Accessing GRIB and BUFR data

Accessing GRIB and BUFR data.

J.D. Chambers

May 1994 Original version.
These routines were developed in the Meteorological Application Section at the European Centre for Medium-range Weather Forecasts, Reading, U.K.

John D. Chambers


Designed and printed at ECMWF.
Overview ........................................................................... 7
  FORTRAN subroutines ........................................ 7
  C functions ...................................................... 8
  Libraries ........................................................ 8

Specification of FORTRAN subroutines ......................... 9
  PBOPEN ......................................................... 9
  PBGRIB ......................................................... 9
  PBBUFR ......................................................... 10
  PBCLOSE ......................................................... 11

Specification of file handling FORTRAN subroutines .13
  PBSEEK ......................................................... 13
  PBREAD ......................................................... 13
  PBWRITE ......................................................... 15

Specification of C functions .................................. 17
  readgrib ..................................................... 17
  readbufr ..................................................... 17
  readnext ..................................................... 18

A sample FORTRAN program .................................. 21
Overview

FORTRAN subroutines

There are a number of subroutines which can be called from FORTRAN to handle products in an unblocked binary file:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBOPEN</td>
<td>To open a file. This supplies a file pointer which must be used by the other routines.</td>
</tr>
<tr>
<td>PBGRIB</td>
<td>To read GRIB products</td>
</tr>
<tr>
<td>PBBUFR</td>
<td>To read BUFR products</td>
</tr>
<tr>
<td>PBCLOSE</td>
<td>To close a file.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSEEK</td>
<td>To position a file</td>
</tr>
<tr>
<td>PBREAD</td>
<td>To read a given number of bytes from a file</td>
</tr>
<tr>
<td>PBREAD2</td>
<td>To read a given number of bytes from a file (different behaviour from PBREAD on end-of-file)</td>
</tr>
<tr>
<td>PWRITE</td>
<td>To write a given number of bytes to a file</td>
</tr>
</tbody>
</table>

There are lower-level routines which provide a FORTRAN-callable interface to file handling; it is not normally necessary to use these routines for reading BUFR or GRIB products:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSEEK</td>
<td>To position a file</td>
</tr>
<tr>
<td>PBREAD</td>
<td>To read a given number of bytes from a file</td>
</tr>
<tr>
<td>PBREAD2</td>
<td>To read a given number of bytes from a file (different behaviour from PBREAD on end-of-file)</td>
</tr>
<tr>
<td>PWRITE</td>
<td>To write a given number of bytes to a file</td>
</tr>
</tbody>
</table>

These subroutines are written in C and use standard C library functions for file handling (fopen, fclose, fseek, fread and fwrite).
C functions

There are three C functions to handle unblocked binary files.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>readgrib</td>
<td>To read GRIB products</td>
</tr>
<tr>
<td>readbufr</td>
<td>To read BUFR products</td>
</tr>
<tr>
<td>readnext</td>
<td>To read any mixture of GRIB and BUFR products.</td>
</tr>
</tbody>
</table>

These functions are written in C and use C standard library functions for file handling (fopen, fclose, fseek, fread and fwrite).

Libraries

On the workstations, the library /usr/local/lib/libemos.a contains the binary file reading and writing routines for GRIB and BUFR products.

The library name follows the normal UNIX convention and can be specified in the compile/link command using the usual method, for example:

```
cc -o program program.c -lemos
```
Specification of FORTRAN subroutines

PBOPEN

A subroutine which can be called from FORTRAN to open an unblocked binary file and return a suitable file pointer for use in calls to PBGRIB, PBBUFR, PBSEEK, PBREAD, PBREAD2 and PBWRITE.

The format and arguments for the subroutine are as follows:

SUBROUTINE PBOPEN (KUNIT, FILENAME, MODE, KRET)

Input parameters are CHARACTERS.

* FILENAME a character string describing the file.
* MODE a character string describing the mode of access to the file: r for read, w for write, a for append.

Output parameters are INTEGERS.

* KUNIT file pointer for the file.
* KRET Status return code.

Table 4: KRET from PBOPEN

<table>
<thead>
<tr>
<th>KRET</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK.</td>
</tr>
<tr>
<td>-1</td>
<td>Could not open file.</td>
</tr>
<tr>
<td>-2</td>
<td>Invalid file name.</td>
</tr>
<tr>
<td>-3</td>
<td>Invalid open mode specified</td>
</tr>
</tbody>
</table>

PBGRIB

A subroutine which can be called from FORTRAN to retrieve GRIB products from an unblocked binary file. The subroutine is an interface to a set of C routines which handle the binary file as a stream of bytes.

Products are delivered one per call to a user array. The format and arguments for the subroutine are as follows:

SUBROUTINE PBGRIB (KUNIT, KARRAY, KINLEN, KOUTLEN, KRET)

Input parameters are INTEGERS.

* KUNIT file pointer obtained from PBOPEN.
• KARRAY    array big enough to hold the GRIB product.
• KINLEN    size in BYTES of the array.

Output parameters are INTEGERS.
• KOUTLEN   actual size in BYTES of the GRIB product read into KARRAY.
• KRET       Status return code

Table 5: KRET from PBGRIB

<table>
<thead>
<tr>
<th></th>
<th>Product has been successfully read.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read.</td>
</tr>
<tr>
<td></td>
<td>(Note that EOF does not cause a program fail. This value</td>
</tr>
<tr>
<td></td>
<td>must be explicitly caught by the caller to avoid looping at</td>
</tr>
<tr>
<td></td>
<td>EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>There has been an error in the file-handling (e.g. the file</td>
</tr>
<tr>
<td></td>
<td>contains a truncated product).</td>
</tr>
<tr>
<td>-3</td>
<td>The size of KARRAY is not sufficient for the product.</td>
</tr>
</tbody>
</table>

PBBUFR

A subroutine which can be called from FORTRAN to retrieve BUFR products from an unblocked binary file. The subroutine is an interface to a set of C routines which handle the binary file as a stream of bytes.

Products are delivered one per call to a user array. The format and arguments for the subroutine are as follows:

SUBROUTINE PBBUFR(KUNIT,KARRAY,KINLEN,KOUTLEN,KRET)

Input parameters are INTEGERS.
• KUNIT    file pointer obtained from PBOPEN.
• KARRAY    array big enough to hold the product.
• KINLEN    size in BYTES of the array.

Output parameters are INTEGERS.
• KOUTLEN   actual size in BYTES of the product read into KARRAY.
• KRET       Status return code
Table 6: KRET from PBUFR

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Product has been successfully read.</td>
</tr>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>There has been an error in the file-handling (e.g. the file contains a truncated product).</td>
</tr>
<tr>
<td>-3</td>
<td>The size of KARRAY is not sufficient for the product.</td>
</tr>
</tbody>
</table>

PBCLOSE

A subroutine which can be called from FORTRAN to close an unblocked binary file previously opened with PBOPEN.

The format and arguments for the subroutine are as follows:

SUBROUTINE PBCLOSE(KUNIT,KRET)

Input parameter is an INTEGER.
- * KUNIT file pointer obtained from PBOPEN.

Output parameter is an INTEGER.
- * KRET Status return code

Table 7: KRET from PBCLOSE

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>error in handling the file.</td>
</tr>
</tbody>
</table>
Specification of file handling
FORTRAN subroutines

PBSEEK

A subroutine which can be called from FORTRAN to position an unblocked binary file at any desired byte position.

The format and arguments for the subroutine are as follows:

SUBROUTINE PBSEEK(KUNIT,KOFFSET,KSTART,KRET)

Input parameters are INTEGERS.

- **KUNIT** file pointer obtained from POPEN.
- **KOFFSET** number of bytes to offset the file; this is used as either an absolute or relative offset depending on the value of KSTART.
- **KSTART**
  - 0, if KOFFSET is an absolute count from the beginning of the file,
  - 1, if KOFFSET is a relative offset from the current byte position in the file,
  - 2, if KOFFSET is an absolute offset from the end of file.

Output parameter is an INTEGER.

- **KRET** Status return code.

Table 8: KRET from PBSEEK

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= 0</td>
<td>Byte offset from the start of file after positioning.</td>
</tr>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>Error in handling the file</td>
</tr>
</tbody>
</table>

PBREAD

A subroutine which can be called from FORTRAN to read a block of bytes from an unblocked binary file. (Note that this routine behaves differently from PBREAD2 on hitting end-of-file.)

The format and arguments for the subroutine are as follows:
SUBROUTINE PBREAD(KUNIT,KARRAY,KOUNT,KRET)

Input parameters are INTEGERS.

- KUNIT file pointer obtained from PBOPEN.
- KOUNT number of BYTES to read from the file.

Output parameters are INTEGERS.

- KARRAY an integer array to accept the bytes from the read
- KRET Status return code.

**Table 9: KRET from PBREAD**

<table>
<thead>
<tr>
<th>&gt;= 0</th>
<th>Number of bytes read from the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>Error in handling the file</td>
</tr>
</tbody>
</table>

PBREAD2

A subroutine which can be called from FORTRAN to read a block of bytes from an unblocked binary file. (Note that this routine behaves differently from PBREAD on hitting end-of-file.)

The format and arguments for the subroutine are as follows:

SUBROUTINE PBREAD2(KUNIT,KARRAY,KOUNT,KRET)

Input parameters are INTEGERS.

- KUNIT file pointer obtained from PBOPEN.
- KOUNT number of BYTES to read from the file.

Output parameters are INTEGERS.

- KARRAY an integer array to accept the bytes from the read.
• KRET Status return code

<table>
<thead>
<tr>
<th>Table 10: KRET from PBREAD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
</tr>
<tr>
<td>-2</td>
</tr>
</tbody>
</table>

PBWRITE

A subroutine which can be called from FORTRAN to write a block of bytes to an unblocked binary file.

The format and arguments for the subroutine are as follows:

SUBROUTINE PBWRITE(KUNIT,KARRAY,KOUNT,KRET)

Input parameters are INTEGERS.

• KUNIT file pointer obtained from PBOPEN.
• KARRAY an integer array holding the bytes for the write.
• KOUNT number of BYTES to write to the file.

Output parameter is an INTEGER.

• KRET Status return code

<table>
<thead>
<tr>
<th>Table 11: KRET from PBREAD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
</tr>
<tr>
<td>-1</td>
</tr>
</tbody>
</table>
Specification of C functions

readgrib

A C function which can retrieve GRIB products from an unblocked binary file.

The products are delivered one per call to a user array. The format and arguments for the function are as follows:

long readgrib(FILE *file, char *buffer, long *grib_prod_len)

where:

- file file pointer obtained from PBOPEN
- buffer buffer big enough to hold the product, or NULL
- grib_prod_len size of the buffer on input; becomes size in BYTES of the product read. If end-of-file is hit, the value is returned unchanged (i.e. when the function return code is -1).

Table 12: readgrib return value

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Product has been successfully read</td>
</tr>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>Error in the file-handling (e.g. if the file contains a truncated product)</td>
</tr>
<tr>
<td>-3</td>
<td>The size of buffer is not sufficient for the product. In this case, the buffer is filled with bytes from the product and the file is positioned at the beginning of the following product, if any.</td>
</tr>
<tr>
<td>-4</td>
<td>The user buffer is too small to allow any processing of the product at all (e.g. less than 8 bytes long)</td>
</tr>
</tbody>
</table>

If the second parameter is a NULL pointer, the function will pass back the length of the next product and will position the file on the byte following the end of the product.

readbufr

A C function which can retrieve BUFR products from an unblocked binary file.
Products are delivered one per call to a user array. The format and arguments for the function are as follows:

```c
long readbufr(FILE *file, char *buffer, long *bufr_prod_len)
```

where:

- **file** file pointer obtained from POPEN
- **buffer** buffer big enough to hold the product, or NULL
- **bufr_prod_len** size of the buffer on input; becomes size in BYTES of the product read. If end-of-file is hit, the value is returned unchanged (i.e. when the function return code is -1).

**Table 13: readbufr return value**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Product has been successfully read</td>
</tr>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-2</td>
<td>Error in the file-handling (e.g. if the file contains a truncated product)</td>
</tr>
<tr>
<td>-3</td>
<td>The size of buffer is not sufficient for the product. In this case, the buffer is filled with bytes from the product and the file is positioned at the beginning of the following product, if any.</td>
</tr>
<tr>
<td>-4</td>
<td>The user buffer is too small to allow any processing of the product at all (e.g. less than 8 bytes long)</td>
</tr>
</tbody>
</table>

If the second parameter is a NULL pointer, the function will pass back the length of the next product and will position the file on the byte following the end of the product.

**readnext**

A C function which can retrieve GRIB or BUFR products from an unblocked binary file.

Products are delivered one per call to a user array. The format and arguments for the function are as follows:
long readnext(char * buffer, long * prod_len, long (*read_func)(char *, long, void *), void * stream)

where:
- buffer buffer big enough to hold the product, or NULL. I
- prod_len size of the buffer on input, becomes size in BYTES of the product read. If the end-of-file is hit, this value is returned unchanged (i.e. when the function return code is -1).
- read_func function to read input stream
- stream data for read_function, e.g. file pointer returned from PBOPEN

If the first parameter is a NULL pointer, the function will pass back the length of the next product and position the file on the byte following the end of the product.

**Table 14: readnext return value**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Product has been successfully read</td>
</tr>
<tr>
<td>-1</td>
<td>End-of-file is hit before a product is read. (Note that EOF does not cause a program fail. This value must be explicitly caught by the caller to avoid looping at EOF)</td>
</tr>
<tr>
<td>-3</td>
<td>The size of buffer is not sufficient for the product. In this case, the buffer is filled with bytes from the product and the file is positioned at the beginning of the following product, if any.</td>
</tr>
<tr>
<td>-4</td>
<td>The user buffer is too small to allow any processing of the product at all (e.g. less than 8 bytes long)</td>
</tr>
</tbody>
</table>
A sample FORTRAN program

PROGRAM MAINPRG
**** MAINPRG -

Purpose.
-------

To read BUFR products from an unblocked binary file and write
them out to another file.

Interface.
--------

PB FORTRAN-callable subroutines

Method.
------

Open input file of BUFR products; filename bufr.u.
Open new output file; filename bufr.ext.
Read up to 500 products or until end-of-file.
Write each product to output file.

Exterals.
-------

PBOPEN
PBBUFR
PBWRITE
PBCLOSE

Reference.
-------


Comments.
--------

Program is an example of how to make use of the PB routines.

Author.
------

J.D.Chambers ECMWF 12:05:94.
A sample FORTRAN program

---
C
C* Section 0. Definition of variables. Data statements.
---
C
IMPLICIT NONE
C
INTEGER IRET, ILEN, IBUFF, IBFLEN, IUNIT1, IUNIT2, ILOOP
C
CHARACTER*1 MODER, MODEW, MODEA
CHARACTER*24 FILNM
CHARACTER*24 NEWFL
C
DIMENSION IBUFF(50000)
C
DATA IBFLEN/200000/
DATA MODER/'r', MODEW/'w', MODEA/'a'/
DATA FILNM/'bufr.u'/
DATA NEWFL/'bufr.ext'/

---
C
C*
---
C* Section 1. Initial house-keeping.
---
C
100 CONTINUE
C
C Open - for read
CALL PBOPEN(IUNIT1, FILNM, MODER, IRET)
WRITE(*,*) 'After PBOPEN, IRET = ', IRET
IF ( IRET .NE. 0 ) GO TO 900

C Open - for write
CALL PBOPEN(IUNIT2, NEWFL, MODEW, IRET)
WRITE(*,*) 'After PBOPEN, IRET = ', IRET
IF ( IRET .NE. 0 ) GO TO 900
C
C* Go to section 9, if error in initial house-keeping.
C
C*
---
C* Section 2. Processing control.
---
C
200 CONTINUE
DO 250 ILOOP = 1,500
C
C Get the next BUFR product
CALL PBBUFRI(IUNIT1, IBUFF, IBFLEN, ILEN, IRET)
WRITE (*,*) 'BUFR product number = ', ILOOP
WRITE (*,*) 'Length of BUFR product = ', ILEN

---

22 16 May 1994 Accessing GRIB and BUFR data
WRITE (*,*) 'IRET = ', IRET

IF (IRET .EQ. -1 ) WRITE (*,*) 'EOF'
IF (IRET .EQ. -1 ) GO TO 300
IF (IRET .EQ. -2 ) WRITE (*,*) 'File handling problem'
IF (IRET .EQ. -3 ) WRITE (*,*) 'Array too small'
IF (IRET .NE. 0 ) GO TO 900

C Write product to new file
CALL PBWRITE(UNIT2,IBUFF,ILEN,IRET)
WRITE(*,*) 'After PBWRITE, IRET = ', IRET
IF (IRET .LT. 0 ) WRITE (*,*) 'Write failed'
IF (IRET .LT. 0 ) GO TO 900

250 CONTINUE

C
C -------------------------------------------------------------
C*    Section 3. Final house-keeping.
C -------------------------------------------------------------
C

300 CONTINUE

C Close files
CALL PBCLOSE(UNIT1,IRET)
WRITE(*,*) 'After PBCLOSE, IRET = ', IRET
IF (IRET .NE. 0 ) WRITE (*,*) 'Close failed'
IF (IRET .NE. 0 ) GO TO 900

CALL PBCLOSE(UNIT2,IRET)
WRITE(*,*) 'After PBCLOSE, IRET = ', IRET
IF (IRET .NE. 0 ) WRITE (*,*) 'Close failed'

C
C -------------------------------------------------------------
C*    Section 9. End or abort. Format statements.
C -------------------------------------------------------------
C

900 CONTINUE

C
C IF (IRET.NE.0) CALL ABORT
C 9001 FORMAT (1H,'MAINPRO: ')
C
STOP
END