INTERNAL DOCUMENT No. 319

The main runs and datasets of the Fine Resolution Antarctic Model Project (FRAM) Part III: The data extraction routines

T Hateley & B de Cuevas

1992
The main runs and datasets of the Fine Resolution Antarctic Model Project (FRAM) Part III: The data extraction routines

T Hateley & B de Cuevas

1992
### TITLE

The main runs and datasets of the Fine Resolution Antarctic Model Project (FRAM).
Part III: The data extraction routines.

### ABSTRACT

The output of the Fine Resolution Antarctic Model was stored at regular intervals during the model run and is available to researchers. This document describes the software interface developed to allow user access to the data for analysis and display purposes.

### KEYWORDS

- NUMERICAL MODELLING
- PROJECT - FRAM

### ISSUING ORGANISATION

Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB, UK.

Director: Colin Summerhayes DSc

Telephone Wormley (0428) 684141
Telex 858833 OCEANS G
Facsimile (0428) 683066

Copies of this report are available from: The Library, PRICE
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>4</td>
</tr>
<tr>
<td>2 DESCRIPTION OF THE ARCHIVE DATASET</td>
<td>4</td>
</tr>
<tr>
<td>3 THE EXTRACT PROGRAM</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Installation of the program</td>
<td>7</td>
</tr>
<tr>
<td>3.3 Input file</td>
<td>8</td>
</tr>
<tr>
<td>3.4 Output file</td>
<td>8</td>
</tr>
<tr>
<td>3.5 Running the program</td>
<td>9</td>
</tr>
<tr>
<td>4 FRAM USER INTERFACE SUBROUTINES</td>
<td>11</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>4.2 Description of the subroutines</td>
<td>11</td>
</tr>
<tr>
<td>4.2.1 High level routine</td>
<td>11</td>
</tr>
<tr>
<td>4.2.2 Medium level routines</td>
<td>12</td>
</tr>
<tr>
<td>4.2.3 Low level routine</td>
<td>15</td>
</tr>
<tr>
<td>4.2.4 Input and output routines</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Model depth array</td>
<td>17</td>
</tr>
<tr>
<td>4.4 COMMON blocks and variables</td>
<td>17</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>21</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>22 - 61</td>
</tr>
</tbody>
</table>

Appendix I  Fortran listing of the Extract program  22

1 Program EXTRACT  22

2 Setup routines used by program EXTRACT  31

Appendix II  Fortran listings of FRAM user interface subroutines  36
1. INTRODUCTION

The Fine Resolution Antarctic Model Project (FRAM) is a Community Research Project of the UK National Environmental Research Council, designed to set up, run and analyse the results of a fine resolution primitive equation model of the Southern Ocean (The FRAM group, 1991). It forms part of the UK contribution to the World Ocean Circulation Experiment. The model output was stored at regular intervals during the run and is available to researchers. This document gives the basic information required to use the data. Section 2 gives a description of the contents of the archive datasets. Section 3 describes the software interface developed to allow user access to the plotting programs developed at IOSDL to display the FRAM data. These programs are supplied with the data. Section 4 describes the subroutines which provide a user interface to the data for extraction and analysis purposes. Listings of the subroutines are given in the Appendices.

2. DESCRIPTION OF THE ARCHIVE DATASET

The FRAM archive datasets are stored on exabyte tapes at IOSDL and will shortly be available from the British Oceanographic Data Centre (BODC), Bidston. Each dataset contains data at one timestep in the model run.

A single file can be considered as a continuous string of 8-bit bytes. In the standard format adopted for the FRAM data, this file starts with a header section made up of ASCII characters, followed by the main dataset written as binary images in the computer's floating point data format. Integers are transformed to floating point before being archived. The format used for the exabyte tapes is the 32-bit (4 bytes) IEEE floating point format which is used internally by the Sun computers and many other mini-computers (eg. Silicon Graphics).

The information stored in the file is defined in detail in the header, a copy of which follows:

FRAM Archive dataset. Format 1

Model constants

&PARM
IMT=722, JMT=221, KM=32, NT=2, LSEG=7, NISLE=3, LBC=2, MSI=2
&END
Arrangement of storage on tape

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Length</th>
<th>Description in words</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Header</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header</td>
<td>16000 bytes</td>
<td>This header as ASCII characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16000 bytes arranged as 200 lines of 80 characters</td>
</tr>
<tr>
<td><strong>2. KONTR data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTIT</td>
<td>1</td>
<td>Model timestep (integer converted to floating point format).</td>
</tr>
<tr>
<td>TISEC</td>
<td>1</td>
<td>Time in seconds from start of run.</td>
</tr>
<tr>
<td>AREA</td>
<td>1</td>
<td>Area of model ocean.</td>
</tr>
<tr>
<td>VOLUME</td>
<td>1</td>
<td>Volume of model ocean.</td>
</tr>
<tr>
<td><strong>3. FIELDS data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>IMT*JMT</td>
<td>Stream function at ITT-1.</td>
</tr>
<tr>
<td>P</td>
<td>IMT*JMT</td>
<td>Stream function at ITT.</td>
</tr>
<tr>
<td>HR</td>
<td>IMT*JMT</td>
<td>Inverse of depth.</td>
</tr>
<tr>
<td>PIDB</td>
<td>IMT*JMT</td>
<td>Rate of change of stream function at ITT-1.</td>
</tr>
<tr>
<td>PID</td>
<td>IMT*JMT</td>
<td>Rate of change of stream function at ITT.</td>
</tr>
<tr>
<td>USTAR</td>
<td>IMT*JMT</td>
<td>USTAR array used to calculate pressure field.</td>
</tr>
<tr>
<td>VSTAR</td>
<td>IMT*JMT</td>
<td>VSTAR array used to calculate pressure field.</td>
</tr>
<tr>
<td>PRESS</td>
<td>IMT*JMT</td>
<td>Pressure field.</td>
</tr>
<tr>
<td>FINS</td>
<td>2<em>LSEG</em>JMT+4*NISLE</td>
<td>Indices converted to floating point.</td>
</tr>
<tr>
<td><strong>4. Slabs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>IMT<em>KM</em>NT</td>
<td>Tracer fields for the Jth slab.</td>
</tr>
<tr>
<td>U</td>
<td>IMT*KM</td>
<td>U velocity for the Jth slab.</td>
</tr>
</tbody>
</table>
V $\text{IMT}^*\text{KM}$ V velocity for the $j$th slab.

Sea-ice arrays included if MSI = 2

RNICE $\text{IMT}$ Number of ice levels converted to floating point.
SNOICE IMT*(MSI+6) Sea-ice array.

*** End of sea ice arrays ***

FKMT IMT KMT array converted to floating point.
WSX IMT Wind stress in the x-direction.

*** For the JMT-2 and JMT-1 rows the following data is included for open boundary ***

TB IMT*KM*NT Tracer fields for ITT-1.
UB IMT*KM U velocity for ITT-1.
VB IMT*KM V velocity for ITT-1.

*** End of additional data for JMT-2 and JMT-1 ***

FKMU IMT KMU array converted to floating point.
WSY IMT Wind stress in the y-direction.

*** End of slab data ***

In the header, the length shown for each array also indicates how that array is stored. Thus if the length of array $A$ is shown as IMT*KM then the array will be stored as $A(\text{IMT}, \text{KM})$. If the archive dataset contains space for the arrays for the sea ice model, the parameter MSI will have a value of 2, otherwise it will have a value of 0. If the model is stopped at an odd timestep, the position of the arrays FKMT, WSX and FKMU, WSY in the archive dataset will be exchanged. However, in practice this did not occur.

The USTAR, VSTAR and PRESS arrays will be zero everywhere if unset. The USTAR and VSTAR arrays were calculated after day 3256 (8 years 11 months) of the model run. As is usual with the Cox model, the $u$ and $v$ velocity fields stored in the archive data are the baroclinic part of the full velocity field. The barotropic velocity may be obtained from the stream function and
added to the baroclinic velocity to give the full horizontal velocity. Temperature refers to potential temperature in the model data and throughout this document.

The Exabyte driver software at Rutherford Appleton Laboratory requires the data to be packed in 16000 byte blocks. The last block is therefore padded with a dummy array of zeros.

3. THE EXTRACT PROGRAM

3.1 Introduction

The program, extract.f, has been written to provide a simple user interface to the data. Using the program, horizontal or vertical (north-south or east-west) sections can be extracted from the archive dataset in a form that can then be plotted by the FRAM plotting programs. The horizontal sections to be extracted are referenced by the level number from the model. The vertical sections are referenced by their longitude (±°E) or latitude value (+°S).

The program creates an output file which consists of a header, describing the data in the file, followed by 'ASCOUT' encoded data. Each output file may contain data for several sections. The output file is created in the current or working directory.

The program was written in Fortran 77 on a Sun 4 workstation under the UNIX operating system. It assumes that the dataset has first been copied to disk. The approximate size of the disk file is 96 Mbytes. A full listing of the program is given in Appendix I.

3.2 Installation of the program

The source code for the program should be copied to sub-directory src of directory fram_extract. The compiled program will be placed in sub-directory bin.

Change directory to fram_extract/src.

In the file extract.f, set the variable DEFDIR to the absolute pathname of the default directory to contain the FRAM datasets.

Type

    make

followed by

    make clean

These commands will compile the program and remove temporary files generated during compilation.

Then to your .login file, add the line:
setenv FRAM_EXTRACT "[name of directory containing fram_extract]"
and to your .cshrc file, add the line:

alias extract '$FRAM_EXTRACT/fram_extract/bin/extract'

After you next login, this will enable the program 'extract' to be run from any directory,
creating the output files in the current (working) directory.

3.3 Input file

When the program is run it asks for the name of an input file containing the FRAM archive data. The naming convention adopted for the FRAM archive datasets is `fxyyy.data`, where:

- **x** = `r` for the main model run
- **s** for the repeat run with sea ice and full surface forcing
- **yyyy** = model day number of archive dataset.

3.4 Output file

The program creates an output file name. The naming convention adopted for this file is `fxyyy.cards`, where:

- **x** =
  - `a` - salinity at constant latitude (longitude vs depth slices)
  - `b` - salinity at constant longitude (latitude vs depth slices)
  - `c` - salinity at constant depth (longitude vs latitude slices)
  - `d` - temperature at constant latitude
  - `e` - temperature at constant longitude
  - `f` - temperature at constant depth
  - `g` - u-velocity at constant latitude
  - `h` - u-velocity at constant longitude
  - `i` - u-velocity at constant depth
  - `j` - v-velocity at constant latitude
  - `k` - v-velocity at constant longitude
  - `l` - v-velocity at constant depth
  - `m` - USTAR field
  - `n` - VSTAR field
  - `o` - pressure field
  - `p` - stream function
\( s \) - ice fraction (i.e. % area of grid box covered by ice)
\( t \) - ice thickness

\( yyyy \) = model day number of archive dataset (same as input file).

Note: Each file may contain one or more slabs of data of the same type.

Example: \texttt{fc2191.cards} contains horizontal slabs of salinity from the end of 6 years (day 2191).

### 3.5 Running the program

On entry to the program, the user is prompted for the name of the input FRAM archive dataset and the type of data to be extracted. An output filename is created and the output stream is initialised. A header is written to the output file describing the data the file will contain. This information is needed by the FRAM plotting routines.

The user is prompted for the field variable (TRAC) and the type of section required (DEPVAR). The program then reads in the appropriate masking array from the archive dataset. The masking array contains information about the location of land and submerged land. Data is then extracted, transformed if necessary, and masked. The transformations are:

- **Salinity:** output salinity = model salinity * 1000 + 35.
- **Velocity:** output velocity = baroclinic velocity (from the slabs) + barotropic velocity (from the stream function).

The data are converted to 'ASCOUT' format, in which each number is represented by 2 to 5 ASCII characters, and sent to the output file.

An example of the use of the extract program is given below. Prompts from the program are in italics, and bold typeface is used to denote user input in the correct format. The character # is used to enclose comments.

To run extract:

```
extract
FRAM Data Extraction Program

Directory [<CR> to select default directory /data/fram1b/data ]

Enter name of input data file : fr2191.data

input file = /data/fram1b/data/fr2191.data
```
TRAC
1. Stream Function
2. USTAR
3. VSTAR
4. Pressure
5. Temperature
6. Salinity
7. U Velocity
8. V Velocity
9. Ice Fraction
10. Ice Thickness

Enter number of field required: 5

# Note: If the field is 1 - 4 the output file name will be displayed on the screen and the data will be extracted. The following is an example of the procedure for obtaining a tracer or velocity section from the slabs. #

DEPVAR
1. Vertical Section (E/W) - constant latitude
2. Vertical Section (N/S) - constant longitude
3. Horizontal Section - constant depth

# Note: For E-W sections the latitude value (°S) is converted to the correct I value #
# and for N-S sections the longitude value (±°E) is converted to the correct J value. #

Enter number of field required 1

No of slabs (MAX=8) : 1

Slab 1. Latitude = 70.0

output file = fa2191.cards

extracting slab 1

Note: A response of 0 to the TRAC or DEPVAR prompts will exit from the program.
4. USER INTERFACE SUBROUTINES

4.1 Introduction

This section describes the software developed to provide an interface between the FRAM archive datasets and the researcher. The software is based on routines developed for the NERC Ocean Modelling group on the ULCC CRAY. Because of the problems with using the BACKSPACE command with exabytes tapes and the relative slowness in reading the tapes, the software assumes that the archive datasets have first been copied to disk.

Data can be accessed from the archive datasets using the high level routine MGSDDAT, or the lower level routines described below. The physical reading of the data is carried out by the low level routine XREAD.

Certain COMMON blocks must be declared at the beginning of any program written to access the archive datasets. GRIPAR contains the main model variables (eg. IMT, the number of east-west grid points), and must be set up before calling any of the user routines. PLITTYP must be set before calling the high level routine MGSDDAT. It defines the data which MGSDDAT is to extract. IEBUFF and MGSCCR are used by low level routines. MGSCCR is set by medium level routines and should be set by user programs calling the low level routine XREAD. TIME and TSTEP are used by the output routine HEADER2 and should be set up before calling this routine. (TIME is set by a call to READ40).

A brief description of the user routines is given in section 4.2. Details of the COMMON blocks and the definitions and values of the variables in them, are given in the following section. Complete listings of the routines are in Appendix II.

4.2 Description of the subroutines

4.2.1 High level routine

This comprises a single call within a program to read 2-dimensional arrays of selected variables. These can be stream function, pressure and the other 2-dimensional model arrays or latitude, longitude or depth slices of the 3-dimensional tracer or velocity fields. Before calling the routine, the variables in COMMON blocks GRIPAR and PLITTYP must be set correctly.

SUBROUTINE MGSDDAT(A, B, C, HR, ZDZ)

ZDZ (KM) - on entry, this contains the ZDZ array as defined in the Cox model (vertical position of bottom of levels).

A(IMT, JMT) - on exit, array for model data
B(IMT,JMT) - on exit, array for model data
C(IMT,JMT) - on exit, stream function data
HR(IMT,JMT) - on exit, this contains the reciprocal of total depth at U,V points

The actions carried out by the routine depend on the values set for the variables IDIR, ITYP and INDEX in COMMON block PLTTYP. ITYP specifies the field required and IDIR the orientation of the section. IDIR is only valid for ITYP = 1 or 2.

ITYP = 0 subroutine reads the stream function into array C. IDIR should equal 0.
1 subroutine reads U into A, V into B, the stream function into C, and the reciprocal depth field into HR.
2 subroutine reads temperature into A and salinity into B.
3 subroutine reads USTAR into A and VSTAR into B.

IDIR = 0 subroutine reads a horizontal section of data from depth level INDEX
1 subroutine reads an east-west section of data from row INDEX. The fields are stored in A(IMT,KM) and B(IMT,KM).
2 subroutine reads a north-south section of data from column INDEX. The fields are stored in A(JMT,KM) and B(JMT,KM).

INDEX defines the level, row or column required.

4.2.2 Medium level routines

These are designed for moving round and reading in parts of the full archive dataset in an efficient manner. For example, to calculate the velocity field on density surfaces, one might first call the routine to read the stream function and then move systematically through the slab fields reading the required temperature, salinity and velocity data, calculating the density and interpolating the velocity field in the process. By reading through the full dataset only once, during even the most complex calculations, programs using these routines can be more efficient than programs which only use calls to the high level subroutine MGSDAT.

The routines need the integer constants in COMMON block GRIPAR to be set before they are called. A number of them modify the position of the pointer MWORD which points to the next word to be read from the file.

SUBROUTINE READ40()
This must only be called immediately after opening the archive dataset. It reads the variables RITT, TTSEC, AREA, VOLUME from the dataset and puts ITT (integer value of RITT) into COMMON block PLTTYP and TTSEC into COMMON block TIME.
SUBROUTINE MGSADV(ISECT)
This moves the pointer MWORD to the beginning of the section of data defined by ISECT:
ISECT = 1 - start of header
       2 - variable RTTT (ITT in floating point format)
       3 - start of stream function
       4 - start of HR array
       5 - start of USTAR, VSTAR arrays
       6 - start of PRESS array
       7 - start of slabs.

SUBROUTINE MGSRDS(C)
This moves the pointer MWORD to, and reads, the stream function.
C (IMT, JMT) - array for data.

SUBROUTINE MGSRDH(HR)
This moves the pointer MWORD to, and reads, the HR array.
HR(IMT, JMT) - array for data.

SUBROUTINE MGSRDP(A)
This moves the pointer MWORD to, and reads, the pressure array.
A(IMT, JMT) - array for data.

SUBROUTINE MGSRDU(A, B)
This moves the pointer MWORD to, and reads, the USTAR and VSTAR arrays.
A(IMT, JMT) - array for USTAR data
B(IMT, JMT) - array for VSTAR data.

SUBROUTINE MGFKM1(FKMP)
This reads the masking array for tracer points.
FKMP(IMT, JMT) - array for masking data.

SUBROUTINE MGFKM2(FKMQ)
This reads the masking array for velocity points.
FKMQ(IMT, JMT) - array for masking data.
SUBROUTINE MGSRD0(A, J)
This reads the data, for the surface level, from one of the 1-dimensional (IMT) sea-ice model sub-arrays (ie. ice fraction, ice thickness) in the current slab and places it into row J of the array A(IMT, JMT). On entry MWORD should point to the first word of the sub-array, on exit it points to the first word of the next sub-array.
A - array for data
J - Jth slab.

SUBROUTINE MGSRD1(A, J, INDEX)
This reads the data for level INDEX, from one of the 2-dimensional (IMT*KM) sub-arrays in the current slab (ie. T, U or V) and places it into row J of the array A(IMT, JMT). On entry MWORD should point to the first word of the sub-array, on exit it points to the first word of the next sub-array. By looping through the full set of slabs, this routine can be used to build up a horizontal section at depth level INDEX.
A - array for data
J - Jth slab
INDEX - level.

SUBROUTINE MGSRD2(A)
This reads a complete 2-dimensional (IMT, KM) sub-array from the current slab into A(IMT, KM). On entry MWORD should point to the first word of one of the 2-dimensional sub-arrays in the current slab. On exit it points to the first word of the next sub-array.
A - array for data.

SUBROUTINE MGSRD3(A, J, I)
This reads the data for column I from one of the 2-dimensional (IMT, KM) sub-arrays in the current slab and places it into column J of array A(IMT, KM). On entry MWORD should point to the first word of the sub-array. On exit it points to the first word of the next sub-array. By looping through the full set of slabs, this routine can be used to build up a north-south section.
A - array for data
J - Jth slab
I - Ith column.

SUBROUTINE MGSRDF(A, J)
This reads the data for one of the 1-dimensional (IMT) sub-arrays from the current slab and places it into row J of the array A(IMT, JMT). On entry MWORD should point to the first word of the sub-array. On exit it points to the first word of the next sub-array.
A - array for data
J - Jth slab.

SUBROUTINE MGSSK1(K)
This skips over K 2-dimensional (IMT, KM) sub-arrays, ie. K*IMT*KM words.
K - integer.

SUBROUTINE MGSSK2(K)
This skips over K 1-dimensional (IMT) sub-arrays, ie. K*IMT words.
K - integer.

SUBROUTINE MGSDZZ(DZ, DZZ, ZDZ, ZDZZ)
This is a useful routine which calculates the depth arrays DZZ, ZDZ and ZDZZ from the DZ array.
Array ZDZ needs to be set up correctly for use by routines which calculate the barotropic velocity.
DZ (KM) - array of level thicknesses
DZZ(KM+1) - vertical grid spacing between T,U,V points
ZDZ(KM) - vertical position of bottom of levels
ZDZZ (KM+1) - vertical position of T,U,V grid points.

4.2.3 Low level routine

This is the basic low level reading routine which is used by all the above high and medium level routines. It will usually not be called by a user's program.

SUBROUTINE XREAD(A, M, N, IFAIL)
This routine reads in N words of data and places them into array A. The pointer M is then increased by N.
A - array for data
M - starting point of data (words)
N - number of words of data to be read in
IFAIL - returns non-zero if subroutine fails.
4.2.4 Input and output routines

These routines open the input and output files and write the header and data to the output file.

SUBROUTINE INSTR(NUNIT, DIRN, INFIL, IFAIL)
This opens the input file as an unformatted, direct access file with a block size of 16000 bytes. It also initializes the pointers used by XREAD.

NUNIT - input stream
DIRN - directory containing the input data files
INFIL - name of input file
IFAIL - returns non-zero if subroutine fails.

SUBROUTINE OUTSTR(NUNIT, OUTFIL, IFAIL)
This opens an output file and associates it with fortran stream NUNIT.

NUNIT - output stream
OUTFIL - output filename
IFAIL - returns non-zero if subroutine fails.

SUBROUTINE OFILEN(TRAC, DEPVAR, OUTFIL, IFAIL)
This creates the output filename according to field variable TRAC and orientation variable DEPVAR, following the convention described in section 3.4.

TRAC      - 1. Stream Function
             2. USTAR
             3. VSTAR
             4. Pressure
             5. TempTerature
             6. Salinity
             7. U Velocity
             8. V Velocity
             9. Ice Fraction
            10. Ice Thickness

DEPVAR    - 1. East-west vertical section
             2. North-south vertical section
             3. Horizontal section
OUTFIL - output filename
IFAIL - returns non-zero if subroutine fails.

SUBROUTINE HEADER2(NUNIT, TRAC, DEPVAR, 'CD', NAMRUN)
This routine writes the header on the output file.
NUNIT - output stream
TRAC - field variable (as above)
DEPVAR - orientation variable (as above)
'CD' - format
NAMRUN - comment line in header of output file.

SUBROUTINE ASCOUT(ARRAY, IDIM, ID, JD, VMASK, NCHAR, NOUT)
This will encode a section of an array as sets of 'NCHAR' printable characters, and write as a
formatted card-image dump (using ASCII characters 0-9; A-Z; a-z; ; ).
ARRAY - contains the data to be encoded
IDIM - declared first dimension of 2-dimensional array, ARRAY
ID - actual first dimension of data in ARRAY
JD - actual second dimension of data in ARRAY
VMASK - array containing the 4 masking values
NCHAR - number of characters encoding each data point (2-5)
NUNIT - output stream.

4.3 Model depth array

The thicknesses of the model levels are stored in the DZ array, which is used to calculate
all the depth parameters calculated by subroutine MGSDZZ. The thicknesses (cms) of the 32
levels used in FRAM are given by the following DATA statement:
DATA DZ / 20.7 E2, 23.3 E2, 26.5 E2, 31.0 E2, 37.3 E2, 46.7 E2, 61.6 E2, 85.9 E2,

4.4 COMMON blocks and variables

The following COMMON blocks must be declared at the beginning of any user program.
COMMON /GRIPAR/ PSIDEG, DXDEG, PHIDEG, DYDEG, IMT, JMT, KM, NT, LSEG, NISLE,
The variables in COMMON block GRIPAR are set at the beginning of the program. Their definitions and values in the FRAM archive datasets are given in the following table:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSIDEG</td>
<td>-0.5</td>
<td>The longitude (+ east) of the western boundary of the model.</td>
</tr>
<tr>
<td>DXDEG</td>
<td>0.5</td>
<td>The east-west grid spacing of the model.</td>
</tr>
<tr>
<td>PHIDEG</td>
<td>-79.0</td>
<td>The latitude (+ north) of the southern wall of the model.</td>
</tr>
<tr>
<td>DYDEG</td>
<td>0.25</td>
<td>The north-south grid spacing of the model.</td>
</tr>
<tr>
<td>IMT</td>
<td>722</td>
<td>The number of grid points in the east-west direction.</td>
</tr>
<tr>
<td>JMT</td>
<td>221</td>
<td>The number of grid points in the north-south direction.</td>
</tr>
<tr>
<td>KM</td>
<td>32</td>
<td>The number of vertical levels in the model.</td>
</tr>
<tr>
<td>NT</td>
<td>2</td>
<td>The number of tracer variables (temperature and salinity).</td>
</tr>
<tr>
<td>LSEG</td>
<td>7</td>
<td>The maximum number of sets of start and end indices for vorticity.</td>
</tr>
<tr>
<td>NISLE</td>
<td>3</td>
<td>The number of islands.</td>
</tr>
<tr>
<td>LCYC</td>
<td>1</td>
<td>Non-zero for cyclic east-west boundary conditions.</td>
</tr>
<tr>
<td>LBC</td>
<td>2</td>
<td>The number of one dimensional arrays stored with each model slab (excluding sea ice arrays).</td>
</tr>
<tr>
<td>MSI</td>
<td>0 or 2</td>
<td>The maximum number of ice layers in the sea ice model. Equal to 0 if no sea ice arrays are present.</td>
</tr>
</tbody>
</table>

The variables in COMMON block PLTTYP are used in subroutine MGSDAT. They must be set in the user program. (ITT may be set by a call to READ4 immediately after opening the archive dataset.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITYP = 0</td>
<td>subroutine reads the stream function</td>
</tr>
<tr>
<td>1</td>
<td>subroutine reads U, V and the stream function.</td>
</tr>
</tbody>
</table>
2 subroutine reads tracers.
3 subroutine reads USTAR and VSTAR.
IDIR = 0 subroutine reads a horizontal section of data from depth level INDEX
1 subroutine reads an east-west section of data from row INDEX
2 subroutine reads a north-south section of data from column INDEX.
INDEX Constant I, J or K value of slab to be extracted.
ITT Model timestep.
IDIM Declared I-dimension of array sent to subroutine ASCOUT. (IMT for E-W sections and horizontal slabs, JMT for N-S sections).

The variables in COMMON block IEBUFF are set in subroutines INSTR, OUTSTR and used in subroutine XREAD. They normally need not be set by the user.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEBUF(4000)</td>
<td>Input buffer for 32-bit word input data.</td>
</tr>
<tr>
<td>LBUFF</td>
<td>Position in the input file of the first element of the current buffer.</td>
</tr>
<tr>
<td>MBUFF</td>
<td>Length of buffer (MBUFF = 4000).</td>
</tr>
<tr>
<td>IUNIT</td>
<td>Input stream.</td>
</tr>
<tr>
<td>OUNIT</td>
<td>Output stream.</td>
</tr>
</tbody>
</table>

The variable in COMMON block MGSCCR is modified in a number of subroutines. It normally need not be set by the user.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWORD</td>
<td>Pointer to the next word to be read from the file.</td>
</tr>
</tbody>
</table>

The variables in COMMON block TSTEP are used in subroutine HEADER2. If this is to be called, they should be set up by the user main program. In the FRAM archive datasets, NDLAS always equals NDFIR (=ITT) and NDINC is zero. This is because only one timestep is stored in each dataset.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDFIR</td>
<td>Timestep of first slab in output file.</td>
</tr>
<tr>
<td>NDLAS</td>
<td>Timestep of last slab in output file.</td>
</tr>
<tr>
<td>NDINC</td>
<td>Incremental timestep between slabs in output file.</td>
</tr>
</tbody>
</table>
The variable in COMMON block TIME is set in subroutine READ4 and used in subroutine OFILEN.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTSEC</td>
<td>Total elapsed time of model run in seconds.</td>
</tr>
</tbody>
</table>
REFERENCES

APPENDIX I  FORTRAN LISTINGS OF THE EXTRACT PROGRAM

1  PROGRAM EXTRACT

Program to create a 'cutout' file of one of the output fields of
the Fine Resolution Model, using a compressed data set.

Version 4.0  12/05/92  T. Hateley, IOSDL.

DEFDIR - default directory for the input data files
DIRN - directory used to open input file
INFIL - full input filename (including directory name)
OUTFIL - output filename

COMMON /IEBUFF/IEBUFF(4000), LBUFF, MBUFF, IUNIT, OUNIT
COMMON /PLTTYP/IDIR, ITYP, INDEX, IT, IDIM
COMMON /GRIPAR/PSIDEG, DXDEG, PHIDEG, DYDEG,
&     IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /TSTEP/ NDFIR, NDLAS, NDINC
COMMON /TIME/ TTSEC
DIMENSION FKMP(722, 221), FKMQ(722, 221)
INTEGER VALS(8)
REAL LAT(8), L0NG(8)
COMMON /LVALS/ TSLAB(8)
DIMENSION A(722,221), B(722,221), C(722,221), HR(722,221)
DIMENSION DZ(32), DZZ(33), ZDZ(32), ZDZZ(33)
CHARACTER NAMRUN*80
CHARACTER*9 DEPVAR
CHARACTER*12 DEFDIR
CHARACTER*15 TRAC
CHARACTER*512 DIRN
CHARACTER*1024 INFIL
CHARACTER*12 OUTFIL
REAL TDEP(722, 221), TL0NG(221,32)
REAL VMASK(4)
REAL SLARR(722, 221)
EXTERNAL SSOUR

RADIUS = 6370.ES
RADIANS = 57.29578
PSIDEG = -0.5
PHIDEG = -79.0
DXDEG = 0.5
DYDEG = 0.25
IMT = 722
JMT = 221
KM = 32
NT = 2
LSEG = 7
NISLE = 3
LBC = 2
MSI = 2

IMTM1 = IMT - 1
JMTM1 = JMT - 1
IMTM2 = IMT - 2
JMTM2 = JMT - 2

C

DEFDIR = '/data/fram3a/data'

PRINT *, **
PRINT *, 'FRAM Data Extraction Programme.'
PRINT *, '================================='

C

default directory containing the input data files

set the input file directory

CALL SETDIR(DEFDIR, DIRN)

 initialise the input stream

CALL INSTR(20, DIRN, INHL, IFAIL)

read the first 4 variables from the input file (INFIL)

CALL READ40

choose type of slices required

CALL SETUP(TRAC, DEPVAR, NAMRUN)

set output filename

CALL OFILEN(TRAC, DEPVAR, OUTFIL, IFAIL)

initialise the output stream

CALL OUTSTR(18, OUTFIL, IFAIL)
NDFIR = ITT
NDLAS = 1000000

C set DTTS and NDINC from ITT
C
IF (ITT .LE. 19440 .OR. (ITT .GT. 28296 .AND. ITT .LE. 32688)) THEN
   NDINC = 720
   DTTS = 1200
ELSE
   NDINC = 360
   DTTS = 2400
ENDIF

C set level depths arrays
C
CALL MGSDZZ (DZ, DZZ, ZDZ, ZDZZ)
C write header for ascout cutout file
C
CALL HEADER2(18, TRAC, DEPVAR, 'CD', NAMRUN)
C read in masking array
C
IF (TRAC(1:3) .EQ. 'STR') THEN
   CALL MGSRDH(HR)
ELSE IF (TRAC(1:3) .EQ. 'TEM' .OR. TRAC(1:3) .EQ. 'SAL') THEN
   CALL MGFKM1 (FKMP)
ELSE IF (TRAC(1:3) .EQ. 'U V' .OR. TRAC(1:3) .EQ. 'V V' .OR.
   & TRAC(1:3) .EQ. 'ICE') THEN
   CALL MGFKM2 (FKMQ)
ENDIF
C
C extract Stream Function, Pressure, USTAR, VSTAR
C
IF (TRAC(1:3) .EQ. 'STR') THEN
   C extract stream function

   WRITE (18, 900) ' '
   CALL MGSDAT(A, B, C, HR, ZDZ)
   SCL = 1.E12
   DO 10 I = 2, IMTM1
      DO 10 J = 2, JMT
         SLARR(I - 1, J - 1) = C(I, J) / SCL
         IF ((HR(I, J) LT 1. E-9) .AND.
            & (HR(I-1, J) LT 1. E-9) .AND.
            & (HR(I-1, J-1) LT 1. E-9)) THEN
            SLARR(I-1, J-1) = VMASK(1)
         ENDIF
      10 CONTINUE
   WRITE(18, 910) TTSEC, DTTS
CALL ASCOUT(SLARR, IMT, IMTM2, JMTM1, VMASK, 2, 18)
GOTO 999
ELSE IF (TRAC(1:3) .EQ. 'UST') THEN

extract USTAR array

WRITE (18, 900) '
CALL MGSDAT(A, B, C, HR, ZDZ)
WRITE(18, 910) TTSEC, DTTS
CALL ASCOUT(A, IMT, IMTM2, JMTM1, VMASK, 2, 18)
GOTO 999

ELSE IF (TRAC(1:3) .EQ. 'VST') THEN

extract VSTAR array

WRITE (18, 900) '
CALL MGSDAT(A, B, C, HR, ZDZ)
WRITE(18, 910) TTSEC, DTTS
CALL ASCOUT(B, IMT, IMTM2, JMTM1, VMASK, 2, 18)
GOTO 999

ELSE IF (TRAC(1:3) .EQ. 'PRE') THEN

extract PRESSURE array

WRITE (18, 900) '
CALL MGSDAT(A, B, C, HR, ZDZ)
DO 20 I = 2, IMTM1
DO 20 J = 2, JMT
   SLARR(I-1,J-1) = A(I,J)
20 CONTINUE
WRITE(18, 910) TTSEC, DTTS
CALL ASCOUT(SLARR, IMT, IMTM2, JMTM1, VMASK, 2, 18)
GOTO 999
ENDIF

extract Temperature, Salinity, U Velocity, V Velocity
if MSI .ne. 0) Ice Fraction, Ice Thickness

IF (DEPVAR (1:3) .EQ. 'LAT') THEN

extract E/W section along latitude line INDEX

complete header for latitude slice

DO 250 IH = 1, IN

convert latitude to J value

LAT(IH) = TSLAB(IH)
VALS(IH) = 4 * (80.0 - LAT(IH)) - 3
250 CONTINUE
WRITE (18, 920) (LAT(IH), IH = 1, IN)

C IF (TRAC(1:3).EQ.'TEM'.OR.TRAC(1:3).EQ.'SAL') THEN
   DO 290 IH = 1, IN
      PRINT *, 'extracting slab ', IH
      INDEX = VALS (IH)
      C read in the data
      CALL MGSDAT(A, B, C, HR, ZDZ)
      C scale temperature and salinity fields and mask with FKMP array
      DO 220 I = 2, IMTM1
         DO 220 K = 1, KM
            IF (INT(FKMP(I,INDEX)) .EQ. 0) THEN
               SLARR (I-1,K) = VMASK(1)
            ELSE IF (INT(FKMP(I,INDEX)) .GE. K) THEN
               IF (TRAC(r.l) .EO. T) THEN
                  SLARR (I-1,K) = A(I,K)
               ELSE IF (TRAC(1:I) .EO- 'S') THEN
                  SLARR(I-1,K) = B(I) * 1000
                  SLARR(I-1,K) = SLARR(I-1,K) + 35.
               ENDIF
            ELSE
               SLARR (I-1, K) = VMASK(2)
            ENDIF
      220 CONTINUE
      WRITE (18, 910) TrSEC,DTTS
      C convert data to ASCOUT format and send to output stream
      CALL ASCOUT(SLARR, IMT, IMTM2, KM, VMASK, 2, 18)
   290 CONTINUE
C ELSE IF (TRAC(1:3).EO.'U V.OR.TRAC(1:3).EO.'VV)THEN
   DO 390 IH = 1, IN
      PRINT *, 'extracting slab ', IH
      INDEX = VALS (IH)
      C read in the data
      CALL MGSDAT(A, B, C, HR, ZDZ)
      C mask velocity fields with FKMQ array
      DO 320 I = 2, IMTM1
         DO 320 K = 1, KM
            IF (INT(FKMQ(I,INDEX)) .EQ. 0) THEN
               SLARR (I-1,K) = VMASK(1)
            ELSE IF (INT(FKMQ(I,INDEX)) .GE. K) THEN
               IF (TRAC(1:1).EQ. 'T') THEN
                  SLARR(I-1,K) = A(I,K)
               ELSE IF (TRAC(1:1) .EO. 'S') THEN
                  SLARR(I-1,K) = SLARR(I-1,K) + 35.
               ENDIF
            ELSE
               SLARR (I-1, K) = VMASK(2)
            ENDIF
      320 CONTINUE
IF (TRAC(1:1) .EQ. 'U') THEN
  SLARR(I-1, K) = A(I, K)
ELSE IF (TRAC(1:1) .EQ. 'V') THEN
  SLARR(I-1, K) = B(I, K)
ENDIF
ELSE
  SLARR(I-1, K) = VMASK(2)
ENDIF
320 CONTINUE
WRITE (18, 910) TTSEC, DTTS

C convert data to ASCOUT format and send to output stream
C
CALL ASCOUT(SLARR, IMT, IMTM2, KM, VMASK, 2, 18)
390 CONTINUE
ENDIF

ELSE IF (DEPVAR(1:3) .EQ. 'LON') THEN
  C extract N/S section along longitude line INDEX

  C complete header for longitude slice

  DO 400 IH = 1, IN
  C convert longitude to I value

      LONG(IH) = TSLAB(IH)
      VALS(IH) = 2.0 * LONG(IH) + 2
  400 CONTINUE
  WRITE (18, 920) (LONG(IH), IH = 1, IN)

  C IF (TRAC(1:3).EQ.'TEM'.OR.TRAC(1:3).EQ.'SAL') THEN

      DO 490 IH = 1, IN
      PRINT *, 'extracting slab ', IH
      INDEX = VALS(IH)

      C read in the data

      CALL MGSDAT(A, B, C, HR, ZDZ)

      C scale temperature and salinity fields and mask with FKMP array

      DO 420 J = 2, JMT
      DO 420 K = 1, KM
      IF (INT(FKMP(INDEX,J)) .EQ. 0) THEN
          TLONG(J,K) = VMASK(1)
      ELSE IF (INT(FKMP(INDEX,J)) .GE. K) THEN
          JK = (K-1)*JMT+J
          IF (TRAC(1:1) .EQ. 'T') THEN
              TLONG(J,K) = A(JK,1)
          ELSE IF (TRAC(1:1) .EQ. 'V') THEN
              TLONG(J,K) = B(JK,1)
          ENDIF
      ENDIF
ELSE IF (TRAC(1:1) .EQ. 'S') THEN
    TLONG(J,K) = B(K,1) * 1000
    TLONG(J,K) = TLONG(J,K) + 35.
ENDIF
ELSE
    TLONG(J,K) = VMASK(2)
ENDIF

DO 440 J = 2, JMT
    DO 440 K = 1, KM
        SLARR(J-1,K) = TLONG(J,K)
    440 CONTINUE
WRITE (18, 910) TTSEC, DTTS

C convert data to ASCOUT format and send to output stream
CALL ASCOUT(SLARR,IMT,JMTM1,KM,VMASK,2,18)

ELSE IF (TRAC(1:3).EQ.'U V'.OR.TRAC(1:3).EQ.'V V') THEN
    DO 590 IH = 1, IN
        PRINT *, 'extracting slab IH
INDEX = VALS(IH)

C read in the data
CALL MGSDAT(A, B, C, HR, ZDZ)

C mask velocity fields with FKMO array
DO 520 J = 2, JMTM1
    DO 520 K = 1, KM
        IF (INT(FKMO(INDEX,J)).EQ. 0) THEN
            TLONG(J,K) = VMASK(1)
        ELSE IF (INT(FKMO(INDEX,J)).GE. K) THEN
            JK = (K-1)*JMT+J
            IF (TRAC(1:1).EQ. 'U') THEN
                TLONG(J,K) = A(K,1)
            ELSE IF (TRAC(1:1).EQ. 'V') THEN
                TLONG(J,K) = B(K,1)
            ENDIF
        ELSE
            TLONG(J,K) = VMASK(2)
       ENDIF
    520 CONTINUE
DO 540 J = 2, JMTM1
    DO 540 K = 1, KM
        SLARR(J-1,K) = TLONG(J,K)
    540 CONTINUE
WRITE (18, 910) TTSEC, DTTS
C convert data to ASCOUT format and send to output stream
C
CALL ASCOUT(SLARR,IMT,JMTM2,KM,VMASK,2,18)
S90 CONTINUE
ENDIF
C
ELSE IF (DEFVAR(1:3) .EQ. 'DEP') THEN
C
extract horizontal section at level INDEX
C
complete header for horizontal slice
C
IF (TRAC(I:3) .EQ. 'ICE') THEN
WRITE (18, 900)  '  '
ELSE
DO 600 IH= I, IN
C
C convert level depth to K value
C
VALS(IH) = INT(TSLAB(IH))
600 CONTINUE
WRITE(18, 930) (VALS(IH), IH = I, IN)
ENDIF
C
C
IF (TRAC(I:3) .EQ. 'TEM' .OR. TRAC(1:3) .EQ. 'SAL') THEN
C
DO 690 IH = 1, IN
PRINT *, 'extracting slab ', IH
INDEX = VALS(IH)
C
read in the data
C
CALL MGSDAT(A, B, C, HR, ZDZ)
C
scale temperature and salinity fields and mask with FKMP array
C
DO 620 J = 2, JMT
DO 620 I = 1, IMT
IF (INT(FKMP(I,J)) .EQ. 0) THEN
TDEP(I,J) = VMASK (1)
ELSE IF (INT(FKMP(I,J)) .GE. INDEX) THEN
IF (TRAC(I:1) .EQ. 'T') THEN
TDEP(I,J) = A(I,J)
ELSE IF (TRAC(1:1) .EQ. 'S') THEN
TDEP(I,J) = B(I,J) * 1000
TDEP(I,J) = TDEP(I,J) + 35
ENDIF
ELSE
TDEP(I,J) = VMASK (2)
ENDIF
620 CONTINUE
DO 640 J= 2, JMT
DO 640 I= 2, IMTM1
   SLARR(I-1, J-1) = TDEP(I,J)
   CONTINUE
 WRITE (18,910) TTSEC, DTTS

C convert data to ASCOUT format and send to output stream
C
 CALL ASCOUT(SLARR, IMT, IMTM2, JMTM1, VMASK, 2, 18)
 CONTINUE
C
ELSE IF (TRAC(1:3) .EQ. 'U V' .OR. TRAC(1:3) .EQ. 'V V') THEN

DO 690 IH = 1, IN
   PRINT *, 'extracting slab ', IH
   INDEX = VALS(IH)

C read in the data
C
 CALL MGSDAT(A, B, C, HR, ZDZ)

C mask velocity fields with FKMO array
C
   DO 720 J = 2, JMTM1
   DO 720 I = 1, IMT
      IF (INT(FKMO(IJ)) .EQ. 0) THEN
         TDEP (I,J) = VMASK(1)
      ELSE IF (INT(FKMO(IJ)) .GE. INDEX) THEN
         IF (TRAC(1:1) .EQ. 'U') THEN
            TDEP(I,J) = A(I,J)
         ELSE IF (TRAC(1:1) .EQ. 'V') THEN
            TDEP(I,J) = B(I,J)
         ENDIF
      ELSE
         TDEP (I,J) = VMASK(2)
      ENDIF
   ENDIF

   720 CONTINUE
   DO 740 J=2,JMTM1
   DO 740 I=2,IMTM1
      SLARR(I-1, J-1) = TDEP(I,J)
   ENDIF

   740 CONTINUE
 WRITE (18,910) TTSEC, DTTS

C convert data to ASCOUT format and send to output stream
C
 CALL ASCOUT(SLARR, IMT, IMTM2, JMTM2, VMASK, 2, 18)
 CONTINUE
C
ELSE IF (TRAC(1:3) .EQ. 'ICE' .AND. MSI .NE. 0) THEN

C extract horizontal section at the surface - ice fraction, ice thickness
C
 read in the data
CALL MGSDAT(A, B, C, HR, ZDZ)

mask ice fields with FKMQ array

DO 820 J = 2, JMT
DO 820 I = 1, IMT
IF (INT(FKMQ(I,J)) .EQ. 0) THEN
    TDEP(I,J) = VMASK(1)
ELSE IF (INT(FKMQ(I,J)) .GE. INDEX) THEN
    IF (TRAC(1:5) .EQ. 'ICE F') THEN
        TDEP(I,J) = A(I,J)
    ELSE IF (TRAC(1:5) .EQ. 'ICE T') THEN
        TDEP(I,J) = B(I,J)
    ENDIF
ELSE
    TDEP(I,J) = VMASK(2)
ENDIF
820 CONTINUE

DO 840 J = 2, JMTM1
DO 840 I = 2, IMTM1
SLARR(I-1, J-1) = TDEP(I,J)
840 CONTINUE

WRITE (18,910) ITSEC, DTTS

convert data to ASCOUT format and send to output stream

CALL ASCOUT(SLARR,IMT,IMTM2,JMTM2,VMASK,2,18)

FORMAT (A50)
FORMAT ('FIRST TTSEC ',F12.0,' DTTS ','F5.0)
FORMAT (8F8.3)
FORMAT (16I5)
STOP

SETUP ROUTINES USED BY PROGRAM EXTRACT

The variables in COMMON block LVALS are set in subroutine SETUP and used in the main program.

Variable. Definition
TSLAB(8) Position of slabs to be extracted (latitude, longitude, level)
IN Number of levels to be extracted (maximum of 8).
2.1 SUBROUTINE SETDIR (DEFDIR, DIRN)
C This routine sets the directory to search for the FRAM archive datasets.
C DEFDIR - defined default directory containing the archive datasets
C DIRN - directory to be used by the program.
C
CHARACTER*512 DEFDIR
CHARACTER*512 DIRN
CHARACTER*512 TDIR
LOGICAL AROUND
C
PRINT *, 'Directory $<CR> to select default directory ',
& DEFDIR(:LNBLNK(DEFDIR)), ' ' &
READ (*, '(A512)') TDIR
WRITE(*, '(A1)') ' '
C
IF (LEN(TDIR(:LNBLNK(TDIR))) .EQ. 0) THEN
   DIRN = DEFDIR
ELSE
   INQUIRE(FILE = TDIR, EXIST = AROUND)
   IF (AROUND) THEN
      DIRN = TDIR
   ELSE
      PRINT *, 'Cannot find directory ', TDIR(:LNBLNK(TDIR))
      STOP
   ENDIF
ENDIF
END

2.2 SUBROUTINE SETUP(TRAC, DEPVAR, NAMRUN)
C This is a control routine to set type and contents of slab(s) to be extracted from the
C dataset.
C TRAC - field variable
C DEPVAR - orientation variable (longitude, latitude, depth)
C NAMRUN - comment line in header of output file.
C
CHARACTER NAMRUN*50
CHARACTER*9 DEPVAR
CHARACTER*1S TRAC
INTEGER ITRAC, IDEP
COMMON /PLTTYP/IDIR,ITYP,INDEX,ITT,IDIM
COMMON /LVALS/ TSLAB(8), IN
C
NAMRUN = ' FINE RESOLUTION MODEL
C
ITRAC = -1
DO 100 WHILE (ITRAC .LT. 0 .OR. ITRAC .GT. 10)
   WRITE(*, 910)
WRITE(*, 910) ' TRAC  
WRITE(*, 910) ' 1. Stream Function  
WRITE(*, 910) ' 2. USTAR   
WRITE(*, 910) ' 3. VSTAR    
WRITE(*, 910) ' 4. Pressure  
WRITE(*, 910) ' 5. Temperature 
WRITE(*, 910) ' 6. Salinity  
WRITE(*, 910) ' 7. U Velocity  
WRITE(*, 910) ' 8. V Velocity 
WRITE(*, 910) ' 9. Ice Fraction  
WRITE(*, 910) ' 10. Ice Thickness' 
WRITE(*, '(A38, $') ' Enter number of field required:' 
READ(*, '(12)') ITRAC 
100 CONTINUE 
C IF (ITRAC .EQ. 0) THEN 
   STOP 
ELSE IF (ITRAC .EQ. 1) THEN 
   TRAC = 'STREAM FUNCTION' 
   DEPVAR = 'STREAM' 
   IDIR = 0 
   ITYP = 0 
   TSLAB(1) = 0.0 
   IN = 1 
ELSE IF (ITRAC .EQ. 2) THEN 
   TRAC = 'USTAR' 
   DEPVAR = 'STREAM' 
   IDIR = 0 
   ITYP = 3 
   TSLAB(1) = 0.0 
   IN = 1 
ELSE IF (ITRAC .EQ. 3) THEN 
   TRAC = 'VSTAR' 
   DEPVAR = 'STREAM' 
   IDIR = 0 
   ITYP = 3 
   TSLAB(1) = 0.0 
   IN = 1 
ELSE IF (ITRAC .EQ. 4) THEN 
   TRAC = 'PRESSURE' 
   DEPVAR = 'STREAM' 
   IDIR = 0 
   ITYP = 4 
   TSLAB(1) = 0.0 
   IN = 1 
ELSE IF (ITRAC .EQ. 5) THEN 
   TRAC = 'TEMPERATURE' 
   ITYP = 2 
ELSE IF (ITRAC .EQ. 6) THEN 
   TRAC = 'SALINITY'
ITYP = 2
ELSE IF (ITRAC .EQ. 7) THEN
  TRAC = 'U VELOCITY'
  ITYP = 1
ELSE IF (ITRAC .EQ. 8) THEN
  TRAC = 'V VELOCITY'
  ITYP = 1
ELSE IF (ITRAC .EQ. 9) THEN
  TRAC = 'ICE FRACTION'
  DEPVAR = 'DEPTH'
  IDIR = 0
  ITYP = 5
  TSLAB(1) = 0.0
  IN = 1
ELSE IF (ITRAC .EQ. 10) THEN
  TRAC = 'ICE THICKNESS'
  DEPVAR = 'DEPTH'
  IDIR = 0
  ITYP = 5
  TSLAB(1) = 0.0
  IN = 1
ENDIF

IF (TTRAC .GT. 4 .AND. ITRAC .LE. 8) THEN
  IDEP = -1
  DO 200 WHILE (IDEP .LT. 0 .OR. IDEP .GT. 3)
    WRITE(*, 910)
    WRITE(*, 910) '  DEPVAR  '
    WRITE(*, 950) & '  1. Vertical Section (E/W) - constant latitude  '
    WRITE(*, 950) & '  2. Vertical Section (N/S) - constant longitude'
    WRITE(*, 950) & '  3. Horizontal Section - constant depth  '
    WRITE(*, 910)
    WRITE(*, '(A38, $)') & '  Enter number of field required :'
    READ(*, '(II)') IDEP
  200 CONTINUE
ENDIF
C
300 WRITE(*, 910)
WRITE(*, '(A27, $)') ' No of slabs (MAX=8) : '
READ(*, '(I1)') IN
IF (IN.LT. 0 .OR. IN .GT. 8) GOTO 300
WRITE (*, 910)
DO 400 I = 1, IN
IF (IDEP .EQ. 1) THEN
WRITE(*, '(A10, I1, A13, $)') Slab I, Latitude = '
READ(*, '(F5.2)') TSLAB(I)
IF (TSLAB(I) .LT. E-4) TSLAB(I) = -TSLAB(I)
ELSE IF (IDEP .EQ. 2) THEN
WRITE(*, '(A10, I1, A14, $)') Slab I, Longitude = '
READ(*, '(F5.2)') TSLAB(I)
IF (TSLAB(I) .LT. E-4) TSLAB(I) = 360.0 + TSLAB(I)
ELSE IF (IDEP .EQ. 3) THEN
WRITE(*, '(A10, I1, A10, $)') Slab I, Level = '
READ(*, '(I2)') rSLAB
TSLAB(I) = FLOAT(rSLAB)
ENDIF
400 CONTINUE
ENDIF
910 FORMAT(A20)
930 FORMAT(A16)
940 FORMAT(A24)
950 FORMAT(A47)
999 RETURN
END
APPENDIX II FORTRAN LISTINGS OF FRAM USER INTERFACE SUBROUTINES

1 High level routine

The variables in COMMON blocks GRIPAR and PLITYP must be set before this routine is called by a user program.

SUBROUTINE MGSDAT(A, B, C, HRR, ZDZ)

C Reads in slabs of data according to the variables IDIR, ITYP and INDEX.

C If IDIR = 0 - reads in a horizontal section of data
C from depth level INDEX
C = 1 - reads in an East-West section of data,
C along the INDEX-th line of grid points
C = 2 - reads in a North-South section of data,
C along the INDEX-th line of grid points

C If ITYP = 0 - reads the stream function into C; A and B are unused,
C applicable only when IDIR = 0
C = 1 - reads U velocity into A, V velocity into B
C and stream function into C
C The inverse depth array is passed in HR
C The barotropic and baroclinic velocities are combined
C to give the full velocity
C = 2 - reads T into A, S into B; C not used
C = 3 - reads USTAR into A, VSTAR into B,
C and stream function into C
C IDIR not applicable.
C = 4 - reads the pressure field PRESS into A; B, C not used
C = 5 - reads the ice fraction into A, ice thickness into B;
C C not used

COMMON /PLITYP/IDIR,ITYP,INDEX,ITT,IDIM
COMMON /GRIPAR/PSIDEG,DXDEG,PHIDEG,DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
DIMENSION A(1),B(1),C(1), HRR(IMT,JMT),ZDZ(KM)
DATA RADIAN,RADIUS/57.29578,6370.E5/

C

RAD = RADIAN / RADIUS
IDIM = IMT
JMTM1 = JMT - 1
JMTM2 = JMT - 2
IF (ITYP .EQ. 0 .OR. ITYP .EQ. 1) THEN
    read stream function
    CALL MGSRDS(C)
    IF (ITYP .EQ. 0) THEN
        RETURN
    ELSE IF (ITYP .EQ. 1) THEN
        read the inverse depth array
        CALL MGRDH(HR)
        ENDIF
    ELSE IF (ITYP .EQ. 3) THEN
        read (USTAR, VSTAR) into (A, B)
        CALL MGRDU(A, B)
        RETURN
    ELSE IF (ITYP .EQ. 4) THEN
        read the pressure field
        CALL MGRDP(A)
        ENDIF
    IF (k/ISI .EQ. 0) THEN
        LSI = 0
    ELSE
        LSI = MSI + 6 + 1
    ENDIF
    CALL MGSADV(7)
    IF (IDIR .EQ. 0) THEN
        ' Horizontal Section '
        IF (ITYP .EQ. 5) THEN
            IF (MSI .NE. 0) THEN
                DO 100 J = 1, JMT
                    CALL MGSSK1(NT + 2)
                    CALL MGSSK2(1)
                    CALL MGRD0(A, J)
                    CALL MGSSK2(1)
                    CALL MGRD0(B, J)
                    CALL MGSSK2(LSI - 4)
                    CALL MGSSK2(LBC)
                    IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
                        CALL MGSSK1(NT + 2)
                    ENDIF
                100 CONTINUE
            ENDIF
        ENDIF
    ENDIF
ENDIF
CALL MGSSK2(LBC)
100 CONTINUE
ENDIF
RETURN
ELSE IF (ITYP .EQ. 2) THEN
C
C read (T, S) at depth level INDEX into (A, B)
C
DO 200 J=1, JMT
   CALL MGSRD1(A,J,INDEX)
   CALL MGSRD1(B,J,INDEX)
   CALL MGSSKI(NT)
   CALL MGSSK2(LBC + LSI)
   IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGSSK2(LBC)
200 CONTINUE
RETURN
ELSE IF (ITYP .EQ. 1) THEN
C
C read (U, V) at depth level INDEX into (A, B)
C
DO 300 J=1, JMT
   CALL MGSSK1(NT)
   CALL MGSRD1(A,J,INDEX)
   CALL MGSRD1(B,J,INDEX)
   CALL MGSSK2(LBC + LSI)
   IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGSSK2(LBC)
C
Add in barotropic velocities

IF (J .EQ. JMT) GOTO 300
CSR1 = 1.0/COS((PHIDEG+(J-1)*DYDEG)/RADIAN)
IMU = IMT-1
DO 350 I = 1, IMU
   HRR= HR(1,J)
   IF (HRR .NE. 0.0) THEN
      IF (1.0 + 1.0 / HRR .GE. ZDZ(INDEX)) THEN
         J = (J-1) * IMT + 1
         DIAG1 = C(JJ+IMT+1)-C(JJ)
         DIAG2 = C(JJ+IMT )-C(JJ+1)
         W1 = -(DIAG1+DIAG2)*HRR*0.5*RAD/DYDEG
         W2 = (DIAG1-DIAG2)*HRR*CSR1*0.5*RAD/DXDEG
         J = (J-1)*IMT+1
         A(JJ) = A(JJ)+W1
         B(JJ) = B(JJ)+W2
      ENDIF
   ENDIF
350 CONTINUE

ENDIF
 CALL MGSSK2(LBC)
ENDIF
ENDIF
350 CONTINUE
300 CONTINUE
RETURN
ENDIF
ELSE IF (IDIR .EQ. 1) THEN

C
C East-West Section
C
JJ = INDEX - 1
IF (JJ .NE. 0) THEN
   DO 400 J = 1, JJ
      CALL MGSSK1(NT + 2)
      CALL MGSSK2(LBC + LSI)
      IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
         CALL MGSSK1(NT + 2)
      ENDIF
      CALL MGSSK2(LBC)
   400 CONTINUE
ENDIF

C Have now reached relevant data
C
IF (ITYP .EQ. 2) THEN
C
read (T, S) into (A, B)
C
   CALL MGSRD2(A)
   CALL MGSRD2(B)
   CALL MGSSK1(NT)
   CALL MGSSK2(LBC + LSI)
   IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGSSK2(LBC)
ELSE IF (ITYP .EQ. 1) THEN
C
read (U, V) into (A, B)
C
   CALL MGSSK1(NT)
   CALL MGSRD2(A)
   CALL MGSRD2(B)
   CALL MGSSK2(LBC + LSI)
   IF (J .EQ. JMTM2 .OR. J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGSSK2(LBC)
C
Add in barotropic velocities
C
   J = INDEX
IF (J .EQ. JMT) GOTO 590
CSRJ = 1.0/COS((PHIDEG+(J-1)*DYDEG)/RADIAN)
IMU = JMT-1
DO 500 J = 1, IMU
   HRR = HR(J,J)
   IF (HRR .EQ. 0) GOTO 500
   JJ = (J-1)*IMT+1
   DIAG1 = C(J+IMT+1)-C(JJ)
   DIAG2 = C(J+IMT)-C(JJ+1)
   W1 = -(DIAG1+DIAG2)*HRR*0.5*RAD/DYDEG
   W2 = (DIAG1-DIAG2)*HRR*CSRJ*0.5*RAD/DXDEG
   DO 550 K = 1, KM
      IF (1.0 + 1.0/HRR.GE. ZDZ(K))THEN
         IK = (K-1)*IMT+1
         A(IK) = A(IK)+W1
         B(IK) = B(IK)+W2
      ENDIF
   550 CONTINUE
   500 CONTINUE
ENDIF

590 J = INDEX + 1
IF (J .GT. JMT) RETURN
DO 600 J = J, JMT
   CALL MGSSK1(NT + 2)
   CALL MGSSK2(LBC + LSI)
   IF (J .EQ. JMT2 .OR. J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGSSK2(LBC)
   600 CONTINUE
RETURN

ELSE IF (IDIR .EQ. 2) THEN

   ' North-South Section '

   DIM=JMT
   IF (TTYP .EQ. 2) THEN
      read (T, S) into (A, B)
      DO 700 J = 1, JMT
         CALL MGSRD3(A.J,INDEX)
         CALL MGSRD3(B,J,INDEX)
         CALL MGSSK1(NT)
         CALL MGSSK2(LBC + LSI)
         IF (J .EQ. JMT2 .OR. J .EQ. JMTM1) THEN
            CALL MGSSK1(NT + 2)
         ENDIF
         CALL MGSSK2(LBC)
      700 CONTINUE
   RETURN
ELSE IF (ITYP .EQ. 1) THEN

read (U, V) into (A, B)

DO 800 J = 1, JMT
CALL MGSSK1(NT)
CALL MGSRD3(A, J, INDEX)
CALL MGSRD3(B, J, INDEX)
CALL MGSSK2(LBC + LSI)
IF (J .EQ. JMT2 .OR. J .EQ. JMT1) THEN
   CALL MGSSK1(NT + 2)
ENDIF
CALL MGSSK2(LBC)

Add in barotropic velocities

I = INDEX
IF (J .EQ. JMT) GOTO 800
CSRJ = 1.0 / COS((PHIDEG + J-1)*DYDEG/RADIAN)
HRR = HR(I, J)
IF (HRR .EQ. 0.0) GOTO 800
H = 0.001 + 1.0 / HRR
IJJ = (J-1) * IMT + I
DIAG1 = C(JJ + IMT + 1) - C(JJ )
DIAG2 = C(JJ + IMT ) - C(JJ + 1)
W1 = -(DIAG1+DIAG2) * HRR * 0.5 * RAD / DYDEG
W2 = (DIAG1-DIAG2) * HRR * CSRJ * 0.5 * RAD / DXDEG
DO 850 K = 1, KM
IF (H .GE. ZDZ(K)) THEN
   JK = (K-1) * JMT + J
   A(JK) = A(JK) + W1
   B(JK) = B(JK) + W2
ENDIF
850 CONTINUE
800 CONTINUE
RETURN
ENDIF
ENDIF
END
Medium level routines
These routines need the integer constants in COMMON block GRIPAR to be set before they are called by a user program.

SUBROUTINE READ4()
C Read in RIT, TTSEC, AREA, VOLUME from the input file
C COMMON /PLTYP, IDIR, ITYP, INDEX, ITT, IDIM
COMMON /TIME/ TTSEC
COMMON /MGSCCR/ MWORD
REAL ITEM(4)
C CALL MGSADV(2)
CALL XREAD(ITEM, MWORD, 4, IFAIL)
RIT = ITEM(1)
ITT = INT(RIT)
TTSEC = ITEM(2)
RETURN
END

SUBROUTINE MGSADV(ISECT)
C Moves pointer 'MWORD' to start of section 'ISECT'
C
ISECT = 1 Start of header
= 2 Start of RITT
= 3 Start of stream function
= 4 Start of HR
= 5 Start of USTAR and VSTAR arrays
= 6 Start of pressure array
= 7 Start of slabs.
C
Needs the variables in COMMON /GRIPAR/ to be set correctly.
C
DIMENSION L(7)
COMMON /GRIPAR/ X1DEG, DXDEG, Y1DEG, DYDEG, & IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD
C
NWDS = IMT * JMT
NDICES = 2 * LSEG * JMT + 4 * NISLE
L(1) = 1
L(2) = L(1) + 4000
L(3) = L(2) + 4 + IMT*KM*NT + NWDS
L(4) = L(3) + NWDS
L(5) = L(4) + NWDS * 3
L(6) = L(5) + NWDS * 2
L(7) = L(6) + NWDS + NDICES
JSECT = MAX(1, MIN(7, ISECT))
MWORD = L(JSECT)
RETURN
END

SUBROUTINE MGSRDS(A)
C
C Moves to and reads in the stream function.
C
DIMENSION A(1)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD
C
CALL MGSADV(3)
CALL XREAD(A, MWORD, IMT*JMT, IFAIL)
IF (IFAIL .NE. 0) STOP
RETURN
END

SUBROUTINE MGSRDH(HR)
C
C Moves to and reads in the inverse depth array.
C
DIMENSION HR(1)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD
C
CALL MGSADV(4)
CALL XREAD(HR, MWORD, IMT*JMT, IFAIL)
IF (IFAIL .NE. 0) STOP
RETURN
END

SUBROUTINE MGSRDP(A)
C
C Moves to and reads in the pressure array.
C
DIMENSION A(1)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI COMMON/MGSCCR/MWORD

CALL MGSADV(6)
CALL XREAD(A, MWORD, IMT*JMT, IFAIL)
IF(FAIL.NE.0) STOP
RETURN
END

SUBROUTINE MGSRDU(A,B)

C Moves to and reads in the USTAR and VSTAR arrays.
DIMENSION A(1),B(I)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/MWORD
CALL MGSADV(5)
CALL XREAD(A,MWORD,IMT*JMT,FAIL)
IF(FAIL.NE.0) STOP
CALL XREAD(B,MWORD,IMT*JMT,FAIL)
IF(FAIL.NE.0) STOP
RETURN
END

SUBROUTINE MGFKM1(FKMP)

C Reads the FKMP array (masking data for the T,S points).
COMMON /GRIPAR/PSIDEG, DXDEG, PHIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
REAL FKMP(IMT, JMT)

JMTM1 = JMT - 1
JMTM2 = JMT - 2
IF (MSI .EQ. 0) THEN
  LSI = 0
ELSE
  LSI = MSI + 6 + 1
ENDIF

CALL MGSADV(7)
DO 100 J = 1, JMT
  CALL MGSSK1(NT + 2)
  CALL MGSSK2(LSI)
  CALL MGSRDF(FKMP, J)
  CALL MGSSK2(LBC - 1)
100 CONTINUE
IF (J .EQ. JMTM2 OR J .EQ. JMTM1) THEN
   CALL MGSSK1(NT + 2)
ENDIF
CALL MGSSK2(LBC)
100 CONTINUE

C
C Set row JMT equal to row JMTM1 for northern boundary
C
DO 200 I = 1, JMTM1
   FKMP(I, JMT) = FKMP(I, JMTM1)
200 CONTINUE

C
C Set cyclic boundary conditions
C
DO 300 J = 1, JMT
   FKMP(1, J) = FKMP(IMT, J)
   FKMP(IMT, J) = FKMP(2, J)
300 CONTINUE

RETURN
END

SUBROUTINE MGFKM2(FKMQ)

C
C Reads the FKMQ array (masking data for the U,V points).
C
COMMON /GRIPAR/PSIDEG, DXDEG, PHIDEG, DYDEG,
   & IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
REAL FKMQ(IMT, JMT)
C
JMTM1 = JMT - 1
JMTM2 = JMT - 2
IF (MSI .EQ. 0) THEN
   LSI = 0
ELSE
   LSI = MSI + 6 + 1
ENDIF
C
CALL MGSADV(7)
DO 100 J = 1, JMT
   CALL MGSSK1(NT + 2)
   CALL MGSSK2(LSI)
   CALL MGSSK2(LBC)
   IF (J .EQ. JMTM2 OR J .EQ. JMTM1) THEN
      CALL MGSSK1(NT + 2)
   ENDIF
   CALL MGRDF(FKMQ, J)
   CALL MGSSK2(LBC - 1)
100 CONTINUE

C
C Set row JMT equal to row JMTM1 for northern boundary
C
DO 200 I = 1, IMT + 1
   FKMO(I, JMT) = FKMO(I, JMT)
200 CONTINUE

C Set cyclic boundary conditions
C
DO 300 J = 1, JMT
   FKMO(IMT, J) = FKMO(2, J)
300 CONTINUE
RETURN
END

SUBROUTINE MGSRD0(A, J)

C Reads in A(I,J), 1<I<IMT from surface level
C On entry :
C   MWORD should point to the first word of a 1-dimensional
C   (IMT) sub-array in the Jth slab
C   For example the start of the ice fraction field
C   in the Jth slab.
C On exit :
C   MWORD points to the first word of the next sub-array.
C
DIMENSION A(1)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
   & IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD

M = MWORD
L = (J - 1) * IMT + 1
CALL XREAD(A(L), M, IMT, IFAIL)
IF (IFAIL .NE. 0) STOP
MWORD = MWORD + IMT
RETURN
END

SUBROUTINE MGSRD1(A, J, INDEX)

C Reads in A(I,J), 1<I<IMT from depth level INDEX
C On entry :
C   MWORD should point to the first word of a 2-dimensional
C   (IMT, JMT) sub-array in the Jth slab
C   For example the start of the T field in the Jth slab.
C On exit :
C   MWORD points to the first word of the next sub-array.
C
DIMENSION A(1)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG, IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD

M = MWORD + IMT * (INDEX - 1)
L = (J - 1) * IMT + 1
CALL XREAD(A(L), M, IMT, IFAIL)
IF (IFAIL .NE. 0) STOP
MWORD = MWORD + IMT * KM
RETURN
END

SUBROUTINE MGSRD2(A)

C Reads in A(I,K), 1<I<IMT, 1<K<KM
C On entry:
C MWORD should point to the first word of a 2-dimensional
C (IMT,KM) sub-array in the Jth slab
C For example the start of the T field in the Jth slab
C On exit:
C MWORD points to the first word of the next sub-array.
C
DIMENSION A(I)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG, IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD

CALL XREAD(A(I), MWORD, IMT*KM, IFAIL)
IF (IFAIL .NE. 0) STOP
RETURN
END

SUBROUTINE MGSRD3(A, J, INDEX)

C Reads in A(J,K), 1<J<JMT, 1<K<KM, I=INDEX
C On entry:
C MWORD should point to the first word of a 2-dimensional
C (JMT,KM) sub-array in the Jth slab
C On exit:
C MWORD points to the first word of the next sub-array.
C
DIMENSION A(I)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG, IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD

M1 = MWORD + INDEX - 1
DO 100 K = 1, KM
M = M1 + (K-1) * IMT
L = J + (K-1) * JMT
CALL XREAD(A(L), M, 1, IFAIL)
IF (IFAIL .NE. 0) STOP
100 CONTINUE
MWORD = MWORD + IMT * KM
RETURN
END

SUBROUTINE MGSRDF(A, J)
C
C Reads in a 1-dimensional slab array of length IMT into row J
C of the array A(IMT, JMT)
C On entry :-
C MWORD should point to the first word of the 1-dimensional slab array
C On exit :-
C MWORD points to the first word of the next slab array.
C
DIMENSION A(l)
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD
C
L = (J-1) * IMT + 1
CALL XREAD(A(L),MWORD,IMT,IFAIL)
IF (IFAIL .NE. 0) THEN
  PRINT *, 'j = ', J
  STOP
ENDIF
RETURN
END

SUBROUTINE MGSSK1(K)
C
C Skips over K 2-dimensional (IMT, KM) sub-arrays.
C i.e. K * (IMT * KM) data values
C
COMMON /GRIPAR/ XIDEG, DXDEG, YIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD
C
MWORD = MWORD + K * IMT * KM
RETURN
END
SUBROUTINE MGSSK2(K)

C Skips over K 1-dimensional (IMT) sub-arrays.
C ie. K * (IMT) data values

COMMON /GRIPAR/ XIDEG, DXDEG, Y1DEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
COMMON /MGSCCR/ MWORD

MWORD = MWORD + K * IMT
RETURN
END

SUBROUTINE MGSDZZ(DZ,DZZ,ZDZ,ZDZZ)

C Subroutine to calculate depth parameters used.

C DZ - array of slab thicknesses
C DZZ - array; on exit contains the distances between
C vertical T,S points
C ZDZ - array; on exit contains the depth of each level bottom
C ZDZZ - array; on exit contains the depth of T,S grid points

COMMON /GRIPAR/ PSIDEG, DXDEG, PHIDEG, DYDEG,
& IMT, JMT, KM, NT, LSEG, NISLE, LCYC, LBC, MSI
DIMENSION DZ(1), DZZ(1), ZDZ(1), ZDZZ(1)

DZZ(1)=0.5*DZ(1)
ZDZ(1)=DZ(1)
ZDZZ(1)=DZZ(1)
DO 10 I=2,KM
   DZZ(I)=0.5*(DZ(I-1)+DZ(I))
   ZDZ(I)=ZDZ(I-1)+DZ(I)
   ZDZZ(I)=ZDZZ(I-1)+DZZ(I)
10 CONTINUE
DZZ(KM+1)=0.5*DZ(KM)
ZDZZ(KM+1)=ZDZZ(KM)+DZZ(KM+1)
RETURN
END
3 Low level routine

This is the basic low level routine called by all the above high and medium level routines.

SUBROUTINE XREAD(A, M, N, IFAIL)

C Subroutine for reading data from FRAM archive files.
C A(N) - array into which the data is placed
C M  - position in the file of the first variable to be read
C N  - number of variables to be read
C If no fault occurs the subroutine returns with
C IFAIL set to 0
C M  - set to its original value plus N
C The 32-bit input data is stored temporarily in buffer 'EEBUF'
C LBUFF is the position in the input file of the first element of the
C current buffer
C MBUFF is the length of the buffer (4000)
C IUNIT is the input stream
C COMMON /EEBUF/EEBUF(4000), LBUFF, MBUFF, IUNIT, OUNIT
INTEGER  M, N, A(N), IBUFF(4000), LBUFF, IUNIT, OUNIT
DIMENSION EEBUF(6)
EQUIVALENCE (EEBUF(1), IBUFF(1))
C
LBUFF1 = LBUFF
IREC = 1 + (M - 1) / MBUFF
LBUFF = 1 + MBUFF * (IREC - 1)
IF (LBUFF1 NE. LBUFF) THEN
  READ(UNIT = IUNIT, REC = IREC, END = 995, ERR = 999) EEBUF
ENDIF
C
DO 100 I = 1, N
  J = M + I - LBUFF
  IF (J .GT. MBUFF) THEN
    IREC = IREC + 1
    READ(UNIT = IUNIT, REC = IREC, END = 996, ERR = 999) EEBUF
    LBUFF = LBUFF + MBUFF
    J = M + I - LBUFF
  ENDIF
  A(I) = EEBUF(J)
100 CONTINUE
IFAIL=0
M = M + N
RETURN
C
995 PRINT *, ' *** SUBROUTINE XREAD - EOF. 1'
Input and output routines

These routines open the input and output files, name the output file and write the header and data to it.

SUBROUTINE INSTR(NUNIT, DIRN, INFIL, IFAIL)

C Initialise the input stream.
C
C IUNIT = NUNIT = input stream
C DIRN = directory containing the input data files
C INFIL = name of input file
C
COMMON /EBUFF/EBUF(4000), LBUFF, MBUFF, IUNIT, OUNIT
CHARACTER*1024 INFIL
CHARACTER*612 DIRN
CHARACTER*612 FILEN
LOGICAL AROUND
C
MBUFF = 4000
LBUFF = -3999
IUNIT = NUNIT

C WRITE(*, '((A32, T1))') 'Enter name of input data file :'
READ(*, '(A512)') FILEN
WRITE(*, '(A1)') '*'

C INFIL = DIRN(:LNBLNK(DIRN)) // '/' // FILEN(:LNBLNK(FILEN))
INQUIRE(FILE = INFIL(:LNBLNK(INFIL)), EXIST = AROUND)
IF (AROUND) THEN
  PRINT *, 'input file = ', INFIL(:LNBLNK(INFIL))
ELSE
  PRINT *, 'Cannot find file ', INFIL(:LNBLNK(INFIL))
END IF

C OPEN(UNIT = IUNIT, FORM = 'UNFORMATTED',
& FILE = INFIL(:LNBLNK(INFIL)), ACCESS = 'DIRECT',
& RECL = 16000, STATUS = 'OLD',
& IOSTAT = ISTAT, ERR = 999)

C REWIND IUNIT
IREC = 0
DO 100 I = 1, MBUFF
  IEBUF(I) = 0
100 CONTINUE
IFAIL = 0
RETURN

C 999 PRINT *,
& '--- ERROR in opening input file ',
& INFIL(:LNBLNK(INFIL))
IFAIL = 1
STOP
END

C SUBROUTINE READHD()

C Read in header from the input file.

C COMMON /MGSCCR/ MWORD
CHARACTER CTEM(16000)

C CALL MGSADV(1)
CALL XRREAD(CTEM, MWORD, 16000, IFAIL)
WRITE(*, '(A1)') CTEM
RETURN
END
SUBROUTINE OUTFSTR(NUNIT, OUTFIL, IFAIL)

Initialise the output stream.

UNIT  = NUNIT = output stream
OUTFIL  = name of output file

COMMON /IEBUF/IEBUFF(4000), LBUFF, MBUFF, IUNIT, OUNIT
CHARACTER *(* ) OUTFIL
INTEGER  IEBUFF(4000), LBUFF, MBUFF, IUNIT, OUNIT

OUNIT  = NUNIT
OPEN(UNIT  = OUNIT, FILE  = OUTFIL,
&  STATUS  = 'NEW', ERR  = 999)
REWIND OUNIT
IFAIL  = 0
RETURN

999 PRINT *, '----- ERROR in opening output file ', OUTFIL
PRINT *, '----- file may already exist '
IFAIL  = 1
STOP
END

SUBROUTINE OFILEN(TRAC, DEPVAR, OUTFIL, IFAIL)

Creates an output filename for the cards file.

CHARACTER*15 TRAC
CHARACTER*9 DEPVAR
CHARACTER*12 OUTFIL
COMMON /TIME/ TTSEC
CHARACTER LETT*4, A*1, D*6, T1, T2, T3, T4
REAL SECDAY, TTSEC, FACTOR, NDAY
INTEGER IFAC T0, ADAY, TDAY, EXNUM, J, I, INDAY
CHARACTER*1 CVAR, FNAME(4)
INTEGER PVAR1, PVAR2, PA

SECDAY  = 86400.
NDAY  = TTSEC / SECDAY
INDAY  = INT(NDAY)
TDAY  = INDAY

DATA A / "A" /
DATA D / "cards" /

EXNUM  = 1
J  = 1
DO 100 I = 3, 0, -1
FACTOR  = 10 ** I

100 CONTINUE

STOP
END
IFACTOR = INT(FACTOR)
ADAY = TDAY / IFACTOR
IF (ADAY .NE. 0) EXNUM = 0
IF (EXNUM .EQ. 0) THEN
   FNAME() = CHAR(48 + ADAY)
   J = J + 1
   TDAY = TDAY - (ADAY * IFACTOR)
ENDIF
100 CONTINUE

C
T1 = FNAME(1)
T2 = FNAME(2)
T3 = FNAME(3)
T4 = FNAME(4)
LETT = T1 // T2 // T3 // T4

C
IF (TRAC(1:3) .EQ. 'UST') THEN
   CVAR = 'm'
ELSE IF (TRAC(1:3) .EQ. 'VST') THEN
   CVAR = 'n'
ELSE IF (TRAC(1:3) .EQ. 'PRE') THEN
   CVAR = 'o'
ELSE IF (TRAC(1:3) .EQ. 'STR') THEN
   CVAR = 'p'
ELSE IF (TRAC(1:5) .EQ. 'ICE F') THEN
   CVAR = 's'
ELSE IF (TRAC(1:5) .EQ. 'ICE T') THEN
   CVAR = 't'
ELSE
C
IF (TRAC(1:3) .EQ. 'SAL') THEN
   PVAR2 = 1
ELSE IF (TRAC(1:3) .EQ. 'TEM') THEN
   PVAR2 = 2
ELSE IF (TRAC(1:3) .EQ. 'U V') THEN
   PVAR2 = 3
ELSE IF (TRAC(1:3) .EQ. 'V V') THEN
   PVAR2 = 4
ENDIF

C
IF (DEPVAR(1:3) .EQ. 'LAT') THEN
   PVAR1 = 1
ELSE IF (DEPVAR(1:3) .EQ. 'LON') THEN
   PVAR1 = 2
ELSE IF (DEPVAR(1:3) .EQ. 'DEP') THEN
   PVAR1 = 3
ENDIF

C
PA = 3 * (PVAR2 - 1) + PVAR1
CVAR = CHAR(96 + PA)
ENDIF
OUTFIL = A // CVAR // LETT // D
PRINT *, ''
PRINT *, 'output file = ', OUTFIL PRINT *, ''
IFAIL = 0
RETURN
END

SUBROUTINE HEADER2(OP, TRAC, DEPVAR, OPFORM, NRUN)

Subroutine to write headers to the output files.

COMMON /TSTEP/ NDFIR, NDLAS, NDINC
CHARACTER TRAC(*), OPFORM(*), NRUN(*)
CHARACTER*9 DEPVAR, FROM(3), INCR(3), TO(3), QUAN(3)
INTEGER OP, N0P(3)

C Establish details for header
C
IF ((TRAC(1:3) .EQ. 'STR') .OR. (TRAC(1:3) .EQ. 'PRE')) THEN
  QUAN(1) = 'LONGITUDE'
  QUAN(2) = 'LATITUDE'
  FROM(1) = '0.0000'
  FROM(2) = '-78.875'
  INCR(1) = '0.5'
  INCR(2) = '0.25'
  TO(1) = '359.5'
  TO(2) = '-24.375'
  NOP(1) = 720
  NOP(2) = 219
ELSE IF (TRAC(1:3) .EQ. 'ICE') THEN
  QUAN(1) = 'LONGITUDE'
  QUAN(2) = 'LATITUDE'
  FROM(1) = '0.0000'
  FROM(2) = '-78.875'
  INCR(1) = '0.5'
  INCR(2) = '0.25'
  TO(1) = '359.5'
  TO(2) = '-24.375'
  NOP(1) = 720
  NOP(2) = 220
ELSE IF (TRAC(1:3) .EQ. 'TEM' .OR. TRAC(1:3) .EQ. 'SAL') THEN
  IF (DEPVAR(1:3) .EQ. 'LAT') THEN
    QUAN(1) = 'LONGITUDE'
    QUAN(2) = 'DEPTH'
    FROM(1) = '0.0000'
    FROM(2) = '1.0000'
    INCR(1) = '0.5'
    INCR(2) = '1.0'
  ELSE
    QUAN(1) = 'LONGITUDE'
    QUAN(2) = 'LATITUDE'
    FROM(1) = '0.0000'
    FROM(2) = '-78.875'
    INCR(1) = '0.5'
    INCR(2) = '0.25'
    TO(1) = '359.5'
    TO(2) = '-24.375'
    NOP(1) = 720
    NOP(2) = 220
  END IF
ENDIF
TO(1) = ' 359.5 '
TO(2) = ' 32 '
NOP(1) = 720
NOP(2) = 32
ELSE IF (DEPVAR(1:3) .EQ. 'LON') THEN
  QUAN(1) = ' LATITUDE'
  QUAN(2) = ' DEPTH'
  FROM(1) = ' 78.875 '
  FROM(2) = ' 1 '
  INCR(1) = ' 0.25 '
  INCR(2) = ' 1 '
  TO(1) = ' -24.125 '
  TO(2) = ' 32 '
  NOP(1) = 220
  NOP(2) = 220
ELSE IF (DEPVAR(1:3) .EQ. 'DEP') THEN
  QUAN(1) = 'LONGITUDE'
  QUAN(2) = ' LATITUDE'
  FROM(1) = ' 0. '
  FROM(2) = ' -78.875 '
  INCR(1) = ' 0.5 '
  INCR(2) = ' 0.25 '
  TO(1) = ' 359.5 '
  TO(2) = ' -24.125 '
  NOP(1) = 720
  NOP(2) = 220
ENDIF

ELSE IF (TRAC(3:10) .EQ. 'VELOCITY') THEN
  IF (DEPVAR(1:3) .EQ. 'LAT') THEN
    QUAN(1) = 'LONGITUDE'
    QUAN(2) = ' DEPTH'
    FROM(1) = ' 0.25 '
    FROM(2) = ' 1 '
    INCR(1) = ' 0.5 '
    INCR(2) = ' 1 '
    TO(1) = ' 359.75 '
    TO(2) = ' 32 '
    NOP(1) = 720
    NOP(2) = 32
  ELSE IF (DEPVAR(1:3) .EQ. 'LON') THEN
    QUAN(1) = ' LATITUDE'
    QUAN(2) = ' DEPTH'
    FROM(1) = ' -78.750 '
    FROM(2) = ' 1 '
    INCR(1) = ' 0.25 '
    INCR(2) = ' 1 '
    TO(1) = ' -24.250 '
    TO(2) = ' 32 '
    NOP(1) = 219
    NOP(2) = 32
  ELSE IF (DEPVAR(1:3) .EQ. 'DEP') THEN

```
QUAN(1) = 'LONGITUDE'
QUAN(2) = 'LATITUDE'
FROM(1) = '0.25'
FROM(2) = '-78.750'
INCR(1) = '0.5'
INCR(2) = '0.25'
TO(1) = '359.75'
TO(2) = '-24.250'
NOP(1) = 720
NOP(2) = 219
ENDIF
END

C
QUAN(3) = 'TIMESTEP'
NOP(3) = 1
WRITE (FROM(3), '(19)') NDFIR
WRITE (INCR(3), '(19)') NDINC
WRITE (TO(3), '(19)') NDLAS
IF (DEPVAR(I:3) .EQ. 'STR') THEN
WRITE (OP, 5101) TRAC, OPFORM
ELSE
WRITE (OP, 5100) TRAC, DEPVAR, OPFORM
ENDIF
WRITE (OP, 5102) NRUN
WRITE (OP, 5103) (I, I = 1, 3)
WRITE (OP, 5104) (QUAN(I), I = 1, 3)
WRITE (OP, 5105) (FROM(I), I = 1, 3)
WRITE (OP, 5106) (INCR(I), I = 1, 3)
WRITE (OP, 5107) (TO(I), I = 1, 3)
WRITE (OP, 5108) (NOP(I), I = 1, 3)

C
5100 FORMAT ('VARIABLE :',A15,2X,A9,T41,'FORMAT :',A2)
5101 FORMAT ('VARIABLE :',A15,T41,'FORMAT :',A2)
5102 FORMAT ('MODEL :',A15,'FORMAT :',A2)
5103 FORMAT ('INDEX :',9X,'COMMENTS :',A50)
5104 FORMAT ('QUANTITY :',6X,'A9',':',A9,':',A9,':A9,':
5105 FORMAT ('FROM :',6X,'A9',':',A9,':',A9,':',A9,':
5106 FORMAT ('INCREMENT :',6X,'A9',':',A9,':',A9,':',A9,':
5107 FORMAT ('TO :',6X,'A9',':',A9,':',A9,':',A9,':
5108 FORMAT ('NO.OF POINTS :',2X,'A9',':',A9,':',A9,':',A9,':
RETURN
END

SUBROUTINE ASCOUT (ARRAY, IDIM, ID, JD, VMASK, NCHAR, NOUT)

C Subroutine to encode a section of an array as sets of 'NCHAR'
C printable characters, and write as a formatted card-image dump.
C (Uses ASCII characters 0-9, A-Z, a-z and brackets)
C
C ARRAY - 2-D array of values to be converted
IDIM - declared I-dimension of array in calling programme
ID,JD - specify section of array to be converted
VMASK - 4-element array whose values indicate 'masked' points.
Such points are denoted by one of the 4 possible combinations of full stop and comma, padded out to NCHAR characters by repetition of the last character of the pair.
These values are ignored in finding max and mins for scaling.
The VMASK values are normally much larger than other values
NCHAR - Number of characters to be used to represent an array value
NOUT - Fortran channel number of output dataset.

This version (internally declared character array) 14/12/1988
Modified to allow for four types of masked point 01/02/1989

Internal parameters:

LRECL - Maximum length of data record to be output
NASCC - Number of different ASCII characters used in representation of numbers (at unmasked points)
NCMAX - Maximum number of characters which can be used to represent an array element.

INTEGER LRECL,NASCC,NCMAX

PARAMETER ( LRECL=80, NASCC=64, NCMAX=5 )

Local variables

INTEGER ICODE(NCMAX),IDIM,ID,JD,NCHAR,NOUT,
& I,J,NNUM,IC,INTEG,NCBUFF,LINLEN,MTYPE
REAL ARRAY(IDIM,JD),VMASK(4)
REAL FMIN,FMAX,RANGE,ARANG,SCALE
CHARACTER*1 ASCARR(LRECL),LKUP(NASCC),CMASK(2),MASK(NCMAX,4)
CHARACTER*(NASCC) CHAREP

EQUIVALENCE (CHAREP(1:1),LKUP(1))

Specify the NASCC characters to be used in the number representation, and the characters denoting masked points

CHAREP(1:10) = '0123456789'
CHAREP(11:36) = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
CHAREP(37:62) = 'abcdefghijklmnopqrstuvwxyz' CHAREP(63:64) = 'Q'
CMASK(1)='.'
CMASK(2)=','

Write out coding info in first data record
(write warning to unit 6 if it won't fit)

IF ( NASCC.GT.72 ) WRITE(6,50) NASCC
50 FORMAT(1X,'**ASCCOUT WARNING: OVERLENGTH CODING RECORD,'
& ' NASCC =',I3)
WRITE(OUT,'(I4,1X,2A1,1X,72A1)') NASCC,CMASK,(LKUP(I),I=1,NASCC)
C
C Check input value of NCHAR
C
IF ( NCHAR.LT.2 .OR. NCHAR.GT.NCMAX ) THEN
WRITE(6,100) NCHAR
100 FORMAT(/,2X,'**ASCIIT WARNING: ROUTINE CALLED',
& ' WITH INCORRECT NCHAR =',I4)
RETURN
END IF
C
C Check input values of VMASK are all different,
otherwise masking will be ambiguous
C
DO 110 J=1,3
DO 110 I=J+1,4
IF ( VMASK(I).EQ.VMASK(J) ) WRITE(6,120) I,J
110 CONTINUE
120 FORMAT(/,2X.'**ASCIIT WARNING: VMASK(',I1.') = VMASK(',I1,')',
& '/,2X.'**MASKING PRODUCED WILL BE AMBIGUOUS')
C
C Create the 4 types of MASK, including padding characters
C
DO 130 IC=1,NCHAR
   MASK(IC,1) = CMASK(1)
   MASK(IC,2) = CMASK(1)
   MASK(IC,3) = CMASK(2)
   MASK(IC,4) = CMASK(2)
130 CONTINUE
   MASK(2,2) = CMASK(2)
   MASK(2,3) = CMASK(1)
C
C Establish range of data and scaling for conversion
C (typical size of values assumed O(10**5) )
C
FMAX = -9999999.9
FMIN = 9999999.9
DO 150 I=1,ID
  DO 140 J=1,ID
     DO 130 MTYPE=1,4
        IF ( ARRAY(I,J).EQ.VMASK(MTYPE) ) GOTO 145
    130 CONTINUE
     FMIN = MIN( FMIN,ARRAY(I,J) )
     FMAX = MAX( FMAX,ARRAY(I,J) )
140 CONTINUE
150 CONTINUE
IF ( FMAX.LT.-99999.9 .OR. FMIN.GT.99999.9 )
& WRITE(6,200) FMIN,FMAX
200 FORMAT(/,2X,'**ASCIIT WARNING: LARGE +VE MINIMUM OR LARGE',
& '/,2X,'-VE MAXIMUM VALUE',/2X,'FMAX, FMIN = ',1P,2E16.5)
NNUM = ID*JD
WRITE(NOUT,'(1P,2E20.12,4I10)') FMIN,FMAX,ID,JD,NNUM,NCHAR
ARANG = REAL( NASCC**NCHAR - 1 )
RANGE = FMAX - FMIN
SCALE = ARANG/RANGE
IF ( INT(SCALE).LT.1 ) WRITE(6,220) SCALE
220 FORMAT(2X,'**ASCOUT WARNING: SCALE = ',1P,E14.5)
C
IF ( (RANGE*1.0E10).LT.1.0E0 ) THEN
C
WRITE(NOUT,250)
250 FORMAT( '**ASCOUT WARNING: FIELD APPROX. CONSTANT, ',
& ' NOT CHARACTER CODED')
C
ELSE
C
Scale array and encode as NCHAR printable characters
C
NCBUFF = 0
IF ( NCHAR.EQ.3 ) LINLEN=78
IF ( NCHAR.NE.3 ) LINLEN=80
DO 500 J=1,JD
DO 350 I=1,ID
DO 300 MTYPE = 1,4
IF (ARRAY(I,J).EQ.VMASK(MTYPE) ) THEN
C TYPE MTYPE MASKED POINT; COPY FROM MASK(NCMAX,J,MTYPE)
DO 300 IC = 1,NCHAR
ASCARR(NCBUFF+IC) = MASK(IC,MTYPE)
300 CONTINUE
GOTO 450
END IF
350 CONTINUE
C
Normal point; encode as NCHAR characters
C
INTEG=NINT( (ARRAY(I,J)-FMIN)*SCALE )
DO 400 IC=NCHAR,1,-1
ICODE(IC) = 1 + MOD( INTEG, NASCC )
ASCARR(NCBUFF+IC) = LKUP( ICODE(IC) )
INTEG = INTEG/NASCC
400 CONTINUE
450 CONTINUE
NCBUFF = NCBUFF + NCHAR
C
IF ( NCBUFF.EQ.LINLEN ) THEN
C
Channel NOUT (card-image format)
C
IF ( NCHAR.NE.3 ) THEN
WRITE(NOUT,'(80A1)') (ASCARR(IC),IC=1,NCBUFF)
ELSE
    WRITE(NOUT,'(1X,78A1,1X)') (ASCARR(IC),IC=1,NCBUFF)
END IF
NCBUFF = 0
END IF

C 500 CONTINUE
C C Flush character buffer if not empty
C IF (NCBUFF.NE.0) THEN
    IF (NCHAR.NE.3) THEN
        WRITE(NOUT,'(80A1)') (ASCARR(IC),IC=1,NCBUFF)
    ELSE
        WRITE(NOUT,'(1X,78A1,1X)') (ASCARR(IC),IC=1,NCBUFF)
    END IF
    NCBUFF = 0
END IF
RETURN
END