CERES Software Bulletin 99-01

October 25, 1999

Changes to the MOA Interface Software for ECMWF Data

Lisa Coleman (l.h.coleman@larc.nasa.gov)
Ed Kizer (e.a.kizer@larc.nasa.gov)
Joe Stassi (j.c.stassi@larc.nasa.gov)

1.0 Purpose

The Clouds and the Earth's Radiant Energy System (CERES) is now incorporating meteorological data from the European Centres for Medium-Range Weather Forecasting (ECMWF). As a result of this decision, modifications to the interface used by the individual CERES subsystems to access the Meteorological, Ozone, and Aerosol (MOA) data are necessary. The purpose of this CERES Software Bulletin is to describe these modifications.

2.0 Introduction

With previously used sources of meteorological data, a MOA product existed for every hour. Due to the high resolution, and subsequent high volume, of the ECMWF meteorological data, however, an ECMWF-based MOA product exists only every six hours. Two six-hourly MOA files are now needed by the subsystems to obtain the temporally interpolated data once obtainable from the hourly MOA files. The majority of the software modifications driven by the change to six-hourly files have been incorporated into the CERESlib module dedicated to input and output operations on MOA files, moa_io.f90. Some changes to individual subsystem software, however, are still necessary. While the new ECMWF-based MOAs cannot be read with current interfaces, existing MOAs accessed with the current interfaces can also be accessed with the new interfaces.

Two rules governed the changes made to accommodate the six-hourly MOA files. The first was that the same logic must also work for hourly MOA files. The second was that no decision-making logic regarding the source of the meteorological data was to be incorporated in the individual subsystems' software. The changes described in this bulletin adhere to these two rules.

3.0 Opening MOA Files

3. 1 Hourly Subsystems

The ECMWF-based MOA files exist for hours 00, 06, 12, and 18 of each day. Subsystems processing data for one of those hours and using ECMWF-based MOA data need only open that MOA file. For the remaining hours in the day, two ECMWF-based MOAs need to be opened.

MOA files based on sources other than ECMWF exist for every hour of the day. Subsystems processing data for any hour using these MOA files need only open the MOA file corresponding to the hour being processed.

3.1.1 Modifications to the Hourly Subsystems' PCF

Previously, only one version of the Logical Identifier (LID) in the Process Control File (PCF) was necessary. Now, three versions of the same LID are necessary. The following lines in the PCF are needed:

MOA LID | MOA file name for hour to be processed | Version 1
MOA LID | MOA file name for the previous six-hour file | Version 2
MOA LID | MOA file name for the next six-hour file | Version 3

Subroutine MOA_Open_Wrapper in the moa_io module will first try to open the file associated with Version 1 on the first line. This will be successful if the requested MOA is not ECMWF-based, or is ECMWF-based for hour 00, 06, 12, or 18. If not successful, then Subroutine MOA_Open_Wrapper will open the MOA files associated with Versions 2 and 3 on the second and third lines, respectively. Subroutine MOA_Open_Wrapper assumes the Version numbers to be consecutive, as in the below example for a subsystem processing January 1, 1998, hour 03.

1000 | CER_MOA....1998010103 | /CERES/sarb/data/out_comp/data/regridmoa/ | | | | 1
1000 | CER_MOA....1998010100 | /CERES/sarb/data/out_comp/data/regridmoa/ | | | | 2
1000 | CER_MOA....1998010106 | /CERES/sarb/data/out_comp/data/regridmoa/ | | | | 3

3.1.1.1 Modifications to the Hourly Subsystems' ASCII File Generators

To generate the three lines discussed in Section 3.1.1, changes to the subsystems' ASCII file and PCF generators are necessary. The following example is taken from the Instantaneous SARB Subsystem's ASCII file generator. In this example, CurrDay represents the current day in the format *yyyymmdd*, and NextDay represents the following day also in the format *yyyymmdd*. Parameter names in this example will need to be tailored for each subsystem's ASCII file generators.

```
# Build MOA filename for current hour
#
set string12 = $inSS12\_$inPS12\_$CCode12
set MOA_INSTANCE = $string12\.$CurrDay$CERHour
#
#
# Build MOA file names for temporal interpolation
#
if ($CERHour == 0 || $CERHour == 6 || $CERHour == 12 || $CERHour == 18) then
   set MOA_INSTANCEA = $MOA_INSTANCE
   set MOA_INSTANCEB = $MOA_INSTANCE
else if ($CERHour > 0 && $CERHour < 18) then
   if ($CERHour > 0 && $CERHour < 6) then
      set CERHourA = "00"
      set CERHourB = "06"
```

```
   else if ($CERHour > 6 && $CERHour < 12) then
      set CERHourA = “06“
      set CERHourB = “12“
   else if ($CERHour > 12 && $CERHour < 18) then
      set CERHourA = “12“
      set CERHourB = “18“
   endif
   set MOA_INSTANCEA =  $string12\.$CurrDay$CERHourA
   set MOA_INSTANCEB =   $string12\.$CurrDay$CERHourB
else if ($CERHour > 18 && $CERHour <= 23) then
   set CERHourA = “18“
   set CERHourB = “00“
   set MOA_INSTANCEA =   $string12\.$CurrDay$CERHourA
   set MOA_INSTANCEB =   $string12\.$NextDay$CERHourB
endif

#
# Redirect file names to PCF generator
#
echo "SS5.0_Inputfile.g = CER_MOA_$MOA_INSTANCE"  >> $pcf_input
echo "SS5.0_Inputfile.h = CER_MOA_$MOA_INSTANCEA" >> $pcf_input
echo "SS5.0_Inputfile.i = CER_MOA_$MOA_INSTANCEB" >> $pcf_input
```

In the above example, MOA_INSTANCE corresponds to LID 1000, Version 1, in Section 3.1.1. MOA_INSTANCEA and MOA_INSTANCEB correspond to Versions 2 and 3, respectively. This logic can be used whether or not the MOA files are ECMWF-based. If the MOA files are not ECMWF-based, MOA_INSTANCEA and MOA_INSTANCEB are simply ignored by the moa_io module.

3.1.1.2 Modifications to the Hourly Subsystems’ PCF Generators

To include the Versions 2 and 3 of the MOA filename in the PCF, modifications to the hourly subsystems’ PCF generators are also necessary. The following example is from the Instantaneous SARB Subsystem’s PCF generator. Previously, only the first line existed.

```
#
# Include MOA file names in PCF
#
echo "1000|$inputfileg|$InDir4||||1">> $PCFile
echo "1000|$inputfileh|$InDir4||||2">> $PCFile
echo "1000|$inputfilei|$InDir4||||3">> $PCFile
```

In the above example, the value of inputfileg corresponds to the value of SS5.0_Inputfile.g in the example given for the ASCII file generator. Likewise, the values of inputfileh and inputfilei corre-

spond to SS5.0_Inputfile.h and SS5.0_Inputfile.i, respectively. Each subsystem needs to ensure that any existing logic for defining the value of inputfileg is expanded to also define the values inputfileh, and inputfileg.

3.1.2 Modifications to the Hourly Subsystems' Fortran Call to MOA_Open

Previously, the hourly subsystems opened the MOA files with a call to Subroutine MOA_Open. Subroutine MOA_Open_Wrapper has been added to interface with Subroutine MOA_Open. With one call, Subroutine MOA_Open_Wrapper determines how many MOA files to open, and to then open them, so that the individual subsystems do not need to write and maintain such logic. The hourly subsystems should now access Subroutine MOA_Open_Wrapper.

Subroutine MOA_Open_Wrapper returns an array, dimension 2, for the MOA unit number instead of a scalar value as is returned by Subroutine MOA_Open. Previously, the subsystem needed a declaration statement for a scalar integer. To use the ECMWF-based MOA files, the subsystem needs a declaration statement for an array, as in the following example from the Instantaneous SARB Subsystem.

```
INTEGER, SAVE :: MOA_Unit ( 2)                    ! New
```

If a MOA file for the current hour does exist, then the value of MOA_Unit (1) corresponds to the file associated with LID1000, Version 1, in the PCF (See Section 3.1.1), and MOA_Unit (2) is assigned a default value by the moa_io module. If there is not a MOA file for the current hour, MOA_Unit (1) corresponds to the file associated with Version 2 in the PCF, and MOA_Unit (2) corresponds to the file associated with Version 3 in the PCF.

The calling arguments for the new routine, Subroutine MOA_Open_Wrapper, are also different from those for the existing routine, Subroutine MOA_Open. Now, in addition to the unit number being an array, the hour number being processed must be included. IOS, the argument indicating the statement success or failure of the file opening process, is optional. If omitted and an error occurs, processing is terminated by Subroutine MOA_Open_Wrapper through the Toolkit error message utility; otherwise, control is returned to the calling routine. An optional input argument is the LID version number for the first MOA file listed in the PCF (corresponding to $MOA_INSTANCE in Section 3.1.2). If the version number is not included in the call statement, then the first version number is assumed to be equal to 1. Another change is that the argument indicating the read or write action has been eliminated.

```
USAGE: CALL MOA_Open_Wrapper (MOA_LID, HourNum,                  & ! I
                              MOA_Unit, MOA_Header, MOA,         & ! O
                                                    IOS,         & ! O (Opt)
                                                    Version)       ! I (Opt)
```

Argument Information:

MOA_LID	:	INPUT - Integer; Logic ID from PCF
HourNum	:	INPUT - Integer; Current hour number [0.. 23]

MOA_Unit	:	OUTPUT - Integer Array; Toolkit-assigned unit numbers for the MOA files
MOA_Header	:	OUTPUT - MOA_Header_Type MOA Header record contents from first MOA file (corresponding to Version 1 or Version 2) opened
IOS	:	OUTPUT - Integer, OPTIONAL Success status returned from CERESlib Sbr. OpenFile. If omitted, Sbr. MOA_Open_Wrapper controls processing termination if a file opening error occurs.
Version	:	INPUT - Integer, OPTIONAL Logic ID version number corresponding with first input MOA file listed in PCF. If omitted, a value of 1 is assumed.

3.2 PMOA Processor

3.2.1 Modifications to the PMOA Processor's PCF

The PCF for the PMOA Processor has entries for MOA files corresponding to each possible hour of a month, plus overlapping hours from the previous and succeeding months. Prior to the introduction of ECMWF data, 12 hours of MOA data (hours 00 through 11) for the first day of the succeeding month were necessary. With the six-hourly ECMWF-based MOA files, data from 13 hours (00 to 12) need to be included. This is an addition of one more input file that needs to be indicated in the PCF.

3.2.2 Modifications to the PMOA Processor's ASCII File Generators and PCF Generators

The PMOA Processor's ASCII file generator and PCF generator need to be modified to include one more MOA input file, as indicated in Section 3.2.1.

3.2.3 Modifications to the PMOA Processor's Opening of MOA files

Due to restrictions on the number of files that may be open at one time, the PMOA Processor uses two nested DO loops to manage the opening and processing of MOA data for an entire month. Prior to the introduction of ECMWF data, the Temporal Interpolation and Spatial Averaging (TISA) Working Group designed the inner DO loop to open and process six hours of data, one hour at a time. The PMOA Processor was also designed to skip processing of all hours managed by the inner DO loop if Subroutine MOA_Open returned a status value indicating an unsuccessful open for any one of the hours.

To avoid attempting to open a MOA file every hour when ECMWF-based MOAs are used, while simultaneously maintaining the capability to open a MOA file every hour when MOA files based on data from other sources are used, modifications to the inner DO statement are necessary. The addition of an increment variable ensures that the loop is only executed for the hours for which there are existing MOA files. This increment variable is set to the number of hours between MOA files, and is defined by the moa_io module Subroutine MOA_HrIncr_Stats based on the grid index value obtained from the MOA header. If the MOAs used are ECMWF-based, then the increment

variable is set to a value of six. If DAO-GEOS2 MOA files are used, then the value of the increment variable is equal to one. The MOA file version number, passed to the existing Subroutine MOA_Open, is also incremented for each new file to be opened according to the value of the increment variable.

The MOA unit numbers returned from Subroutine MOA_Open are stored in an array. Prior to the use of ECMWF-based MOA files, this array was dimensioned at 24, the number of hours in a day. With the introduction of the six-hourly ECMWF-based MOA files this array must now be dimensioned at 25 so as to include hour 00 of the next day, necessary for the temporal interpolation for the last six hours of a day. In the case of ECMWF-based MOA files, unit numbers corresponding to unavailable hours are set to a default value.

4.0 Reading MOA Files

4.1 Hourly Subsystems

For the hourly subsystems, reading data from the MOA file(s) now requires additional information to what was previously required. This additional information includes an additional MOA_Unit for the second MOA file that has been opened, and the hour. Previously, the hourly subsystems read the MOA files with a call to Subroutine MOA_Read. Subroutine MOA_Read_Wrapper has been added to interface with Subroutine MOA_Read. With one call, Subroutine MOA_Read_Wrapper determines how many MOA files to read, and to then reads them, so that the individual subsystems do not need to write and maintain such logic. The hourly subsystems should now access Subroutine MOA_Read_Wrapper.

If the MOA files are ECMWF-based, Subroutine MOA_Read_Wrapper retrieves the MOA data from the two MOA files opened by Subroutine MOA_Open_Wrapper. The data are temporally interpolated to the hour specified, returning one MOA structure to the subsystem's calling routine. If the MOA data is not ECMWF-based, or the hour is 00, 06, 12, or 18, no temporal interpolation takes place. The decision on whether or not to interpolate occurs in the MOA_Read_Wrapper routine, and is based on the value of the second MOA_Unit, assigned by Subroutine MOA_Open_Wrapper (see Section 3.1.3).

As in Subroutine MOA_Open_Wrapper, the IOS argument for Subroutine MOA_Read_Wrapper is optional. If IOS is present and an error occurs reading the MOA data, control is returned to the calling argument; otherwise, Subroutine MOA_Read_Wrapper terminates processing through the Toolkit error message utility.

While passing the MOA grid index obtained from the MOA header is necessary for Subroutine MOA_Read, it is not included in the argument list for Subroutine MOA_Read_Wrapper.

With the use of the MOA_Read_Wrapper interface contained in the moa_io module, the MOA data can be accessed either according to a CERES nested region number, or according to latitudinal and longitudinal coordinates. To access the MOA data corresponding to a predetermined CERES nested grid region number, the following calling sequence is necessary.

```
CALL MOA_Read_Wrapper (MOA_Unit, HourNum, MOA_Region,          & ! I
                                                    MOA,       & ! O
                                                        IOS)     ! O (Opt)
```

To access the MOA data corresponding to specific colatitudinal and longitudinal coordinates, the following calling sequence is necessary.

```
CALL MOA_Read_Wrapper (MOA_Unit, HourNum, Lat, Long,           & ! I
                                                    MOA,       & ! O
                                                        IOS)     ! O (Opt)
```

Argument Information:

MOA_Unit	:	INPUT - Integer Array Toolkit-assigned unit numbers for the MOA files returned from Sbr. MOA_Open_Wrapper
HourNum	:	INPUT - Integer Current hour number [0.. 23]
Lat	:	INPUT - Real Colatitudinal coordinate of sample [0.. 180]
Long	:	INPUT - Real Longitudinal coordinate of sample [0.. 360]
MOA_Region	:	INPUT - Integer MOA_Region number
MOA	:	OUTPUT -MOA_Type MOA record corresponding to region number MOA_Region (or the values of Lat and Long)
IOS	:	OUTPUT - Integer, OPTIONAL Success status returned from Fortran READ statement. If omitted, Sbr. MOA_Read_Wrapper controls processing termination upon encountering a file read error.

4.2 PMOA Processor

The PMOA Processor will continue to retrieve MOA data via the moa_io module Subroutine MOA_to_TISA. Changes to the calling sequence include passing in the MOA_Unit array in its entirety instead of passing in only the MOA_Unit array element corresponding to the specified hour, and passing in the hour.

```
CALL MOA_to_TISA (TISAregnum, UnitNums, Hour, GridIdx,         & ! I
                                                    TISA)      ! O
```

Argument Information:

TISAregnum	:	INPUT - Integer TISA NESTED Region number

UnitNums : INPUT - Integer Array
Toolkit-assigned unit numbers for the MOA files returned from Sbr. MOA_Open_Wrapper

Hour : INPUT - Integer
Requested hour of day of MOA data [0 .. 23]

GridIdx : INPUT - Integer
MOA grid index, obtained from MOA header

TISA : OUTPUT - MOA_TYPE
A MOA type structure for the input TISA region

5.0 Closing MOA Files

5.1 Hourly Subsystems

Since there now can be multiple MOA files opened by the hourly subsystems, calls to MOA_Close need to be replaced with a call to Subroutine MOA_Close_Wrapper. The calling arguments include the MOA unit number array instead of a scalar argument. Subroutine MOA_Close_Wrapper determines whether to close one or two MOA files based on the value of the second MOA unit number array element.

As in Subroutines MOA_Open_Wrapper and MOA_Read_Wrapper, the IOS argument for Subroutine MOA_Close_Wrapper is optional. If IOS is present and an error occurs closing the MOA data, control is returned to the calling argument; otherwise, Subroutine MOA_Close_Wrapper terminates processing through the Toolkit error message utility.

```
CALL MOA_Close_Wrapper (MOA_Unit,                    & ! I
                        IOS)                           ! O (Opt)
```

Argument Information:

MOA_Unit : OUTPUT - Integer Array
Toolkit-assigned unit numbers for the MOA files returned from Sbr. MOA_Open_Wrapper

IOS : OUTPUT - Integer, OPTIONAL
Success status returned from Fortran CLOSE statement If omitted, Sbr. MOA_Close_Wrapper controls processing termination upon encountering a file read error.

5.2 PMOA Processor

The PMOA Processor will continue to close one MOA file at a time, thus the existing call to Subroutine MOA_Close should remain the same.