iret

! set data values
 hdr(1)=85.0;hdr(2)=50.0;hdr(3)=0.2
 obs(1,1)=300.0
 idate=2008120101  ! YYYYMMDDHH
 subset='ADPSFC'   ! surface land reports

! get bufr table from existing bufr file
 open(unit_table,file='table_prepbufr_app.txt')

open(unit_out,file='sample.bufr',status='old',form='unformatted')
 call openbf(unit_out,'IN',unit_out)
 call dxdump(unit_out,unit_table)
 call closbf(unit_out)

! append

open(unit_out,file='sample.bufr',status='old',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'APN',unit_table)
   call openmb(unit_out,subset,idate)
      call ufbint(unit_out,hdr,3,1,iret,hdstr)
      call ufbint(unit_out,obs,1,1,iret,obstr)
      call writsb(unit_out)
   call closmg(unit_out)
 call closbf(unit_out)

end program
```

Specifically, this example will append one temperature observation value with observation location and time to an existing BUFR file named as *sample.bufr*.

If we compare this code with the example code for encoding, we can find the code structure and BUFRLIB functions used are very similar in two codes. But there is a key point that needs special attention for appending:

- Appending has to use the exact same BUFR table as the existing BUFR file. To ensure this, we add the following three lines to the code in order to extract the BUFR table from the existing BUFR file:

```
call openbf(unit_out,'IN',unit_out)
call dxdump(unit_out,unit_table)
call closbf(unit_out)
```

Let's learn subroutine dxdump.

```
CALL DXDUMP  ( LUBFR, LDXOT )

Input arguments:
    LUBFR INTEGER        Logical unit for BUFR file
    LDXOT INTEGER        Logical unit for output BUFR tables file
```

This subroutine provides a handy way to view the BUFR table information that is embedded in the first few messages of a BUFR file. The user needs only to have identified the file to the BUFRLIB software via a prior call to subroutine *OPENBF*, and then a subsequent call to subroutine *DXDUMP* will unpack the embedded tables information and write it out to the file pointed to by logical unit *LDXOT*. The output file is written with ASCII-text table format. Subroutine *DXDUMP* can be most useful for learning the contents of archive BUFR files.

In this example, the BUFR table embedded in the BUFR file *sample.bufr* will be read in and written into a text file called *table_prepbufr_app.txt.*

Comparing with the encode example again, there are two more slight differences in setups, which are highlighted in the code as Bold and explained below:

- In the Fortran open command, the status has to be set as **'old'** because appending requires an existing BUFR file.
- In the subroutine *openbf*, the existing BUFR file and dumped BUFR table are connected to BUFRLIB, the second input parameter has to be set as '**APN**'.

2.2 Encode, Decode, Append the PrepBUFR file

In last section, we use three simplified examples to illustrate the code structure of the BUFR file process (read, write and append) and explained commonly used BUFRLIB functions in the example code. In this section, we will learn how to use the skills we learned in previous sections to process a PrepBUFR file, which is one of major BUFR files used in GSI for all conventional observations and retrieved standard observations.

2.2.1 Decoding/reading data from a PrepBUFR file

The following is from the code *prepbufr_decode_all.f90*, which reads all major conventional observations and BUFR table out from a PrepBUFR file.

```
program prepbufr_decode_all
!
! read all observations out from prepbufr.
! read bufr table from prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=250
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
```

Compared to the mnemonic list used in the examples in 2.1, a clear difference here is that more BUFR table mnemonics are involved because we want to read all major observations, such as temperature (TOB), moisture (QOB), Pressure(POB), Height (ZOB), wind (UOB and VOB). Also, we want to read the quality flags and observation errors with these observations at the same time. Here is a list of content in these mnemonics strings:

- hdstr: defines report header information including the station ID, longitude, latitude, time, report type, elevation, satellite ID, data dump report type.
- `obstr`: defines observation for pressure, specific humidity, temperature, height, u and v component of wind, total precipitable water, data level category, surface pressure.
- `qcstr`: defines the quality markers for each of observation variables listed in the string `obstr`.
- `oestr`: defines the observation error for each of observation variables listed in the string `obstr`.

More detailed information on these mnemonics can be found from the BUFR table named with "*prepobs_prep.bufrtable*", which is a text file dumped out during the decoding process.

```
real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)
```

The associated arrays are defined to hold the data values of mnemonics specified in `hdstr, obstr, qcstr, oestr`. Note, `mxmn`=35, `mxlv`=250, which make the array can hold up to 250 levels of observations with up to 35 mnemonics in each level.

```
INTEGER         :: ireadmg,ireadsb
character(8)    :: subset
integer         :: unit_in=10,unit_table=24,idate,nmsg,ntb


character(8)    :: c_sid
real(8)         :: rstation_id
equivalence(rstation_id,c_sid)
```

From our earlier discussions, it was noted that data values are normally read from or written to BUFR subsets using REAL*8 arrays via subroutine. The character values are read and written in the same way using a REAL*8 variable. Here, `rstation_id` is real(8); `c_sid` is character(8); then FORTRAN EQUIVALENCE is used to covert the station ID from REAL*8 to string that can be easily read by humans.

```
integer         :: i,k,iret


open(unit_table,file='prepobs_prep.bufrtable')
```

Fortran open command to link BUFR table with a logical unit, `unit_table`.

```
open(unit_in,file='prepbufr',form='unformatted',status='old')
```

Fortran open command to link a PrepBUFR file with a logical unit, `unit_in`.

```
call openbf(unit_in,'IN',unit_in)
```

Connect the PrepBUFR file to BUFRLIB. Since BUFR table is embedded in the PrepBUFR file, the third argument is the same as first argument in this call.

```
call dxdump(unit_in,unit_table)
```

Dump BUFR table out from the existing PrepBUFR file and write to a ASCII file named "*prepobs_prep.bufrtable*" through unit `unit_table`.

```
call datelen(10)
```

Specifies the date format as YYYYMMDDHH.

```
nmsg=0
   msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
     nmsg=nmsg+1
     ntb = 0
     write(*,*)
     write(*,'(3a,i10)') 'subset=',subset,' cycle time =',idate
     sb_report: do while (ireadsb(unit_in) == 0)
```

The `msg_report` loop reads each of messages until reaching the end of file. The `sb_report` loop reads each of data subsets within the current message until the end of the message.

```
        ntb = ntb+1
        call ufbint(unit_in,hdr,mxmn,1   ,iret,hdstr)
        call ufbint(unit_in,obs,mxmn,mxlv,iret,obstr)
        call ufbint(unit_in,oer,mxmn,mxlv,iret,oestr)
        call ufbint(unit_in,qcf,mxmn,mxlv,iret,qcstr)
```

Calling rhe subroutine *ufbint* to read data based on mnemonics defined in `hdstr, obstr, oestr, qcstr` from a subset and write to corresponding arrays `hdr,obs, oer, qcf`. The `iret` is the actual returned number of pressure levels which have be read in even though `mxlv`=250.

```
        rstation_id=hdr(1)
        write(*,*)
        write(*,'(2I10,a14,8f14.1)') ntb,iret,c_sid,(hdr(i),i=2,8)

       DO k=1,iret
          write(*,'(i3,a10,9f14.1)') k,'obs=',(obs(i,k),i=1,9)
          write(*,'(i3,a10,9f14.1)') k,'oer=',(oer(i,k),i=1,7)
          write(*,'(i3,a10,9f14.1)') k,'qcf=',(qcf(i,k),i=1,7)
        ENDDO

     enddo sb_report
   enddo msg_report

call closbf(unit_in)
end program
```

From this PrepBUFR decoding example, we can see that the code structure and functions used are the same as the simple decoding example in section 2.1. But this example defines more mnemonics and larger dimensions of the REAL*8 arrays to read all major observation elements from the PrepBUFR file, including observation values, quality markers, and observation errors.

2.2.2 Encoding/Writing surface data into a PrepBUFR file

The following is from the code *prepbufr_encode_surface.f90*, which writes a surface observation into a PrepBUFR file. Let's focus only on the differences compared with *prepbufr_decode_all.f90.*

```
program prepbufr_encode_surface
!
!  write a surface observation into prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=1
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '

 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)
```

The level parameter `mxlv`=1 since surface observation is single level data.

```
 character(8) :: subset
 integer      :: unit_out=10,unit_table=20,idate,iret

 character(8) :: c_sid
 real(8)      :: rstation_id
 equivalence(rstation_id,c_sid)

 ! write observation into prepbufr file
 !
 open(unit_table,file='prepobs_prep.bufrtable',action='read')
 open(unit_out,file='prepbufr',action='write',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'OUT',unit_table)
```

Connect the `unit_out` and `unit_table` to BUFRLIB. Here, the BUFR table is needed because of encoding and will be written into the PrepBUFR file. The parameter `'OUT'` tells BUFRLIB to access a new file for writing.

```
   idate=2010050700 ! cycle time: YYYYMMDDHH
   subset='ADPSFC'  ! surface land (SYNOPTIC, METAR) reports
   call openmb(unit_out,subset,idate)
```

Opens and initializes a new BUFR message for writing to the PrepBUFR file, using "*ADPSFC*" and "*2010050700*" as message type and analysis time. "*ADPSFC*" is the surface land report.

```
! set headers
      hdr=10.0e10
```

Initialize report header array. 10.0e10 is the missing value in the PrepBUFR file.

```
c_sid='KTKI'; hdr(1)=rstation_id
      hdr(2)=263.4; hdr(3)=33.2; hdr(4)=-0.1; hdr(6)=179.0

! set obs, qcf, oer for  wind
      hdr(5)=287          ! report type
```

Set up report header values. `hdr(5)=287` in which 287 is one of surface wind report types for surface message type "ADPSFC".

```
      obs=10.0e10;qcf=10.0e10;oer=10.0e10
      obs(1,1)=985.2; obs(5,1)=-2.8; obs(6,1)=-7.7; obs(8,1)=6.0
      qcf(1,1)=2.0  ; qcf(5,1)=2.0
      oer(5,1)=1.6
```

Assign observation values, quality markers, and errors for this wind report.

```
! encode  wind obs
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,mxlv,iret,obstr)
      call ufbint(unit_out,oer,mxmn,mxlv,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,mxlv,iret,qcstr)
      call writsb(unit_out)
```

Using `writsb` to tell BUFRLIB that data subset is ready and can be written to the PrepBUFR file. Here, we had written a wind surface observation into the PrepBUFR file. Next, let's write a surface mass observation (temperature, moisture …):

```
! set obs, qcf, oer for  temperature and moisture
      hdr(5)=187          ! report type
```

`hdr(5)=187` in which 187 is one of the surface mass report type for surface message "*ADPSFC*". Here we can see NCEP classifies observations into many observation types with unique numbers for each observation type. Also, NCEP use number 100-199 for mass observations and 200-299 for wind observations. For the observations from the same station, the mass observations go to 100-199, for example 187 in this example, while the wind observations from the same station go to 200-299 with the same last 2 digital number, for example, 287 here.

```
      obs=10.0e10;qcf=10.0e10;oer=10.0e10
      obs(1,1)=985.2;obs(2,1)=12968.0;obs(3,1)=31.3
      obs(4,1)=179.0;obs(8,1)=0.0
      qcf(1,1)=2.0  ;qcf(2,1)=2.0    ;qcf(3,1)=2.0 ;qcf(4,1)=2.0
      oer(1,1)=0.5  ;oer(2,1)=0.6    ;oer(3,1)=2.3
```

Assign mass report observation, quality markers and error.

```
! encode temperature and moisture
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,mxlv,iret,obstr)
      call ufbint(unit_out,oer,mxmn,mxlv,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,mxlv,iret,qcstr)
      call writsb(unit_out)
```

Again, write the mass observations into a subset to the PrepBUFR file.

```
 call closmg(unit_out)
```

Close the current message and write the current message into the PrepBUFR file.

```
 call closbf(unit_out)

end program
```

2.2.3 Encoding/Writing upper air data into a PrepBUFR file

The following is from the code *prepbufr_encode_upperair.f90*, which writes an upper air observation into PrepBUFR file. Let's look at the differences compared with *prepbufr_encode_surface.f90*.

```
program prepbufr_encode_upperair
!  write a upper air observation into prepbufr file
implicit none

 integer, parameter :: mxmn=35, mxlv=200
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)
 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)
```

These arrays are defined with second dimension `mxlv=200` to hold multiple level upper air data subsets.

```
 character(8) :: subset
 integer      :: unit_out=10,unit_table=20,idate,iret,nlvl
 character(8) :: c_sid
 real(8)      :: rstation_id
 equivalence(rstation_id,c_sid)
```

```
! write observation into prepbufr file
 open(unit_table,file='prepobs_prep.bufrtable',action='read')
 open(unit_out,file='prepbufr',action='write',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'OUT',unit_table)

 idate=2010050700 ! cycle time: YYYYMMDDHH
 subset='ADPUPA'  ! upper-air (raob, drops) reports
 call openmb(unit_out,subset,idate)
```

Opens and initializes a new BUFR message for writing to PrepBUFR file with message type "*ADPUPA*", which is upper air report.

```
! set headers
  hdr=10.0e10
  c_sid='72293'; hdr(1)=rstation_id; hdr(2)=242.9; hdr(3)=32.9
  hdr(4)=0.0; hdr(6)=134.0

! set obs, qcf, oer for  wind
  hdr(5)=220          ! report type: sounding
```

Wind report type for message "ADPUPA" is 220, which is soundings.

```
  obs=10.0e10; qcf=10.0e10; oer=10.0e10
  obs(1,1)=998.0; obs(5,1)=4.6 ; obs(6,1)=2.2;  obs(8,1)=3.0;
  qcf(1,1)=2.0; qcf(5,1)=2.0;
  oer(5,1)=2.3

  obs(1,2)=850.0; obs(5,2)=2.0 ; obs(6,2)=-1.7;obs(8,2)=1.0;
  qcf(1,2)=2.0; qcf(5,2)=2.0;
  oer(5,2)=2.6

  obs(1,3)=700.0; obs(5,3)=12.1;obs(6,3)=-4.4;obs(8,3)=1.0;
  qcf(1,3)=2.0;
  qcf(5,3)=2.0;
  oer(5,3)=2.5
```

Assign wind report observation values, quality markers, and errors for three pressure levels: 998.0, 850.0,700.0.

```
  nlvl=3
! encode  wind obs
  call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
  call ufbint(unit_out,obs,mxmn,nlvl,iret,obstr)
  call ufbint(unit_out,oer,mxmn,nlvl,iret,oestr)
  call ufbint(unit_out,qcf,mxmn,nlvl,iret,qcstr)
  call writsb(unit_out)
```

Write wind report data subset to the PrepBUFR file. Note `nlvl=3` which is the actual number of pressure levels and represents three repetition of the mnemonics defined in `obstr, oestr, qcstr`.

So far, we have dealt with the wind observations. Now we start with mass observations from the same station below:

```
! set obs, qcf, oer for  temperature and moisture
  hdr(5)=120          ! report type: sounding
```

Set up mass report type for message "ADPUPA" as 120.

```
obs=10.0e10; qcf=10.0e10; oer=10.0e10
obs(1,1)=998.0; obs(2,1)=8112.0; obs(3,1)=22.3; obs(4,1)=134.0; obs(8,1)=0.0
qcf(1,1)=2.0; qcf(2,1)=2.0; qcf(3,1)=2.0; qcf(4,1)=2.0;
oer(1,1)=0.7; oer(2,1)=0.7; oer(3,1)=1.4

obs(1,2)=925.0; obs(2,2)=6312.0; obs(3,2)=14.1; obs(4,2)=779.0; obs(8,2)=1.0
qcf(1,2)=2.0; qcf(2,2)=2.0; qcf(3,2)=2.0; qcf(4,2)=2.0;
oer(2,2)=0.9; oer(3,2)=1.5

obs(1,3)=850.0; obs(2,3)=2161.0; obs(3,3)=14.8; obs(4,3)=1493.; obs(8,3)=1.0
qcf(1,3)=2.0; qcf(2,3)=2.0; qcf(3,3)=2.0; qcf(4,3)=2.0;
oer(2,3)=1.1; oer(3,3)=1.4

obs(1,4)=700.0; obs(2,4)=2131.0; obs(3,4)=9.2; obs(4,4)=3118.; obs(8,4)=1.0
qcf(1,4)=2.0; qcf(2,4)=2.0; qcf(3,4)=2.0; qcf(4,4)=2.0;
oer(2,4)=1.4; oer(3,4)=1.0;
```

Assign mass report observation values, quality markers, and errors for four pressure levels: 998.0, 925.0, 850.0, 700.0. Only the lowest pressure level has observation error denoted by `oer(1,1)=0.7.` For the others, missing value 10.0e10 for pressure observation error are filled in the PrepBUFR file. Please note that missing pressure observation error in level 2-4 will not impact the regional data analysis because the observation errors in regional GSI analysis are from the error table read in from a fixed file directory.

```
nlvl=4
! encode temperature and moisture
  call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
  call ufbint(unit_out,obs,mxmn,nlvl,iret,obstr)
  call ufbint(unit_out,oer,mxmn,nlvl,iret,oestr)
  call ufbint(unit_out,qcf,mxmn,nlvl,iret,qcstr)
  call writsb(unit_out)
```

Write mass report data subset to PrepBUFR file. Note `nlvl=4.`

```
  call closmg(unit_out)
  call closbf(unit_out)
end program
```

2.2.4 Appending surface data into a PrepBUFR file

The following is from the code *prepbufr_append_surface.f90*, which appends a surface observation into an existing PrepBUFR file. Let's again look at the differences compared with *prepbufr_encode_surface.f90*.

```
program prepbufr_append_surface
!
!  append a surface observation into prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=1
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)

 character(8) :: subset
 integer      :: unit_out=10,unit_table=20,idate,iret

 character(8) :: c_sid
 real(8)      :: rstation_id
 equivalence(rstation_id,c_sid)

! get bufr table from existing bufr file
 open(unit_table,file='prepobs_prep_app.bufrtable')
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call openbf(unit_out,'IN',unit_out)
 call dxdump(unit_out,unit_table)
 call closbf(unit_out)
```

Dump BUFR table out from the PrepBUFR file and write the table to a text file specified by `unit_table`. This BUFR table will be used later in this example to append the surface observation. This step is to make sure the BUFR table used in appending is the same BUFR table as the existing BUFR file. Please refer to the appending case in section 2.1 for more details of the BUFR table requirement for the appending.

```
!
! write observation into prepbufr file
!
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'APN',unit_table)
```

Connect the same BUFR table dumped from the existing PrepBUFR file to BUFRLIB. The `'APN'` tells BUFRLIB to append the data subset into the existing PrepBUFR file.

The rest of the code has the same logic as *prepbufr_encode_surface.f90* so we will not explain them anymore.

```
   idate=2010050700 ! cycle time: YYYYMMDDHH
   subset='ADPSFC'  ! surface land (SYNOPTIC, METAR) reports
   call openmb(unit_out,subset,idate)

! set headers
      hdr=10.0e10
      c_sid='72408'; hdr(1)=rstation_id
      hdr(2)=284.8; hdr(3)=39.9; hdr(4)=-0.1; hdr(6)=9.0

! set obs, qcf, oer for  wind
      hdr(5)=281          ! report type
      obs=10.0e10;qcf=10.0e10;oer=10.0e10
      obs(1,1)=1008.6; obs(5,1)=4.0; obs(6,1)=-4.7; obs(8,1)=6.0
      qcf(1,1)=2.0;    qcf(5,1)=2.0
      oer(5,1)=1.6
! encode  wind obs
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,mxlv,iret,obstr)
      call ufbint(unit_out,oer,mxmn,mxlv,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,mxlv,iret,qcstr)
      call writsb(unit_out)

! set obs, qcf, oer for  temperature and moisture
      hdr(5)=181          ! report type
      obs=10.0e10;qcf=10.0e10;oer=10.0e10

obs(1,1)=1008.6;obs(2,1)=4925.0;obs(3,1)=25.1;obs(4,1)=9.0;obs(8,1)=0.0
      qcf(1,1)=2.0;   qcf(2,1)=2.0;   qcf(3,1)=2.0; qcf(4,1)=2.0
      oer(1,1)=0.5;   oer(2,1)=0.6;   oer(3,1)=1.5
! encode temperature and moisture
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,mxlv,iret,obstr)
      call ufbint(unit_out,oer,mxmn,mxlv,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,mxlv,iret,qcstr)
      call writsb(unit_out)

   call closmg(unit_out)
 call closbf(unit_out)

end program
```

2.2.5 Appending upper air data into a PrepBUFR file

The following is from the code *prepbufr_append_upperair.f90*, which appends an upper air observation into an existing PrepBUFR file. Let's once again focus on the differences compared with *prepbufr_encode_upperair.f90.*

```
program prepbufr_append_upperair
!
!  append a upper air observation into prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=200
```

```
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)

 character(8) :: subset
 integer      :: unit_out=10,unit_table=20,idate,iret,nlvl

 character(8) :: c_sid
 real(8)      :: rstation_id
 equivalence(rstation_id,c_sid)

! get bufr table from existing bufr file
 open(unit_table,file='prepobs_prep_app.bufrtable')
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call openbf(unit_out,'IN',unit_out)
 call dxdump(unit_out,unit_table)
 call closbf(unit_out)
```

These arrays are defined with second dimension `mxlv=200` to hold multiple level upper air data subsets. Same as previous example, dump BUFR table out from the PrepBUFR file and write the table to file specified by `unit_table`.

```
!
! write observation into prepbufr file
!
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'APN',unit_table)
```

Connect the same table dumped from the existing PrepBUFR file to BUFRLIB. The `'APN'` tells BUFRLIB to append the data subset into the existing PrepBUFR file. The rest of the code has the same logic as *prepbufr_encode_upperair.f90* so we will not explain then anymore.

```
   idate=2010050700 ! cycle time: YYYYMMDDHH
   subset='ADPUPA'  ! upper-air (raob, drops) reports
   call openmb(unit_out,subset,idate)

! set headers
      hdr=10.0e10
      c_sid='71823'; hdr(1)=rstation_id
      hdr(2)=286.3; hdr(3)=53.8; hdr(4)=0.0; hdr(6)=307.0

! set obs, qcf, oer for  wind
      hdr(5)=220          ! report type: sounding
      obs=10.0e10; qcf=10.0e10; oer=10.0e10
      obs(1,1)=500.0; obs(5,1)=-2.0; obs(6,1)=0.7; obs(8,1)=1.0
      qcf(1,1)=2.0;   qcf(5,1)=2.0
                      oer(5,1)=2.5

     obs(1,2)=432.5; obs(4,2)=6401; obs(5,2)=-7.1; obs(6,2)=-1.2; obs(8,2)=4.0
      qcf(1,2)=2.0;   qcf(4,2)=2.0;  qcf(5,2)= 2.0
                                     oer(5,2)=2.6
```

```
      nlvl=2
! encode  wind obs
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,nlvl,iret,obstr)
      call ufbint(unit_out,oer,mxmn,nlvl,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,nlvl,iret,qcstr)
      call writsb(unit_out)

! set obs, qcf, oer for  temperature and moisture
      hdr(5)=120          ! report type: sounding
      obs=10.0e10; qcf=10.0e10; oer=10.0e10
      obs(1,1)=825.0; obs(2,1)=3672.0; obs(3,1)=0.8; obs(8,1)=2.0
      qcf(1,1)=2.0;   qcf(2,1)=2.0;    qcf(3,1)=2.0;
                      oer(2,1)=1.2;    oer(3,1)=1.3

obs(1,2)=700.0;obs(2,2)=1157.0;obs(3,2)=-7.3;obs(4,2)=2841.;obs(8,2)=1.0
qcf(1,2)=2.0;  qcf(2,2)=2.0;   qcf(3,2)=2.0; qcf(4,2)=2.0
               oer(2,2)=1.4;   oer(3,2)=1.0

      obs(1,3)=623.0; obs(2,3)=254.0; obs(3,3)=-12.9; obs(8,3)=2.0
      qcf(1,3)=2.0;   qcf(2,3)=2.0;   qcf(3,3)=2.0
                      oer(2,3)=1.5;   oer(3,3)=1.0
      nlvl=3

! encode temperature and moisture
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,nlvl,iret,obstr)
      call ufbint(unit_out,oer,mxmn,nlvl,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,nlvl,iret,qcstr)
      call writsb(unit_out)

   call closmg(unit_out)
 call closbf(unit_out)

end program
```

The data type is 220 for wind and 120 for mass observation for upper air data type.

2.2.6 Appending retrieve data into a PrepBUFR file

The following is from the code *prepbufr_append_retrieve.f90*, which appends a retrieved data into an existing PrepBUFR file. Compared with *prepbufr_append_surface.f90,* this example is simpler since it appends only the wind report into the existing PrepBUFR file. Let's examine only the differences.

```
program prepbufr_append_retrieve
!
!  append a retrieved data into prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=200
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
```

```
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
 real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)

 character(8) :: subset
 integer      :: unit_out=10,unit_table=20,idate,iret,nlvl

 character(8) :: c_sid
 real(8)      :: rstation_id
 equivalence(rstation_id,c_sid)

! get bufr table from existing bufr file
 open(unit_table,file='prepobs_prep_app.bufrtable')
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call openbf(unit_out,'IN',unit_out)
 call dxdump(unit_out,unit_table)
 call closbf(unit_out)
!
! write observation into prepbufr file
!
 open(unit_out,file='prepbufr',status='old',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'APN',unit_table)

   idate=2010050700 ! cycle time: YYYYMMDDHH
   subset='SATWND'  ! upper-air (raob, drops) reports
```

Message type “SATWND” is the satellite-derived wind reports. For this reason, only wind report is appended in this example.

```
 call openmb(unit_out,subset,idate)

! set headers
      hdr=10.0e10
      c_sid='A114424Z'; hdr(1)=rstation_id
      hdr(2)=199.6; hdr(3)=13.9; hdr(4)=-1.0; hdr(6)=934.0; hdr(7)=255.0

! set obs, qcf, oer for  wind
      hdr(5)=251          ! report type: NESDIS VISIBLE CLOUD DRIFT
                          !             (ALL LEVELS) (GOES) - u, v
```

Wind report type for “SATWND” is 251.

```
      obs=10.0e10;qcf=10.0e10;oer=10.0e10
      obs(1,1)=906.0;obs(4,1)=934.0;obs(5,1)=-11.4;obs(6,1)=-
3.3;obs(8,1)=6.0
      qcf(1,1)=2.0  ;qcf(4,1)=2.0  ;qcf(5,1)=1.0
                                    oer(5,1)=3.8
      nlvl=1
! encode  wind obs
      call ufbint(unit_out,hdr,mxmn,1   ,iret,hdstr)
      call ufbint(unit_out,obs,mxmn,nlvl,iret,obstr)
      call ufbint(unit_out,oer,mxmn,nlvl,iret,oestr)
      call ufbint(unit_out,qcf,mxmn,nlvl,iret,qcstr)
      call writsb(unit_out)

   call closmg(unit_out)
 call closbf(unit_out)

end program
```

2.3 Decoding/reading radiance data

The following is from the code *bufr_decode_radiance.f90*, which read TOVS 1b radiance observations from radiance BUFR files. Compared with *prepbufr_decode_all.f90,* radiance report header and observation information are different from the conventional PrepBUFR report. This example also introduces the new subroutine *ufbrep* and code to inventory satellite type information.

```
program bufr_decode_radiance

! read all radaince observations out from bufr.
! read bufr table from prepbufr file

 implicit none

 integer, parameter :: mxmn=35, mxlv=250
 character(80):: hdstr= &
   'SAID FOVN YEAR MNTH DAYS HOUR MINU SECO CLAT CLON CLATH CLONH HOLS'
 character(80):: hdr2b='SAZA SOZA BEARAZ SOLAZI'
 character(80):: obstr='TMBR'
```

Two strings (`hdstr, hdr2b`) are used to define report header information. The detailed information for these header mnemonics can be found in the dumped BUFR table for radiance. The `obstr` defines observation 'TMBR' which is the brightness temperature.

```
 real(8) :: hdr(mxmn),hdr2(mxmn),obs(mxlv)

 INTEGER        :: ireadmg,ireadsb
 character(8)   :: subset
 integer        :: unit_in=10,unit_table=24,idate,nmsg,ntb
 integer,parameter:: max_sat_type=20
 integer        :: nsat_type(max_sat_type),nsat_num(max_sat_type)
 integer        :: i,k,iret,ksatid,nchanl,num_sat_type,ii

 nchanl=15
 nsat_num=0
 nsat_type=0

 open(unit_table,file='radiance.bufrtable')
 open(unit_in,file='1bamua',form='unformatted',status='old')
 call openbf(unit_in,'IN',unit_in)
 call dxdump(unit_in,unit_table)
 call datelen(10)

 nmsg=0
 ntb = 0
 num_sat_type = 0
 msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
     nmsg=nmsg+1
     write(*,*)
     write(*,'(3a,i10)') 'subset=',subset,' cycle time =',idate
```

```
sb_report: do while (ireadsb(unit_in) == 0)
  ntb = ntb+1
  call ufbint(unit_in,hdr ,mxmn,1  ,iret,hdstr)
  call ufbint(unit_in,hdr2,mxmn,1  ,iret,hdr2b)
```

The same structure is used here as in all other decoding code, using *ireadmg* and *ireadsb* to read all messages and all subsets in a message out. Two ufbint calls get the report header information and write to array `hdr` and `hdr2`.

```
call ufbrep(unit_in,obs ,1   ,nchanl,iret,obstr)
```

Call *ufbrep* to get brightness temperature and write to array `obs`. Here *ufbrep* is a new function and is explained below:

- ufbrep

```
UFBREP  ( LUBFR, R8ARR, MXMN, MXLV, NLV, CMNSTR )

      Input arguments:
          LUBFR INTEGER     Logical unit for BUFR file
          CMNSTR      CHAR*(*)    String of blank-separated mnemonics
                            associated with R8ARR
          MXMN  INTEGER     Size of first dimension of R8ARR
          MXLV  INTEGER     Size of second dimension of R8ARR
                                  OR number of levels of data values
                                  to be written to data subset

      Input or output argument (depending on context of LUBFR):
          R8ARR(*,*) REAL*8 Data values written/read to/from
                                  data subset

      Output argument:
          NLV   INTEGER     Number of levels of data values
                            written/read to/from data subset
```

ufbrep and *ufbint* are similar. We only specify the difference here. *UFBINT* is used for writing/reading data values corresponding to mnemonics, which are part of a delayed-replication sequence, or for which there is no replication at all. As such, it is the most commonly-used for many basic applications. *UFBREP*, on the other hand, must be used for mnemonics which are part of a regular (i.e. non-delayed) replication sequence or for those which are replicated via being directly listed more than once within an overall subset definition rather than by being included within a replication sequence.

From the BUFR table, we can see the mnemonic `'TMBR'` is regular replication seqeence. We will explain the definition of replication sequence in detail in Chapter 3.

```
ksatid=nint(hdr(1))
write(*,*)
write(*,'(2I10,I14,13f8.1,)') ntb,iret,ksatid,(hdr(i),i=3,10),hdr(13), &
                              (hdr2(i),i=1,4)
write(*,'(a10,15f7.1)') 'obs=',(obs(i),i=1,iret)
```

One TOVS 1b radiance BUFR file could include observations from many satellite. The following chunk of code illustrates how to find which satellite observations are available in the file and the number of the observations:

```
! satellite type inventory
       if(num_sat_type == 0 ) then
          num_sat_type=1
          nsat_type(num_sat_type)=ksatid
          nsat_num(num_sat_type)= 1
       else
          ii=0
          DO i=1,num_sat_type
             if(nsat_type(i) == ksatid) ii=i
          ENDDO
          if( ii > 0 .and. ii <=num_sat_type) then
             nsat_num(ii)=nsat_num(ii) + 1
          else
             num_sat_type=num_sat_type+1
             if(num_sat_type > max_sat_type) then
                write(*,*) 'Error: too many satellite types'
                write(*,*) 'Need to increase max_sat_type'
                stop 1234
             endif
             nsat_type(num_sat_type)=ksatid
             nsat_num(num_sat_type)=1
          endif
       endif
```

This if … else …endif calculates the satellite inventory information which includes the satellite types (`nsat_type`), the number of satellites in the file (`num_sat_type`), the number of data subsets (`nsat_num`) for each of satellite. It is helpful for us to know what observations are saved in the radiance BUFR file.

```
     enddo sb_report
   enddo msg_report
 call closbf(unit_in)

 write(*,*) 'message=',nmsg,'  subset=',ntb
 DO i=1,num_sat_type
   write(*,'(i4,2i10)') i,nsat_type(i),nsat_num(i)
 ENDDO
end program
```

Chapter 3 DX BUFR table

BUFR is a Self-descriptive Table Driving Form in which BUFR table plays a central role in the BUFR file processing. It is difficult to see how the data and their descriptors (pointer to the BUFR table elements) are saved in the BUFR file as a stream of binary, but we can easily read the BUFR table to find the content of a BUFR file and pick the elements in the BUFR file to encode/decode the data in/out of the file. For NCEP BUFR, The following link gives a very good description of NCEP BUFR table, which is called DX BUFR table (DX stands for dictionary):

http://www.nco.ncep.noaa.gov/sib/decoders/BUFRLIB/toc/dfbftab/

Users can read the above link to learn DX BUFR table and create their own table file. Again, in this Chapter, we will use several simplified examples to introduce the DX BUFR table based on the descriptions of the above NCEP website.

3.1 Description of DX BUFR tables

As noted during the discussion of subroutine *OPENBF*, every BUFR file that is presented to the BUFRLIB software must have DX BUFR tables associated with it, unless the 'SEC3' decoding option is specified during the call to *OPENBF*. In the case of an existing BUFR file, the DX table information may be embedded within the first few BUFR messages of the file itself. Otherwise, a separate ASCII text file containing the necessary DX table information must be supplied.

To understand the contents of a DX BUFR table, it is better that we start from a simplified DX BUFR table example. In Chapter 2, the NCEP DX BUFR table named as "*prepobs_prep.bufrtable*", which is located in the released GSI version 3 package under ./util/bufr_tools directory, is used in many code examples. In Figure 3.1, we provide a simplified DX BUFR table to help us explain the concepts of BUFR table.

3.1.1 WMO BUFR tables and DX BUFR table sections

In the section 2 of Chapter 1, we introduced WMO BUFR tables A, B, C, and D, as well as the flag table, code table, and the BUFR table descriptor. As illustrated in the figure 3.1, a DX BUFR table includes WMO BUFR table A, B, and D and consists of three distinct sections. In the first section, all Table A, B and D mnemonics are initially declared, assigned a unique FXY number (descriptor), and given a short free-form text description. Then, in the second section, all previously declared Table A and Table D mnemonics are actually defined as a sequence of one or more Table B (or other Table D!) mnemonics. Finally, in the third section, all previously declared Table B mnemonics are defined in terms of their scale factor, reference value, bit width, and units.

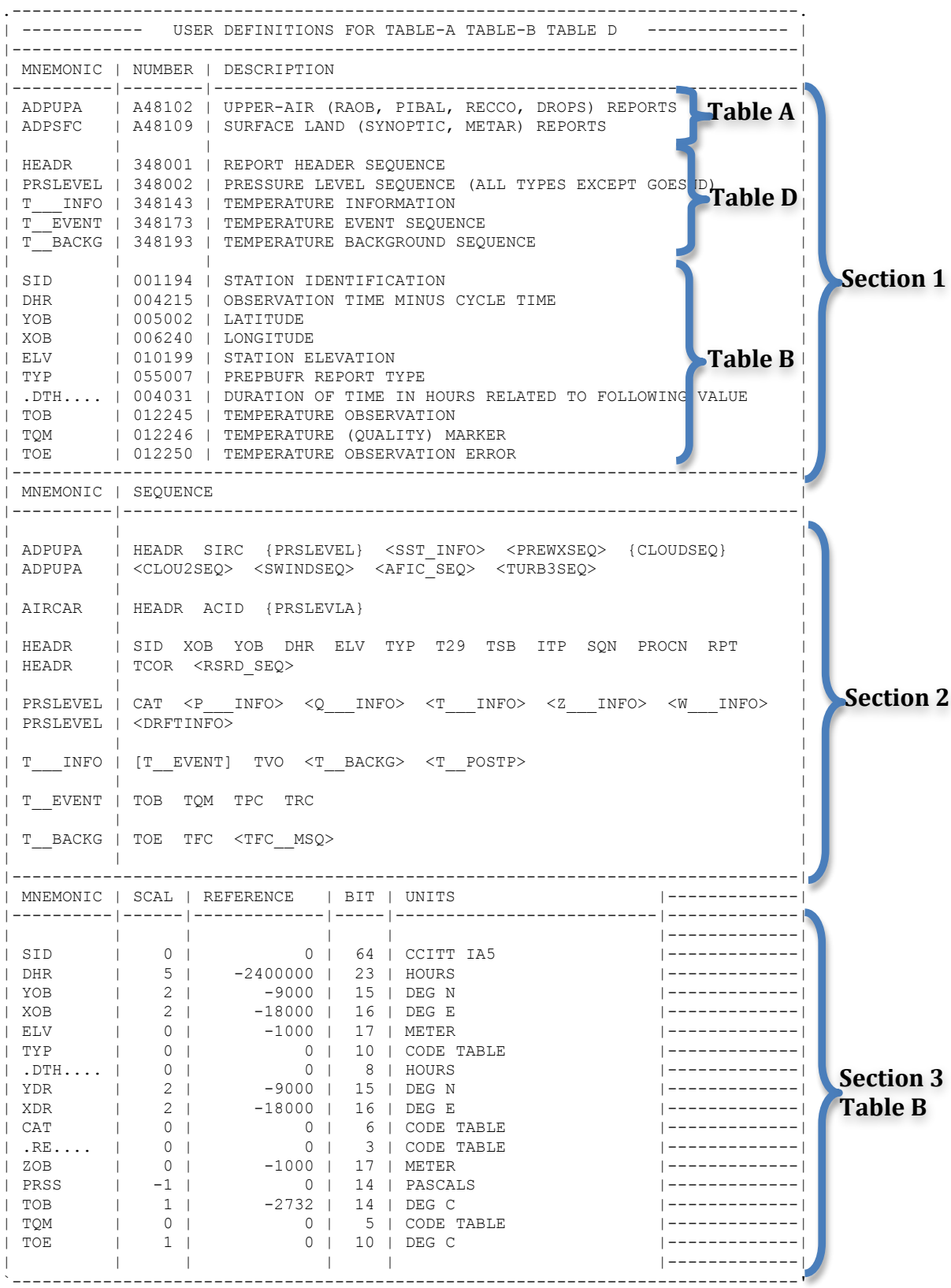


Figure 3.1. Sample of the DX BUFE table and its sections

It may be difficult, at this moment, to understand the above DX BUFR table structure and its relation with the WMO BUFR tables if users skipped section 2 of Chapter 1. Here, we will explain several important BUFR concepts that were introduced in Chapter 1 to help users better understand the DX BUFR table.

In Section 1 of DX BUFR table, the 2nd column defines a NUMBER for each element. This number calls FXY number or Descriptor. In BUFR file, it is this number that is saved in the Data Description Section (BUFR file Section 3) to describe what kind of data is saved in the Data Section (BUFR file Section 4) of the BUFR file. In the WMO BUFR table, the FXY number (or Descriptor) is used as a pointer to link the data to their BUFR table descriptions defined in tables B and D. But in DX BUFR table and the BUFRLIB code, the **mnemonic** has the same function of FXY number. So it is important to first understand the BUFR Descriptors and Mnemonic to better understand BUFR tables:

BUFR Descriptors

A BUFR descriptor is a set of 16 bits, or two octets. The 16 bits are not to be treated as a 16 bit numeric value, but rather as 16 bits divided into 3 parts F, X, and Y, where the parts (F, X and Y) themselves are 2, 6 and 8 bits, respectively. It is the F X Y descriptors in BUFR Section 3 that refer to the data represented in Section 4 of a BUFR file.

Schematically, a BUFR descriptor can be visualized as follows:

F denotes the type of descriptor. With 2 bits, there are 4 possible values for F: 0, 1, 2 and 3. The four values have the following meanings:

F = 0 ➔ Element descriptor (Table B entry)
F = 1 ➔ Replication operator
F = 2 ➔ Operator descriptor (Table C entry)
F = 3 ➔ Sequence descriptor (Table D entry)

X (6 bits: 00-63) indicates the class or category of a descriptor.

Y (8 bits: range from 00-255) indicates the entry within a class X.

DX BUFR table mnemonic

The **mnemonic** has already appeared many times before and has been used to help explain the examples in Chapter 2. In short, a mnemonic is simply a descriptive,

alphanumeric name for a data value. The mnemonic is listed in the first column of the DX BUFR table. In the context of the BUFR table, at the highest level, we have a Table A mnemonic which completely describes a specific type of data subset (e.g. rawinsonde, wind profiler, etc.), and this Table A mnemonic is defined as a sequence of one or more Table B or Table D mnemonics, where each Table D mnemonic is likewise defined as a sequence of one or more Table B or Table D mnemonics, and so on until the entire data subset can be equivalently described as a sequence of one or more Table B mnemonics which, again, all correspond to basic data types (e.g. pressure, temperature, humidity, etc.). In this way, the entire sequence of data values that constitute a particular type of data subset is fully and unambiguously defined, both for purposes of reading/decoding or writing/encoding of reports.
As an example, we will expand the message type "ADPUPA" in figure 3.1, which is a simplified version of "*prepobs_prep.bufrtable*" as an example.

Section 1: TABLE A MNEMONIC for "ADPUPA":

```
|-------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| ADPUPA   | A48102 | UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS       |
|
```

Section 2: Table B, D MNEMONIC making up SEQUENCE of "ADPUPA" and its sub-SEQUENCE of HEADER and PRSLEVEL:

```
| MNEMONIC | SEQUENCE                                                       |
|----------|----------------------------------------------------------------|
| ADPUPA   | HEADR  SIRC  {PRSLEVEL}  <SST_INFO>  <PREWXSEQ>  {CLOUDSEQ}    |
| ADPUPA   | <CLOU2SEQ>  <SWINDSEQ>  <AFIC_SEQ>  <TURB3SEQ>                 |

| HEADR    | SID  XOB  YOB  DHR  ELV  TYP  T29  TSB  ITP  SQN  PROCN  RPT  |
| HEADR    | TCOR  <RSRD_SEQ>                                               |
|          |
| PRSLEVEL | CAT <P___INFO> <Q___INFO> <T___INFO> <Z___INFO> <W___INFO>
| PRSLEVEL | <DRFTINFO>
```

Section 1: Table D MNEMONIC to descript SEQUENCE "HEADER" and "PRSLEVEL":

```
|-------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| HEADR    | 348001 | REPORT HEADER SEQUENCE                              |
| PRSLEVEL | 348002 | PRESSURE LEVEL SEQUENCE (ALL TYPES EXCEPT "GOESND",  |
                        "AIRCFT" and "AIRCAR")
```

Section 1: TABLE B MNEMONIC to descript variable SID, XOB, … and their descriptors:

```
| SID  | 001194 | STATION IDENTIFICATI                                    |
| XOB  | 006240 | LONGITUDE                                               |
| YOB  | 005002 | LATITUDE                                                |
| DHR  | 004215 | OBSERVATION TIME MINUS CYCLE TI                         |
| ELV  | 010199 | STATION ELEVATION                                       |
| TYP  | 055007 | PREPBUFR REPORT TYP                                     |
```

Section 3: TABLE B MNEMONIC to define SCALE, REFRENCE, BIT and UNITS of an element like SID, XOB, … :

MNEMONIC	SCAL	REFERENCE	BIT	UNITS
SID	0	0	64	CCITT IA5
XOB	2	-18000	16	DEG E
YOB	2	-9000	15	DEG N
DHR	3	-24000	16	HOURS
ELV	0	-1000	17	METER
TYP	0	0	9	CODE TABL

Fully expanding all the sequences, even if you need to do it by hand, will help you better understand exactly what is contained within the report; for example, the report "ADPUPA" is expanded here:

MNEMONIC	SEQUENCE									
ADPUPA	HEADR	SIRC	{PRSLEVEL}	<SST_INFO>	<PREWXSEQ>	{CLOUDSEQ}				
ADPUPA	<CLOU2SEQ>	<SWINDSEQ>	<AFIC_SEQ>	<TURB3SEQ>						

HEADR	SID	XOB	YOB	DHR	ELV	TYP	T29	TSB	ITP	SQN
HEADR	PROCN	RPT	TCOR	<RSRD_SEQ>						

PRSLEVEL	CAT	<P___INFO>	<Q___INFO>	<T___INFO>	<Z___INFO>	<W___INFO>
PRSLEVEL		<DRFTINFO>				

P___INFO	[P__EVENT]		<P__BACKG>	<P__POSTP>
Q___INFO	[Q__EVENT]	TDO	<Q__BACKG>	<Q__POSTP>
T___INFO	[T__EVENT]	TVO	<T__BACKG>	<T__POSTP>
Z___INFO	[Z__EVENT]		<Z__BACKG>	<Z__POSTP>

P__EVENT		POB		PQM	PPC	PRC
Q__EVENT		QOB		QQM	QPC	QRC
T__EVENT		TOB		TQM	TPC	TRC
Z__EVENT		ZOB		ZQM	ZPC	ZRC

P__BACKG	POE	PFC	<PFC__MSQ>
Q__BACKG	QOE	QFC	<QFC__MSQ>
T__BACKG	TOE	TFC	<TFC__MSQ>
Z__BACKG		ZFC	<ZFC__MSQ>

P__POSTP	PAN	<PCLIMATO>	POETU	PVWTG	PVWTA	
Q__POSTP	QAN	<QCLIMATO>	QOETU	QVWTG	QVWTA	ESBAK
T__POSTP	TAN	<TCLIMATO>	TOETU	TVWTG	TVWTA	
Z__POSTP	ZAN	<ZCLIMATO>				

3.1.2 WMO BURF table A, B, D in DX BUFR Table

WMO BUFR employs 3 types of tables: content definition tables, code tables and flag tables. The BUFR content definition tables contain information to describe, classify and define the contents of a BUFR message. There are 4 such tables defined: Tables A, B, C and D.

Note: All these tables are available on-line from WMO website:

http://www.wmo.int/pages/prog/www/WMOCodes/TDCFtables.html#TDCFtables

Here we will introduce these tables briefly:

Table A subdivides data into a number of discrete categories, for example "ADPUPA" and "ADPSFC".

Table B describes how individual parameters, or elements, are to be encoded and decoded in BUFR. As an example, extracts of BUFR Table B for Temperature is given below.

Class 12 - Temperature

TABLE REFERENCE			TABLE ELEMENT NAME	BUFR			
				UNIT	SCALE	REFERENCE VALUE	DATA WIDTH (Bits)
F	X	Y					
0	12	001	Temperature/dry-bulb temperature	K	1	0	12
0	12	002	Wet-bulb temperature	K	1	0	12
0	12	245	Temperature	C	1	-2732	14

The following is the Table B from the DX BUFR table for element "TOB"

In section 1 of the DX BUFR table:

```
| TOB      | 012245 | TEMPERATURE OBSERVATION                                  |
```

In Section 3 of the DX BUFR table:

```
| TOB      |    1 |       -2732 |  14 | DEG C                    |-------------|
```

We can see the WMO table B has been divided into two parts that are distributed in DX BUFR Table section 1 and 3. These two parts of Table B for the same element are linked with a common name "TOB", which is called MNEMONIC.

Table B is fundamental to encoding and decoding in BUFR.

TABLE C defines a number of operations that can be applied to the elements. Each such operation is assigned an operator descriptor.

TABLE D defines groups of elements that are always transmitted together (like a regular SYNOP or TEMP report) in what is called a *common sequence*. By using a common sequence descriptor, the individual element descriptors will not need to be listed each time in the data description section. This will

reduce the amount of space required for a BUFR message. An example of BUFR Table D from DX BUFR table is shown below.

```
| T__EVENT | 348173 | TEMPERATURE EVENT SEQUENCE                                    |
| T__EVENT | TOB  TQM  TPC  TRC                                                    |
```

The flag and code tables will be introduced in the section 3.2 of this Chapter.

3.1.3 DX BUFR table sections

We have shown that a DX BUFR table file consists of three distinct sections. Here we will introduce these three section in more details and repeat some description to help user further understand the content of the DX BUFR table and its relation with WMO BUFR tables.

Section 1

As previously mentioned, the first section of a BUFR tables file is where all Table A, B, and D mnemonics are initially declared, assigned a unique FXY number, and given a short free-form text description. In figure 3.1, `TABLE A MNEMONIC`, `TABLE D MNEMONIC`, `TABLE B MNEMONIC` are the examples for this section. Mnemonics may contain any combination of uppercase letters and numbers (or, in certain special cases, a "." character!) and up to a maximum total of 8 characters in length. A mnemonic may be declared only once and must correspond to a unique FXY number, which itself consists of 6 characters, where the first character (i.e. the "F" component) is an "A" if the mnemonic is being declared as a Table A mnemonic, "3" if the mnemonic is being declared as a Table D mnemonic, and "0" if the mnemonic is being declared as a Table B mnemonic. Otherwise, the remainder of the FXY number must be all digits, with the next 2 characters (i.e. the "X" component) as a number between 00 and 63, and the final 3 characters (i.e. the "Y" component) as a number between 001 and 255. When the FXY number corresponding to a Table A mnemonic is actually encoded into a BUFR message, then a "3" is encoded in place of the "A" which is used in the tables file.

All Table A mnemonics are declared first, followed by all of the Table D mnemonics, then followed by the Table B mnemonics.

Section 2

Now, let's move onto the second section of a BUFR tables file. As already stated, this section is used to define the sequence of Table B (and possibly other Table D!) for

each Table A and Table D mnemonic that was previously declared in the first section. See `Table A, D MNEMONIC MAKING UP SEQUENCE` in figure 3.1 as an example.
At this point, readers may notice some punctuation characters and symbols included within the sequence definitions of the second section. It is now time to address these concerns by stating that these are replication indicators for the mnemonic(s) in question:

< >	Indicates that the enclosed mnemonic is replicated using 1-bit delayed replication (either 0 or 1 replications)
{ }	Indicates that the enclosed mnemonic is replicated using 8-bit delayed replication (between 0 and 255 replications)
( )	Indicates that the enclosed mnemonic is replicated using 16-bit delayed replication (between 0 and 65535 replications)
" "n	Indicates that the enclosed mnemonic is replicated using regular (non-delayed) replication, with a fixed replication factor of *n*

Examples of most of these cases are shown within the BUFR tables file. Here we look at the `Table A, D MNEMONIC MAKING UP SEQUENCE` in figure 3.1 for message type "ADPUPA".

1. `"HEADR"`, followed by
2. `"SIRC"`, followed by
3. between 0 and 255 replications of `"PRSLEVEL"`, followed by
4. either 0 or 1 replications of `"SST_INFO"`, followed by
5. either 0 or 1 replications of `"PREWXSEQ"`, followed by
6. between 0 and 255 replications of `"CLOUDSEQ"`, followed by
7. either 0 or 1 replications of `"CLOU2SEQ"`, followed by
8. either 0 or 1 replications of `"SWINDSEQ"`, followed by
9. either 0 or 1 replications of `"AFIC_SEQ"`, followed by
10. either 0 or 1 replications of `"TURB3SEQ"`, followed by

Where, e.g., the Table D mnemonic "PRSLEVEL" itself consists of the following sequence:

1. `"CAT"`, followed by
2. either 0 or 1 replications of `"P___INFO"`, followed by
3. either 0 or 1 replications of `"Q___INFO"`, followed by
4. either 0 or 1 replications of `"T___INFO"`, followed by
5. either 0 or 1 replications of `"Z___INFO"`, followed by
6. either 0 or 1 replications of `"W___INFO"`, followed by
7. either 0 or 1 replications of `"DRFTINFO"`

And where, in turn, `"P_INFO", "Q_INFO", "T_INFO","Z_INFO","W_INFO","DRFTINFO",` are also Table D mnemonics which can themselves be further resolved. Therfore, we can nest certain replication sequences inside of other replication sequences via the judicious use of the < > indicator, and even turn on/off entire sequences of data values simply and efficiently. An example of this is the "`DRFTINFO` " (i.e. "PROFILE LEVEL TIME/LOCATION INFORMATION ") sequence. In this case, enclosing the entire sequence within a < > indicator allows the lack of such data within a report by the use of a single bit set to "0" (i.e. 0 replications), rather than having to store the appropriate "missing" value for each constituent data value. This can add up to significant encoding efficiency and, in turn, the use of less required storage space per BUFR message.

We notice that Table D mnemonics such as `"HEADR"` is used within all data subset types. `"HEADR"` is constituted with Table B mnemonics such as:

```
SID     XOB      YOB      DHR      ELV      TYP      T29      TSB      ITP      SQN ….
```

This brings up a good point: that by logically grouping certain Table B mnemonics together within Table D sequence mnemonics, such mnemonics can be easily and efficiently re-used within different Table A mnemonic definitions within the same BUFR tables file. In fact, when using the BUFRLIB software, Table D sequence mnemonics are the only types of mnemonics upon which any type of replication may be directly performed. Thus, in particular, if we wish to effect the replication of a single, particular Table B mnemonic, then we must do so by defining a Table D sequence mnemonic whose only constituent is that particular Table B mnemonic and then replicate the sequence mnemonic.

In *prepobs_prep.bufrtable*, let's search sequence of Table D mnemonic `"RRTEVENT"` (RAIN RATE EVENT SEQUENCE):

```
      | RRTEVENT | 202130 201134 REQV 201000 202000  RRTQM  RRTPC  RRTRC |
```

Notice 201YYY indicator precedes `"REQV"`. This indicator is called an *operator*, the effect of this operator is that, for each Table B mnemonic which follows it within the current sequence and continuing up until the point in the sequence where a corresponding 201000 operator is reached (and which turns off the effect), (YYY – 128) bits should be added to the bit width that is defined for that Table B mnemonic within the third section of the BUFR tables file. The net effect is to change the number of bits occupied by the data value corresponding to that mnemonic within the overall data subset. In this example, the sequence:

```
      | 201134 |  REQV  201000 |
```

Indicates that (134 - 128) = 6 bits should be added to the data width that was defined for mnemonic `REQV` within the third section of the BUFR tables file; therefore, for `REQV`, the corresponding data value will occupy (12 + 6) = 18 bits.

Other than 201YYY, the BUFRLIB software also supports the similar use of the 202YYY (change scale), 204YYY (add associated field), 205YYY (add character data), 206YYY (define data width for local descriptor), 207YYY (increase scale, reference value and data width) and 208YYY (change data width for CCITT IA5 descriptor) operators from BUFR Table C.

Finally, take a look at the definitions of the Table D sequence mnemonics "TOPC_SEQ" (TOTAL PRECIPITATION/TOTAL WATER EQUIVALENT SEQUENCE)

```
| TOPC_SEQ | .DTHTOPC    TOPC |
```

There is reference to mnemonics ".DTHTOPC" which was not previously-declared within the first section of the table. However, notice that there does exist a declarations for mnemonics ".DTH....". So, what exactly is going on here? The answer is that each of these is a special mnemonic known as a *following-value mnemonic*, meaning that, when it is used within a sequence definition, it implies a special relationship with the mnemonic that immediately follows it within the sequence. In fact, this relationship is so special that, when a following-value mnemonic is used within a sequence definition, the "...." portion of the mnemonic is replaced with the mnemonic that immediately follows it! For example, when ".DTH...." is used within the definition of the Table D sequence mnemonic "TOPC_SEQ", it appears as ".DTHTOPC" because it appears immediately before the mnemonics "TOPC". However, in the definition of "TMXMNSEQ", (MAXIMUM/MINIMUM TEMPERATURE SEQUENCE)

```
| TMXMNSEQ | .DTHMXTM    MXTM    .DTHMITM   MITM |
```

It appears as ".DTHMXTM" since it immediately precedes "MXTM" within that sequence! To be precise, a following-value mnemonic is declared with a "." as the first character, followed by no more than 3 alphanumeric characters as an identifier, followed by 4 more "." characters, which must then be replaced with the mnemonic that immediately follows it whenever and wherever it is used within a sequence definition.

Section 3

It is now time to move on to the third section of a BUFR tables file. As we mentioned earlier, this section is used to define the scale factor, reference value, data width, and units for all of the Table B mnemonics. See TABLE B MNEMONIC UNIT in figure 3.1 as an example. In particular, the reader may recall that the units definition for each Table B mnemonic in turn determines how data values corresponding to that mnemonic are read/written from/to the REAL*8 array “R8ARR” within the BUFRLIB subroutines UFBINT and UFBREP. Note that any mnemonic whose corresponding data values are to be treated as character data must have its units listed as "CCITT IA5", which is just a formal synonym for ASCII.

3.2 Examples of the DX BUFR table's application

From the examples in chapter 2, we know BUFR table is required when a user reads/writes data subset from/into BUFR and PrepBUFR file. After learning the DX BUFR tables content in section 3.1 of this Chapter, we can review the mnemonics used in this example code again to see how a BUFR table can be applied in the user's own application.

Application for PrepBUFR decoding

In Chapter 2, we first used three basic examples to illustrate the code structure and the BUFRLIB functions used in the NCEP BUFR file processing (decoding, encoding, and appending). Then, we gave 6 examples of PrepBUFR file processing for surface, upper level and retrieved observations. Users may already notice that the major differences of these PrepBUFR examples from the basic examples are the mnemonics and the size of data arrays. In chapter 2 PrepBUFR processing examples, the mnemonics look like the following:

```
character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
```

The Strings `hdstr, obstr, qcstr, oestr` define the Table B mnemonic. These mnemonic are contained in the PrepBUFR file, and either need to be read out or written into the file.

When we see these lists of the mnemonics, one question we ask may be: what are exactly meanings of these mnemonics? Obviously, the answer is to check the BUFR table, which already provides enough information for us to understand them. Here, we can use *prepobs_prep.bufrtable* in GSI directory *./util/bufr_tools* as the default BUFR table unless the users have created their own table. The following are some examples of how to search the DX BUFR table for the meanings of a certain mnemonic:

1) mnemonic `"SID"`:

When we look at the table and search for the appearances of `"SID"`, We found three lines of the table including `"SID"` :

```
|-------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| SID      | 001194 | STATION IDENTIFICATION
```

This line is in the BUFR table section 1. The `"SID"` appears as the table B mnemonic, since F is 0 in the FXY number. Here, we also know it is "Station Identification".

```
| MNEMONIC | SEQUENCE                                                          |
|----------|-------------------------------------------------------------------|
| HEADR    | SID  XOB  YOB  DHR  ELV  TYP  T29  TSB  ITP  SQN  PROCN  RPT
```

This line is in the BUFR table section 2. The "SID" appears in the sequence of Table D mnemonic "HEADR".

```
|-------------------------------------------------------------------------|
| MNEMONIC | SCAL | REFERENCE   | BIT | UNITS                |-------------|
|----------|------|-------------|-----|----------------------|-------------|
| SID      |    0 |           0 |  64 | CCITT IA5
```

This line is in the BUFR table section 3. It shows the scale, reference, bit, and units of element "SID". Here, the bit is 64, which means "SID" is saved in 64 bits in the binary file. CCITT IA5 is the fancy name for ASCII, and it shows "SID" is character string. Usually the unit is most often used in user's application.

2) mnemonic "POB":

A search for "POB" also shows three lines in the BUFR table:

```
| POB      | 007245 | PRESSURE OBSERVATION
```

This line is in the BUFR table section 1, "POB" appears as the table B mnemonic and means "pressure observation"

```
| P__EVENT | POB  PQM  PPC  PRC
```

This line is in the BUFR table section 2, "POB" appears in the sequence of Table D mnemonic "P_EVENT".

```
| POB      |    1 |           0 |  14 | MB
```

And finally, this line is in the BUFR table section 3. It shows "POB" uses MB as its unit.

3) mnemonic "PQM":

Clearly, "PQM" will have three lines:

```
| PQM      | 007246 | PRESSURE (QUALITY) MARKER

| P__EVENT | POB  PQM  PPC  PRC

| PQM      |    0 |           0 |   5 | CODE TABLE
```

These three lines are in the BUFR table section 1, 2, and 3 respectably. They tell us that PQM is the quality marker for the pressure observation and it appears in the sequence of Table D mnemonic "P_EVENT" . The unit for "PQM" is "CODE TABLE". If we

keep searching the *prepobs_prep.bufrtable*, users may notice that `CODE TABLE` appears quit frequently as the Table B mnemonic unit. What does it mean? From the previous introduction to the BUFR tables, we learned that, other than Table A, B, C, and D, the BUFR tables also include flag and code tables, which are used for the element based on a code (e.g., Cloud Type) or a set of conditions defined by flags (bits set to 0 or 1). From practical point of view, the following link of NCEP BUFR table is very helpful for us to find the content of code and flag table:

http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_1.htm

This document not only contains Table A, B, D mnemonic information in *prepobs_prep.bufrtable*, but also includes other useful information, e.g. what is `CODE TABLE` for `"PQM"`? Click the above link and search for `"PQM"` in the table until you find the line:

```
PQM    007246    PRESSURE (QUALITY) MARKER    0    0    5    CODE TABLE
```

Then, click link under `CODE TABLE`, it will lead you to see NCEP quality marker value for `"PQM"`.

Another example is to search `"PPC"`. You will find the following line:

```
PPC     007247    PRESSURE EVENT PROGRAM CODE    0    0    5    CODE TABLE
```

Then, click `CODE TABLE` link, which leads you to the page of "NCEP code/flag tables associated with BUFR table B (includes code/flag tables awaiting validation by the WMO)". You can find out the code value for `"PPC"` from there.

From above examples, we can see it is quite straightforward to find the meaning of a BUFR table B element listed in the mnemonic list for the PrepBUFR file process. But a more difficult question is how to get the mnemonic list used in the previous example code. Especially, if users want to decode/encode the observation variables no included in the list. The lists used in the PrepBUFR processing examples are from the GSI BUFR ingest interface subroutine. We will discuss how to find these mnemonics from GSI code in Chapter 4. Here we will follow the general method of expanding the message type that was introduced in section 3.1 of this Chapter.

In the decoding examples of Chapter 2, the following code is used to read in a message:

```
msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
```

After call function *ireadmg*, a message type will be returned in variable "*subset*". Starting from the message type in "subset", we can find the complete contents inside the message. Here, we will use the message type "ADPUPA" as an example. From the DX BUFR table example of figure 3.1, we can expand the message "ADPUPA" based on section 2 of the table:

First, we can see that “HEADER” and “PRSLEVEL” is in the Sequence of ADPUPA:

```
|-------------------------------------------------------------------------------|
| MNEMONIC | SEQUENCE                                                           |
|----------|--------------------------------------------------------------------|
|          |                                                                    |
| ADPUPA   | HEADR  SIRC  {PRSLEVEL}  <SST_INFO>  <PREWXSEQ>  {CLOUDSEQ}        |
| ADPUPA   | <CLOU2SEQ>  <SWINDSEQ>  <AFIC_SEQ>  <TURB3SEQ>                     |
|          |                                                                    |
```

Then, by checking the sequence of “HEADER”, we can find information related to the observation station, such as the observation station ID, location, time, etc.

```
| HEADR    | SID  XOB  YOB  DHR  ELV  TYP  T29  TSB  ITP  SQN  PROCN  RPT      |
| HEADR    | TCOR  <RSRD_SEQ>                                                  |
```

Next, if we check the sequence of “PRSLEVEL”, we can see “PRSLEVEL” includes several sequences related to the observation elements such as pressure (`P___INFO`), moisture (`Q___INFO`), temperature (`T___INFO`), height (`Z___INFO`), and wind (`W___INFO`) observations:

```
|          |                                                                    |
| PRSLEVEL | CAT  <P___INFO>  <Q___INFO>  <T___INFO>  <Z___INFO>  <W___INFO>    |
| PRSLEVEL | <DRFTINFO>                                                         |
```

If we keep checking the sequence for each of the observation elements, for example, temperature (`T___INFO`), we will see the temperature observation mnemonic (`TOB`), temperature observation quality marker mnemonic (`TQM`), and temperature observation error mnemonic (`TOE`):

```
|          |                                                                    |
| T___INFO | [T__EVENT]  TVO  <T__BACKG>  <T__POSTP>                            |
|          |                                                                    |
| T__EVENT | TOB  TQM  TPC  TRC                                                 |
|          |                                                                    |
| T__BACKG | TOE  TFC  <TFC__MSQ>                                               |
|          |                                                                    |
```

In the PrepBUFR decoding example, TOB, TQM, and TOE are listed in the mnemonics that will be read in by the code. We can certainly add TVO or TPC to the mnemonic list to let the code know that we want to know these two values too.

Application for Radiance observation decoding

On the other side, the example *bufr_decode_radiance.f90* in section 2.3 of Chapter 2 introduces the code to read TOVS 1b radiance observations from a radiance BUFR file. The mnemonics listed in this example are quit different from ones in the PrepBUFR examples:

```
hdstr='SAID FOVN YEAR MNTH DAYS HOUR MINU SECO CLAT CLON CLATH CLONH HOLS'
hdr2b='SAZA SOZA BEARAZ SOLAZI'
obstr='TMBR'
```

To get to know what the exactly meanings of these mnemonics are and what other mnemonics could be included in this list for more observation information, we need to check the BUFR DX table for radiance. The best way to get the DX BUFR table for a certain BUFR file is to extract the BUFR table embedded in the BUFR file using the BUFRLIB function: `dxdump`. In the example code *bufr_decode_radiance.f90*, after each run, we will find the DX BUFR table file for the radiance named as `'radiance.bufrtable'` in the running directory. Based on this BUFR table and the message type information from BUFRLIB function *ireadmg*, we can use the same method to expand the message and get all available radiance observation variables out of the radiance BUFR file.

Please follow the instructions in this section for practice and please feel free to forward your questions and comments to our GSI help desk.

Chapter 4 GSI BUFR interface

GSI has a set of code to ingest and process observation data from BUFR/PrepBUFR files for the analysis. This Chapter will first explain the procedure of observation ingest and process within the GSI system. Then, we provide 4 examples from GSI observation ingesting subroutines to illustrate how GSI interfaces with the BUFR files.

4.1 GSI observation data ingest and process procedure

As an important component of any data analysis system, observation data ingesting and processing is a key part of the GSI system. The data types that can be used in the GSI analysis and the corresponding subroutines that read in these data types are listed in the section 6.2.3 of the GSI User's Guide for the release version 3. But there are more details that users should know in order to be able to handle the observation data in GSI with confidence and flexibility. This section introduces the complete structure of GSI observation data ingesting and processing step-by step, including run scripts and namelist setup, data ingesting driver routine, read subroutines, observation data partition, and innovation calculation.

- Step 1: Link BUFR/PrepBUFR file to GSI recognized names in GSI run scripts

In the GSI run script, there is a section to link the BUFR/PrepBUFR files to GSI recognized file names in the GSI run directory. The script looks like:

```
# Link to the prepbufr data
ln -s ${PREPBUFR} ./prepbufr

# Link to the radiance data
# ln -s ${OBS_ROOT}/gdas1.t12z.1bamua.tm00.bufr_d amsuabufr
# ln -s ${OBS_ROOT}/gdas1.t12z.1bhrs4.tm00.bufr_d hirs4bufr
# ln -s ${OBS_ROOT}/gdas1.t12z.1bmhs.tm00.bufr_d  mhsbufr
```

Clearly, the PrepBUFR file: *gdas1.t12z.prepbufr.nr*, which is the file pointed by *${PREPBUFR},* and the BUFR files: *gdas1.t12z.1bamua.tm00.bufr_d* and *gdas1.t12z.1bhrs4.tm00.bufr_d* are the files we downloaded from NCEP data hub. The names of these files are determined by NCEP based on the operation systems that use the files. Please see more details on the NCEP BUFR files in Chapter 5. The BUFR files used in GSI can also be the observation files generated by users and named by users. But GSI itself doesn't recognize the names of these files. So, in the GSI run scripts, these files must be linked to the GSI run directory with a name that GSI knows. In the section 3.1 of the GSI User's Guider Version 3.0, we gave a table that lists all the GSI recognized data file names at the left column, the contents of the data files at the middle column, and the sample GDAS BUFR/PrepBUFR file names at the left column. The following is a sample of the table.

GSI Name	Content	Example file names
prepbufr	Conventional observations, including ps, t, q, pw, uv, spd, dw, sst, from observation platforms such as METAR, sounding, et al.	gdas1.t12z.prepbufr
amsuabufr	AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17,18, 19 and METOP-A	gdas1.t12z.1bamua.tm00.bufr_d
amsubbufr	AMSU-B 1b radiance (brightness temperatures) from satellites NOAA15, 16,17	gdas1.t12z.1bamub.tm00.bufr_d
radarbufr	Radar radial velocity Level 2.5 data	ndas.t12z. radwnd. tm12.bufr_d
gpsrobufr	GPS radio occultation observation	gdas1.t12z.gpsro.tm00.bufr_d
ssmirrbufr	Precipitation rate observations fromSSM/I	gdas1.t12z.spssmi.tm00.bufr_d
hirs4bufr	HIRS4 1b radiance observation from satellite NOAA 18, 19 and METOP-A	gdas1.t12z.1bhrs4.tm00.bufr_d
msubufr	MSU observation from satgellite NOAA 14	gdas1.t12z.1bmsu.tm00.bufr_d

So, in the GSI run script, the files in the right column are linked to the run directory with a new name at the left column. As a matter of fact, the file names in the left column can be changed if users prefer to do so and know how to change them in the GSI namelist data file setup section. But we recommend to leave the file names as is because the current names in the left column are a good indication of the contents of the corresponding BUFR observation files and are used by many the GSI applications.

- Step 2: GSI Namelist data configuration section: &**OBS_INPUT**

In the GSI namelist, section *&OBS_INOUT* is used to setup data usage such as the links between data types and data files, data time window, and satellite data thinning. The following is a sample of the namelist section *&OBS_INOUT*:

```
&OBS_INPUT
 dmesh(1)=120.0,dmesh(2)=60.0,dmesh(3)=60.0,dmesh(4)=60.0,dmesh(5)=120,time_window_max=1.5,
 dfile(01)='prepbufr',  dtype(01)='ps',    dplat(01)=' ',    dsis(01)='ps',        dval(01)=1.0,  dthin(01)=0,
 dfile(02)='prepbufr'   dtype(02)='t',     dplat(02)=' ',    dsis(02)='t',         dval(02)=1.0,  dthin(02)=0,
 dfile(03)='prepbufr',  dtype(03)='q',     dplat(03)=' ',    dsis(03)='q',         dval(03)=1.0,  dthin(03)=0,
 dfile(04)='prepbufr',  dtype(04)='uv',    dplat(04)=' ',    dsis(04)='uv',        dval(04)=1.0,  dthin(04)=0,
……
 dfile(27)='msubufr',   dtype(27)='msu',   dplat(27)='n14',  dsis(27)='msu_n14',   dval(27)=2.0,  dthin(27)=2,
 dfile(28)='amsuabufr', dtype(28)='amsua', dplat(28)='n15',  dsis(28)='amsua_n15', dval(28)=10.0, dthin(28)=2,
 dfile(29)='amsuabufr', dtype(29)='amsua', dplat(29)='n16',  dsis(29)='amsua_n16', dval(29)=0.0,  dthin(29)=2,
```

Users may notice that the first column, *dfile*, is the GSI recognized file names listed in the section 3.1 of the GSI User's Guider Version 3.0. The 2nd column, *dtype*, is the observation type. The 3rd column, *dplat*, is satellite platform ID. And the 4th column, *dsis*, is the data type from convinfo file or Sensor/instrument/satellite flag from satinfo file.

In the GSI data ingesting driver, it is the data type, *dtype*, that is used to decide which routine to call for reading the data from the corresponding input file defined by *dfile*. For example, when the GSI reaches the code to read "`t`", it will open file `'prepbufr'` (`dfile(02)`) to read temperature in. Or when the GSI reaches the point to read in AMSU-A from NOAA 16, it will open file `'amsuabufr'` (`dfile(29)`) to read in the data. From the namelist setup, it is possible that GSI reads in "*t*" from one PrepBUFR file

(`dfile(02)`) but reads in '*q*' from another PrepBUFR file (`dfile(03)`), which gives more flexibility to control the data used in the GSI analysis.

- Step 3: GSI data ingest driver

In GSI, subroutine *read_obs* (inside *read_obs.F90*) is used to read, select, and reformat observation data. It is the driver for routines that read different types of the observational data. This routine loops through all data types listed in *dtype* and checks the data usage and file availability. If the data file exists and the info files indicate the use of the data type, one or several processors will be assigned to read the data from the corresponding file setup in `dfile`. Please refer to the section 4.3 of the GSI User's Guide V3 for more information on using the info file to control data usage. Here we give two chunks of the code from subroutine read_obs as examples to illustrate how to find routines that read different observation data types.

Example 1: Process conventional (prepbufr) data

```
!
  if(ditype(i) == 'conv')then
     if (obstype == 't'  .or. obstype == 'uv' .or. &
         obstype == 'q'  .or. obstype == 'ps' .or. &
         obstype == 'pw' .or. obstype == 'spd'.or. &
         obstype == 'mta_cld' .or. obstype == 'gos_ctp'  ) then
         call read_prepbufr(nread,npuse,nouse,infile,obstype,lunout,twind,sis,&
             prsl_full)
         string='READ_PREPBUFR'
```

From this chunk of the code, we can see the subroutine *read_prepbufr* will be used to read the data type *'t', 'uv', 'q', 'ps', 'pw', 'spd', 'mta_cld', 'gos_ctp*' from PrepBUFR file saved in "*infile*".

Example 2: Process TOVS 1b data

```
!
      if (platid /= 'aqua' .and. (obstype == 'amsua' .or. &
           obstype == 'amsub' .or. obstype == 'msu'   .or.  &
           obstype == 'mhs'   .or. obstype == 'hirs4' .or.  &
           obstype == 'hirs3' .or. obstype == 'hirs2' .or.  &
           obstype == 'ssu')) then
         llb=1
         lll=1
         if((obstype == 'amsua' .or. obstype == 'amsub' .or. obstype == 'mhs') .and. &
            (platid /= 'metop-a' .or. platid /='metop-b' .or. platid /= 'metop-c'))lll=2
         call read_bufrtovs(mype,val_dat,ithin,isfcalc,rmesh,platid,gstime,&
              infile,lunout,obstype,nread,npuse,nouse,twind,sis, &
              mype_root,mype_sub(mm1,i),npe_sub(i),mpi_comm_sub(i),llb,lll)
         string='READ_BUFRTOVS'
```

From this chunk of the code, we can see the subroutine *read_bufrtovs* will be used to read many kinds of radiance data such as *'amsua', 'amsub', 'msu', 'mhs', 'hirs', 'ssu'* from radiance BUFR file saved in "*infile*". But these radiance data are not observed by AQUA.

Table 4.1: List of data types and subroutines of GSI observation IO

Data type *(ditype)*	Observation type *(obstype)*		Subroutine that reads data	File includes Subroutine
conv	t, uv, q, ps, pw, spd, mta_cld, gos_ctp		read_prepbufr	read_prepbufr
	sst	from mods	read_modsbufr	read_modsbufr
		not from mods	read_prepbufr	read_prepbufr
	srw		read_superwinds	read_superwinds
	tcp		read_tcps	read_tcps
	lag		read_lag	read_lag
	rw (radar winds Level-2)		read_radar	read_radar
	dw (lidar winds)		read_lidar	read_lidar
	rad_ref		read_RadarRef_mosaic	read_RadarRef_mosaic
	lghtn		read_lightning	read_lightning
	larccld		read_NASA_LaRC	read_NASA_LaRC
	pm2_5		read_anowbufr	read_anowbufr
rad (satellite radiances)	(platform) not AQUA	amsub	read_bufrtovs (TOVS 1b data)	read_bufrtovs (TOVS 1b data)
		amsua		
		msu		
		mhs		
		hirs4,3,2		
		ssu		
	(platform) AQUA	airs	read_airs (airs data)	read_airs (airs data)
		amsua		
		hsb		
	iasi		read_iasi	read_iasi
	sndr, sndrd1, sndrd2 sndrd3, sndrd4		read_goesndr (GOES sounder data)	read_goesndr (GOES sounder data)
	ssmi		read_ssmi	read_ssmi
	amsre_low, amsre_mid amsre_hig		read_amsre	read_amsre
	ssmis, ssmis*		read_ssmis	read_ssmis
	goes_img		read_goesimg	read_goesimg
	seviri		read_seviri	read_seviri
	avhrr_navy		read_avhrr_navy	read_avhrr_navy
	avhrr		read_avhrr	read_avhrr
ozone	subuv2, omi, gome, o3lev		read_ozone	read_ozone
co	mopitt		read_co	read_co
pcp	pcp_ssmi, pcp_tmi, pcp_amsu,pcp_stage3		read_pcp	read_pcp
gps	gps_ref, gps_bnd		read_gps	read_gps
aero	modis		read_aerosol	read_aerosol

In the subroutine *read_obs*, users can find similar portion of the code deciding which subroutine is used to read in the data for certain data type. For each subroutine, the input variables always includes parameters like:

infile = *dfile* of the namelist section *&OBS_INOUT*
obstype = *dtype* of the namelist section *&OBS_INOUT*
sis = *dsis* of the namelist section *&OBS_INOUT*

- Step 4: Read in observations and initial check of the observations

The data types and the corresponding GSI subroutines that read in these data types are listed in the table 4.1. From the table, we can see there are 28 subroutines employed by GSI to read in different kinds of BUFR/PrepBUFR files. Also from the table, we can easily find the GSI subroutine that actually reads in the certain observations from the BUFR/PrepBUFR files. In the same subroutine, the quality control to the observation data, data thinning, and checks to insure that the data are in the analysis domain and time window.

These read_* subroutines listed in the table 4.1 are the GSI interface to the BUFR/PrepBUFR that users should check when trying to analyze their own data using the GSI system. We will discuss how to check the structure of these read_* subroutines in section 4.2 of this Chapter.

After we read in the observations for each element, such as "*t*", "*q*", "*wind*", GSI will write out observations for certain element in the analysis domain and time to one binary file, which will be read in again by the next step for data partitioning into sub-domains (if run with multiple processors).

- Step 5: sub-domain partition

When GSI runs in parallel mode, both the background and the observation data need to be partitioned into sub-domains. This step is done after the observation data have been read in and saved in the internal format. The code to assign and distribute observations to sub-domains is call "*obs_para*", which is a subroutine inside the file "*obs_para.f90*". Please note that after this step, the observations from all observation elements are saved in the same binary file for each processor.

- Step 6: innovation calculation

As an important step of the data analysis system, observation innovation calculation also involves lots of code. The section 6.2.4 of the GSI User's Guide version 3 provides a table to list innovation calculations for the different kinds of observation elements. We will not introduce these calculations in this document but would like to remind users that innovation calculation is also a key component in the use of observation data in the analysis.

4.2 The BUFR decoding in GSI read files

From the previous section, we can see that there are many steps involved in the GSI system to ingest and process the observation data from BUFR/PrepBUFR files for the final analysis. To encode new data for the GSI, the best way to start is reading the related GSI code for BUFR/PrepBUFR data ingesting and checking the mnemonics used in the code to figure out the data needed in the GSI. In section 4.1, we have provided a complete list of GSI subroutines for the observation data ingesting. Here we will give 2 examples to illustrate how to extract the GSI BUFR interface from the GSI read_* subroutines and delete other functions that are not related to the BUFR decoding from the subroutine, such as observation location and time checking, data thinning, and quality control checking, etc.

Example 1: read_prebufr.f90

The file read_prepbufr.f90 is in GSI source code directory (./src/main) and it reads conventional data from the PrepBUFR file. Specific observation types read by this routine include surface pressure, temperature, winds (components and speeds), moisture, total precipitable water, and cloud and weather. This file has 1375 lines and most of the code are not related to the PrepBUFR decoding. Here, as an example, we deleted all the code that are not for PrepBUFR decoding and shortened the file down to 197 lines. The full code is listed in the Appendix and can be downloaded from the Examples Page of the BUFR website. Here we will only show the mnemonics used by the GSI PrepBUFR decoding to get an idea what are the GSI expected variables from the PrepBUFR file.

```
data hdstr  /'SID XOB YOB DHR TYP ELV SAID T29'/
data hdstr2 /'TYP SAID T29 SID'/
data obstr  /'POB QOB TOB ZOB UOB VOB PWO CAT PRSS' /
data drift  /'XDR YDR HRDR                    '/
data sststr /'MSST DBSS SST1 SSTQM SSTOE           '/
data qcstr  /'PQM QQM TQM ZQM WQM NUL PWQ     '/
data oestr  /'POE QOE TOE NUL WOE NUL PWE     '/
data satqcstr  /'QIFN'/
data prvstr /'PRVSTG'/
data sprvstr /'SPRVSTG'/
data levstr  /'POB'/
data metarcldstr /'CLAM HOCB'/      ! cloud amount and cloud base height
data metarwthstr /'PRWE'/           ! present weather
data metarvisstr /'HOVI'/           ! visibility
data geoscldstr /'CDTP TOCC GCDTT CDTP_QM'/
```

Compared to the PrepBUFR processing examples we provided in the Chapter 2, we can see that there is more information expected by the GSI PrepBUFR interface. Please note that not all the variables listed in the above mnemonics are needed for a GSI run. Some are for certain special GSI applications only, such as the cloud observations, which are used in the Rapid Refresh system only. So, if users only want to generate a PrepBUFR file that contains a part of the observations expected by these mnemonics, the GSI still can run successfully and use the observation data to get a final analysis. But from the previous introduction to the GSI observation data processing procedure, users can see that there are many steps involved in the data usage in the GSI analysis. A complete picture of the data flow in GSI system will be very helpful for users who work on data

impact studies with GSI, especially when they need to generate the new PrepBUFR file for their new data.

Example 2: read_airs.f90:

The file read_airs.f90 is in GSI source code directory (./src/main) and it reads BUFR format AQUA radiance (brightness temperature) observations. This file has 768 lines. To simplify this example, we deleted all the code that is not related to the BUFR decoding and shortened the file down to 82 lines. The full code is listed in the Appendix and can be download from the Examples Page of the BUFR website. Again, we will only show the lines that include mnemonics used by decoding to get an idea what variables are expected by GSI from the AIRS BUFR file.

```
allspotlist='SIID YEAR MNTH DAYS HOUR MINU SECO CLATH CLONH SAZA BEARAZ FOVN'

 call ufbrep(lnbufr,allchan,1,n_totchan,iret,'TMBR')

 call ufbint(lnbufr,aquaspot,2,1,iret,'SOZA SOLAZI')
```

Here, we highlight the mnemonics and we will leave then for users to find out the exactly meaning of these mnemonics by checking the BUFR table.

Summary:

In the course of preparing this document and extending the BUFR/PrepBUFR support for GSI, we outline portions of 4 GSI BUFR ingest interface files for users to reference:

read_prepbufr.f90
read_airs.f90
read_bufrtovs.f90
read_gps.f90

Users can find these files in the Examples Page of the BUFR user's website. There is makefile provided with these files to help users properly compile the code. These files can also be used to decode the corresponding NCEP operation PrepBUFR/BUFR files.

Chapter 5 NCEP Observation Process and BUFR files

5.1 Observation Data Processing at NCEP

Here is a brief outline of the current procedure for processing observations that arrive at NCEP.

1. NCEP receives the majority of its data from the Global Telecommunications System (GTS) and the National Environmental Satellite, Data, and Information Service (NESDIS).
2. The various NCEP networks access the observational database at a set time every day (i.e., the data cutoff time) and perform a time-windowed dump of requested observations. Observations of a similar type [e.g., satellite-derived winds ("satwnd"), surface land reports ("adpsfc")] are dumped into individual BUFR files which maintain the original structure of reports, although some interactive quality control is applied, duplicate reports are removed, and upper-air report "parts" are merged.
3. The final step in preparing most of the observational data for assimilation involves the execution of a series of programs which read in the observations from the various dump files, add forecast ("first guess") background and observation error information, perform automated quality control, and finally output the observations in a monolithic BUFR file known as "PREPBUFR".
4. The PREPBUFR file is read by the Global Statistical Interpolation (GSI) analysis.

The figure depicts the data processing system at NCEP is in the next page and the naming convention used in the figure is listed here:

GTS = Global Telecommunications System
Gathers world wide data, sends data to NWSTG/TOC or the “Gateway”

NWSTG/TOC = NWS Telecommunication Gateway/Telecommunication Operations Center
Intercepts GTS messages, sends data to **NCO** via TNC (TOC to NCEP Communications) line and LDM (Local Data Manager)

GSD = NOAA/ESRL/GSD
Provides Mesonet data to **NCO** via LDM several times hourly

Radar/ROC = NOAA Radar Operations Center
Provides the radial wind and reflectivity data from 158 NEXRAD radar stations.

NESDIS = National Environmental Satellite, Data, and Information Service Servers
Serves up POES data (operational/test), GOES data, winds, radiances, SST, (operational/test)

NCO = NCEP Central Operations

PMB = Production Management Branch
Pulls data from the outside, interacts w/ the "Gateway", LDM, ROC, etc …, retrieves data continuously, gathers all the data then pass it off to SIB for decoding.
Code is running 24/7, implements changes.

SIB = Systems Integration Branch
Decodes data from native format to NCEP BUFR and stores data to tanks. Decoders include: ACARS, Aircraft, Aviation Weather (METAR), Bathymetry, Drifting Buoy, Land surface, Marine surface, NeXRAD Wind, Profiler, Rawinsonde, Satellite Wind, Supplementary Climatology, Tide Gauge.

EMC = Environmental Modeling Center
Runs SMS jobs to periodically query the NESDIS servers for new data. Compares latest available data against a local history file to determine if data is new. Retrieves new data via FTP. Converts native data to NCEP BUFR and stores them in the tanks.

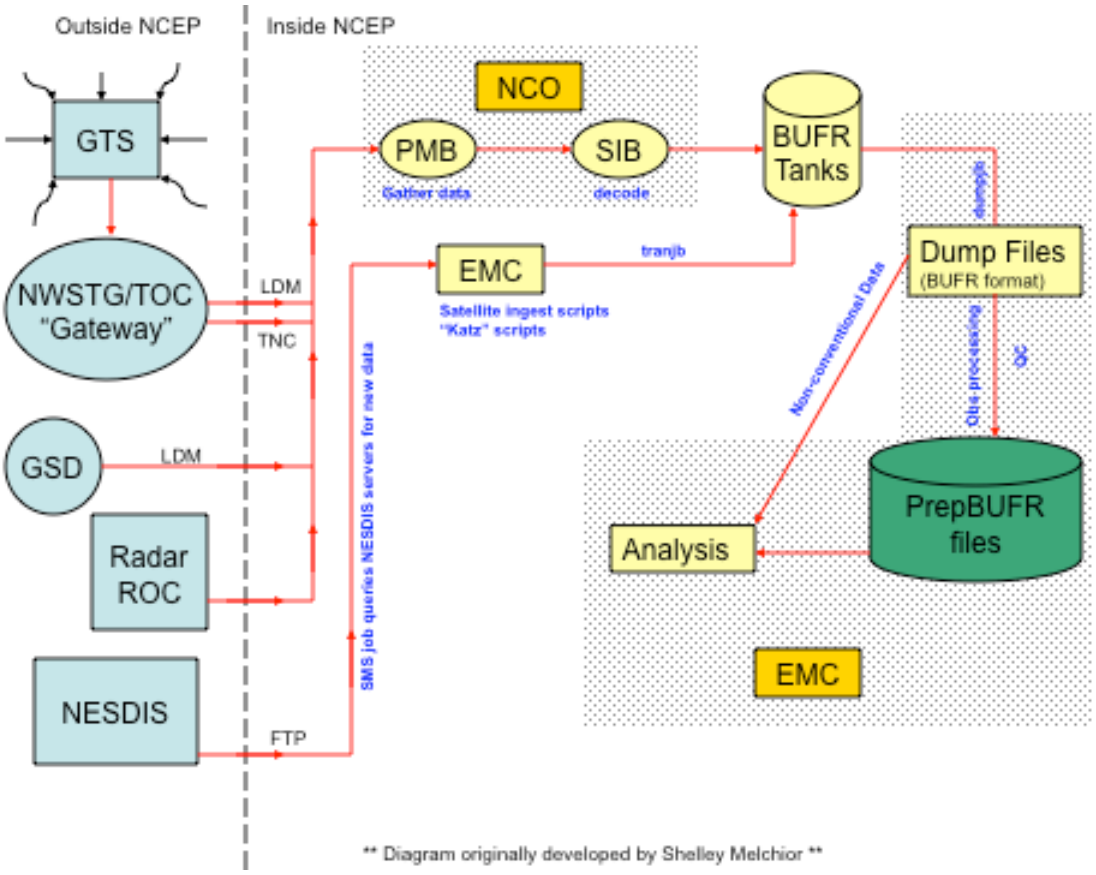


Figure 5.1 data processing system at NCEP

5.2 NCEP BUFR/PrepBUFR

NCEP saves most of the observation data in WMO BUFR format. PrepBUFR is the final step in preparing most of the observations for data assimilation, the NCEP term for "prepared" or QC'd data in BUFR format (NCEP convention/standard). Please note that a PrepBUFR file is still a BUFR file, but has more QC information. NCEP uses PrepBUFR files to organize conventional observations and satellite retrievals as well as other related information (such as quality marks) into single files. The BUFRLIB software and BUFR table are needed for processing BUFR/PrepBUFR files.

NCEP generates different BUFR/PrepBUFR files for each of its operation systems. The "PrepBUFR" includes the major conventional observations for assimilation into the various NCEP analyses, including the North American Model (NAM) and NAM Data Assimilation System (NDAS), unified grid-point statistical interpolation analysis (GSI) (the "NAM" and "NDAS" networks), the Global Forecast System and Global Data Assimilation System unified GSI (the "GFS" and "GDAS" networks), the Climate Data Assimilation System SSI (the "CDAS" network), the Rapid Update Cycle (the "RUC" network) and the Real Time Mesoscale Analysis (the "RTMA" network).

In this section, we will briefly introduce several types of BUFR/ PrepBUFR files mostly accessed by the research community to help users decide which one is the best for their GSI applications. Each type of BUFR/PrepBUFR file has its own coverage, data cut-off time, and quality control procedures, which result in different quality marker values for the same observation in different files.

- File name convention:

  The following is a list of example file names we collected from NCEP FTP site:

  gdas1.t00z.prepbufr.nr
  gfs.t00z.gpsro.tm00.bufr_d
  ndas.t18z.1bamub.tm03.bufr_d
  nam.t00z.aircar.tm00.bufr_d.nr
  ndas.t18z.prepbufr.tm03.nr

  These file names reflect information of the observations within the file. Let us explain the meaning of the filenames, segment by segment, separated by dots:

  - The 1st section is the operation system name, indicating which operation system this file is created/used for. For example: gdas1 is for the Global Data Assimilation System (GDAS), gfs for the Global Forecast System (GFS), ndas for the North American Data Assimilation

System (NDAS), nam for the North American Mesoscale (NAM) forecast system.
    - The second section is analysis hour, indicating which analysis hour this file is used for. For example: t00z is for 00Z analysis, t18z for 18Z analysis.
    - The third section is data type, indicating what kinds of data are included in the file. For example: prepbufr is for conventional observations, gpsro for GSPRO, 1bamub for AMSU-B, and aircar for aircraft observations.
    - From fourth section, there is different information for different operational files:
        - bufr_d tells us it is a BUFR format file. We may think prepbufr as a special data "format" here.
        - nr tells us that the file only includes non-restricted data (we can only access non-restricted data).
        - tm00 and tm03, where the two digital number is hours. They also indicate the time of the file used in the analysis. When the number is 00, the file analysis time is the same as showed in the second segment. When it is a number larger than 0, it indicates the analysis time of the file is the time in the second segment minus this number. For example: the analysis time for ndas.t18z.1bamub.tm03.bufr_d is 15Z (18Z - 03h = 15Z). This file is used in the catch up cycles during NDAS that have a delayed analysis start time to wait for more observations.

- Data coverage and cut off time:

  Each operational system requires different data types, data coverage, cut off time, and quality control procedures. The details of these setups need a long technical note to describe but here we can briefly introduce some major features of each file:

    - GDAS (gdas1) covers the global and has the latest cut off time (6 hours), which means it includes most of the available real-time observation data.
    - GFS (gfs) covers the global but has a shorter cut off time (2:45 hours) compared to GDAS.
    - NDAS(ndas) covers the North America and has a longer cut off time than NAM, which means it includes more real-time data than NAM.

- NAM(nam) covers the North America but has a shorter cut off time comparing to others.
- Data quality control processes for PrepBUFR files in each observation system are different but their results are reflected as quality markers, which can be easily checked by decoding the specific PrepBUFR file.
- For data types in each PrepBUFR file, please check the following section.

- Code table for PrepBUFR report types

  The complete list of the conventional observation types (and their BUFR codes) used by each NCEP operation system are documented at the following links:

  Global GFS and GDAS GSI analyses:

  http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_2.htm

  Global CDAS/reanalysis systems:

  http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_3.htm

  Regional NAM and NDAS GSI analyses:

  http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_4.htm

  Rapid Update Cycle (RUC) 3DVAR analysis:

  http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_5.htm

  Here we give a simplified table for the most commonly used data types:

Observation data used by GSI

Observation Type for T, q, Ps	Code is from 100 to 199
Rawinsonde+ Dropsonde	120+122
Synop+ METAR	181+187
Ship+BUOY	180
AIRCFT	130 AIREP and PIREP aircraft
ADPUPA	132 flight level reconnaissance and profile dropsonde
Observation Type for UV and speed	Code is from 200 to 299
Rawinsonde	220
Profiler	223
VAD	224
AIRCFT	230: AIREP and PIREP aircraft flight level reconnaissance and profile dropsonde
ADPUPA	232
SATWIND	245-NESDIS IR cloud drift 246-NESDIS imager water vapor
Ship+ BUOY	280
ATLAS BUOY	282

- Default observation file names used in GSI

GSI can analyze many types of observational data, including conventional data, satellite observations (such as AIRS, IASI, GPSRO (bending angle or refractivity), SBUV/2 ozone, GOME ozone, GOES sounder, GOES imager, AVHRR), and radar data. For use with GSI, these observations are saved in the BUFR format (with NCEP specified features). A list of the default observation file names used in GSI and the corresponding observations included in the files is provided in the table below:

GSI Name	Content	Example file names
prepbufr	Conventional observations, including ps, t, q, pw, uv, spd, dw, sst, from observation platforms such as METAR, sounding, et al.	gdas1.t12z.prepbufr
amsuabufr	AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17,18, 19 and METOP-A	gdas1.t12z.1bamua.tm00.bufr_d
amsubbufr	AMSU-B 1b radiance (brightness temperatures) from satellites NOAA15, 16,17	gdas1.t12z.1bamub.tm00.bufr_d
radarbufr	Radar radial velocity Level 2.5 data	ndas1.t12z. radwnd. tm12.bufr_d
gpsrobufr	GPS radio occultation observation	gdas1.t12z.gpsro.tm00.bufr_d
ssmirrbufr	Precipitation rate observations fromSSM/I	gdas1.t12z.spssmi.tm00.bufr_d
tmirrbufr	Precipitation rate observations from TMI	gdas1.t12z.sptrmm.tm00.bufr_d
sbuvbufr	SBUV/2 ozone observations from satellite NOAA16, 17, 18, 19	gdas1.t12z.osbuv8.tm00.bufr_d
hirs2bufr	HIRS2 1b radiance from satellite NOAA14	gdas1.t12z.1bhrs2.tm00.bufr_d
hirs3bufr	HIRS3 1b radiance observations from satellite NOAA16, 17	gdas1.t12z.1bhrs3.tm00.bufr_d
hirs4bufr	HIRS4 1b radiance observation from satellite NOAA 18, 19 and METOP-A	gdas1.t12z.1bhrs4.tm00.bufr_d
msubufr	MSU observation from satgellite NOAA 14	gdas1.t12z.1bmsu.tm00.bufr_d
airsbufr	AMSU-A and AIRS radiances from satellite AQUA	gdas1.t12z.airsev.tm00.bufr_d
mhsbufr	Microwave Humidity Sounder observation from NOAA 18 and METOP-A	gdas1.t12z.1bmhs.tm00.bufr_d
ssmitbufr	SSMI observation from satellite f13, f14, f15	gdas1.t12z.ssmit.tm00.bufr_d
amsrebufr	AMSR-E radiance from satellite AQUA	gdas1.t12z.amsre.tm00.bufr_d
ssmisbufr	SSMIS radiances from satellite f16	gdas1.t12z.ssmis.tm00.bufr_d
gsnd1bufr	GOES sounder radiance (sndrd1, sndrd2, sndrd3 sndrd4) from GOES 11, 12, 13.	gdas1.t12z.goesfv.tm00.bufr_d
l2rwbufr	NEXRAD Level 2 radial velocity	ndas.t12z.nexrad. tm12.bufr_d
gsndrbufr	GOES sounder radiance from GOES11, 12	gdas1.t12z.goesnd.tm00.bufr_d
gimgrbufr	GOES imager radiance from GOES 11, 12	gdas1.t12z.?????.tm00.bufr_d
omibufr	Ozone Monitoring Instrument (OMI) observation NASA Aura	gdas1.t12z.omi.tm00.bufr_d
iasibufr	Infrared Atmospheric Sounding Interfero-meter sounder observations from METOP-A	gdas1.t12z.mtiasi. tm00.bufr_d
gomebufr	The Global Ozone Monitoring Experiment (GOME) ozone observation from METOP-A	gdas1.t12z.syndata.tcvitals.tm00
mlsbufr	Aura MLS stratospheric ozone data	gdas1.t12z. mlsbufr.tm00.bufr_d
tcvitl	Synthetic Tropic Cyclone-MSLP observation	gdas1.t12z.gome.tm00.bufr_d
modisbufr	MODIS aerosol total column AOD observations from AQUA and TERRA	

5.3 BUFR/PrepBUFR Data Resources for Community Users

There are several sources to get real-time and archived atmospheric observations and model forecasts. Some of them provide NCEP operation BUFR/PrepBUFR files for community. Below is a list we are aware of. Users are welcome to send us new data source links to share with the community.

Data in BUFR format

- NCEP NOMADS Site:
  - PrepBufr for GDAS (Global) - 1 month buffer:
    http://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/
  - PrepBufr for NDAS (North America) - 1 month buffer:
    http://nomads.ncep.noaa.gov/pub/data/nccf/com/nam/prod/

- NCEP FTP Site:
  - PrepBufr for GDAS (Global) - 3 day buffer:
    ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/
  - PrepBufr for NDAS (North America) - 3 day buffer:
    ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/nam/prod/

- NCDC NOMADS Site:
  - PrepBufr for GDAS (Global) - archive starting May 2007:
    http://nomads.ncdc.noaa.gov/data/gdas/

- NCAR Computational and Information Systems Laboratory (CISL) Research Data Archive (RDA) Site:
  - **DS337.0**: NCEP ADP Global Upper Air and Surface Observations (PREPBUFR and NetCDF PB2NC Output) - archive starting May 1997:
    http://dss.ucar.edu/datasets/ds337.0/
  - **DS337.0 Subset**: Interactive tool for running PB2NC over a specified time period and geographic region:
    http://dss.ucar.edu/datasets/ds337.0/forms/337_subset.php

Data in other formats

- Unidata Program
  http://www.unidata.ucar.edu/data/

- National Environmental Satellite, Data, and Information Service (NESDIS)
  http://www.nesdis.noaa.gov/

- National Climatic Data Center
  http://www.ncdc.noaa.gov/oa/ncdc.html

- MADIS Surface Data
  http://www-frd.fsl.noaa.gov/mesonet/

- NCAR Mass Storage System (MSS)
  Real-time and archived data are available to users who have UCAR computer accounts.

Reference and Useful Links:

- WMO INTERNATIONAL CODES
  http://www.wmo.int/pages/prog/www/WMOCodes.html

- Standard table from WMO: BUFR Table A, B, C, D
  http://www.wmo.int/pages/prog/www/WMOCodes/TDCFtables.html#TDCFtables

- WMO BUFR guide:
http://www.wmo.int/pages/prog/www/WMOCodes/Guides/BUFRCREXPreface_en.html

- NCEP/NCO web to introduce BUFRLIB and how to decode and encode BUFR files:
  http://www.nco.ncep.noaa.gov/sib/decoders/BUFRLIB/

- BUFR documents also can be found in NCAR DSS:
  http://dss.ucar.edu/docs/formats/bufr/

- PREPBUFR documents at NCEP:
  http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/

- Observation data processing at NCEP (by Dennis Keyser):
  http://www.emc.ncep.noaa.gov/mmb/data_processing/data_processing/

Appendix

PrepBUFR decoding example for read_repbufr.f90:

```
program read_prepbufr

  use kinds, only: r_single,r_kind,r_double,i_kind

  implicit none

  character(len=80) :: infile,obstype

! Declare local variables

  character(40) drift,hdstr,qcstr,oestr,sststr,satqcstr,levstr,hdstr2
  character(40) metarcldstr,geoscldstr,metarvisstr,metarwthstr
  character(80) obstr
  character(10) date
  character(8) subset
  character(8) prvstr,sprvstr

  integer(i_kind) ireadmg,ireadsb,icntpnt,icntpnt2,icount,iiout
  integer(i_kind) lunin,i,maxobs,j,idomsfc,itemp,it29
  integer(i_kind) metarcldlevs,metarwthlevs
  integer(i_kind) k,nmsg,kx, nreal,idate,iret,ncsave,levs
  integer(i_kind) ntb

  real(r_double) vtcd
  real(r_double),dimension(8):: hdr
  real(r_double),dimension(8,255):: drfdat,qcmark,obserr
  real(r_double),dimension(9,255):: obsdat
  real(r_double),dimension(8,1):: sstdat
  real(r_double),dimension(2,10):: metarcld
  real(r_double),dimension(1,10):: metarwth
  real(r_double),dimension(1,1) :: metarvis
  real(r_double),dimension(4,1) :: geoscld
  real(r_double),dimension(1):: satqc
  real(r_double),dimension(1,1):: r_prvstg,r_sprvstg
  real(r_double),dimension(1,255):: levdat
  real(r_double),dimension(255,20):: tpc
  real(r_double),dimension(2,255,20):: tobaux

!  data statements
  data hdstr  /'SID XOB YOB DHR TYP ELV SAID T29'/
  data hdstr2 /'TYP SAID T29 SID'/
  data obstr  /'POB QOB TOB ZOB UOB VOB PWO CAT PRSS' /
  data drift  /'XDR YDR HRDR                    '/
  data sststr /'MSST DBSS SST1 SSTQM SSTOE           '/
  data qcstr  /'PQM QQM TQM ZQM WQM NUL PWQ     '/
  data oestr  /'POE QOE TOE NUL WOE NUL PWE     '/
  data satqcstr  /'QIFN'/
  data prvstr /'PRVSTG'/
  data sprvstr /'SPRVSTG'/
  data levstr  /'POB'/
  data metarcldstr /'CLAM HOCB'/      ! cloud amount and cloud base height
  data metarwthstr /'PRWE'/           ! present weather
  data metarvisstr /'HOVI'/           ! visibility
  data geoscldstr /'CDTP TOCC GCDTT CDTP_QM'/   ! NESDIS cloud products: cloud top
pressure, temperature,amount

  logical tob,qob,uvob,spdob,sstob,pwob,psob
  logical metarcldobs,geosctpobs
  logical driftl,convobs

  data lunin / 13 /

! Initialize variables
```

```
  infile='prepbufr'

  nreal=0
  satqc=0.0_r_kind
  obstype='t'
  tob = obstype == 't'
  uvob = obstype == 'uv'
  spdob = obstype == 'spd'
  psob = obstype == 'ps'
  qob = obstype == 'q'
  pwob = obstype == 'pw'
  sstob = obstype == 'sst'
  metarcldobs = obstype == 'mta_cld'
  geosctpobs = obstype == 'gos_ctp'
  convobs = tob .or. uvob .or. spdob .or. qob

!-------------------------------------------------------------------------
! Open, then read date from bufr data
  call closbf(lunin)
  open(lunin,file=infile,form='unformatted')
  call openbf(lunin,'IN',lunin)
  call datelen(10)

  maxobs=0
  nmsg=0
  ntb = 0
  msg_report: do while (ireadmg(lunin,subset,idate) == 0)
     nmsg=nmsg+1
     loop_report: do while (ireadsb(lunin) == 0)
       ntb = ntb+1

!      Extract type information
       call ufbint(lunin,hdr,4,1,iret,hdstr2)
       kx=hdr(1)

!      For the satellite wind to get quality information and check if it will be used
       if( kx ==243 .or. kx == 253 .or. kx ==254 ) then
          call ufbint(lunin,satqc,1,1,iret,satqcstr)
          if(satqc(1) <  85.0_r_double) cycle loop_report   ! QI w/o fcst (su's setup
       endif

!  Save information for next read

!  Save information for next read
       if(ncsave /= 0) then
          call ufbint(lunin,levdat,1,255,levs,levstr)
          maxobs=maxobs+max(1,levs)
       end if

    end do loop_report
  enddo msg_report
  if (nmsg==0) goto 900
  write(6,*)'READ_PREPBUFR: messages/reports = ',nmsg,'/',ntb

!-------------------------------------------------------------------------

! Obtain program code (VTCD) associated with "VIRTMP" step
  if(tob)call ufbqcd(lunin,'VIRTMP',vtcd)

! loop over convinfo file entries; operate on matches
!DTC comment out the loop loop_convinfo because we want to read all typies
!DTC  loop_convinfo: do nx=1, ntread

     call closbf(lunin)
     open(lunin,file=infile,form='unformatted')
     call openbf(lunin,'IN',lunin)
     call datelen(10)

!    Big loop over prepbufr file
```

```
     ntb = 0
     nmsg = 0
     icntpnt=0
     icntpnt2=0
     loop_msg: do while (ireadmg(lunin,subset,idate)== 0)

       loop_readsb: do while(ireadsb(lunin) == 0)
!       use msg lookup table to decide which messages to skip
!       use report id lookup table to only process matching reports
        ntb = ntb+1

!       Extract type, date, and location information
        call ufbint(lunin,hdr,8,1,iret,hdstr)

!       Balloon drift information available for these data
!DTC        driftl=kx==120.or.kx==220.or.kx==221

!       Extract data information on levels
        call ufbint(lunin,obsdat,9,255,levs,obstr)
        call ufbint(lunin,qcmark,8,255,levs,qcstr)
        call ufbint(lunin,obserr,8,255,levs,oestr)
        if(sstob)then
           sstdat=1.e11_r_kind
           call ufbint(lunin,sstdat,8,1,levs,ststr)
        else if(metarcldobs) then
          metarcld=1.e11_r_kind
          metarwth=1.e11_r_kind
          metarvis=1.e11_r_kind
          call ufbint(lunin,metarcld,2,10,metarcldlevs,metarcldstr)
          call ufbint(lunin,metarwth,1,10,metarwthlevs,metarwthstr)
          call ufbint(lunin,metarvis,1,1,iret,metarvisstr)
          if(levs /= 1 ) then
            write(6,*) 'READ_PREPBUFR: error in Metar observations, levs sould be 1 !!!'
             stop 110
          endif
        else if(geosctpobs) then
           geoscld=1.e11_r_kind
           call ufbint(lunin,geoscld,4,1,levs,geoscldstr)
        endif

!       If temperature ob, extract information regarding virtual
!       versus sensible temperature
!        if(tob) then
!           call ufbevn(lunin,tpc,1,255,20,levs,'TPC')
!           if (.not. twodvar_regional .or. .not.tsensible) then
!           else         !peel back events to store sensible temp in case temp is virtual
!              call ufbevn(lunin,tobaux,2,255,20,levs,'TOB TQM')
!           end if
!        end if

       end do loop_readsb

!
!   End of bufr read loop
     enddo loop_msg
!    Close unit to bufr file
     call closbf(lunin)

! Normal exit

!DTC  enddo loop_convinfo! loops over convinfo entry matches

900 continue
  close(lunin)

end program read_prepbufr
```

BUFR decoding example for read_airs.f90:

```
program read_airs
!                .      .    .                                       .
! subprogram:    read_airs                  read bufr format airs data

 use kinds, only: r_kind,r_double,i_kind

! Number of channels for sensors in BUFR
 integer(i_kind),parameter :: n_airschan = 281  !--- 281 subset ch out of
2378 ch for AIRS
 integer(i_kind),parameter :: n_amsuchan =  15
 integer(i_kind),parameter :: n_hsbchan  =   4
 integer(i_kind),parameter :: n_totchan  = n_amsuchan+n_airschan+n_hsbchan+1

! BUFR file sequencial number
 character(len=512)  :: table_file
 integer(i_kind)     :: lnbufr = 10
 integer(i_kind)     :: lnbufrtab = 11
 integer(i_kind)     :: irec,isub,next

! Variables for BUFR IO
 real(r_double),dimension(2) :: aquaspot
 real(r_double),dimension(12,3) :: allspot
 real(r_double),dimension(n_totchan) :: allchan

 character(len=8)  :: subset
 character(len=80) :: allspotlist
 integer(i_kind)   :: iret, ireadmg,ireadsb

 logical           :: airstab
 character(len=80) :: infile

 infile='airsbufr'

 allspotlist='SIID YEAR MNTH DAYS HOUR MINU SECO CLATH CLONH SAZA BEARAZ FOVN'

! Open BUFR file
 open(lnbufr,file=infile,form='unformatted')

! Open BUFR table
 table_file = 'airs_bufr.table'      ! make table file name
 inquire(file=table_file,exist=airstab)
 if (airstab) then
    open(lnbufrtab,file=trim(table_file))

 else
    call openbf(lnbufr,'IN',lnbufr)
 endif
 call datelen(10)

! Big loop to read data file
 next=0
 do while(ireadmg(lnbufr,subset,idate)>=0)
    next=next+1
    read_loop: do while (ireadsb(lnbufr)==0)

!       Read AIRSSPOT , AMSUSPOT and HSBSPOT

       call ufbrep(lnbufr,allspot,12,3,iret,allspotlist)

       if(iret /= 3) cycle read_loop

!       Check that number of airs channel equals n_airschan
```

```
!       only done until they match for one record and ndata is updated

!        Read AIRSCHAN or AMSUCHAN or HSBCHAN

        call ufbrep(lnbufr,allchan,1,n_totchan,iret,'TMBR')

        if( iret /= n_totchan)then
           write(6,*)'READ_AIRS:  ### ERROR IN READING ', ' BUFR DATA:', &
                iret, ' CH DATA IS READ INSTEAD OF ',n_totchan
           cycle read_loop
        endif

!        Read AQUASPOT
        call ufbint(lnbufr,aquaspot,2,1,iret,'SOZA SOLAZI')

    enddo read_loop
 enddo

 call closbf(lnbufr)  ! Close bufr file

end program read_airs
```