UNIFIED MODEL DOCUMENTATION PAPER
No S5

C Utility Routines Used within the Unified Model

by

A. Dickinson
P. Burton

Version 4

04/09/1998

Model version 4.5

Numerical Weather Prediction
Meteorological Office
London Road
BRACKNELL
Berkshire
RG12 2SZ
United Kingdom

Modification Record		
Document version	**Author**	**Description.....**
2	R.Rawlins	Include new 'C' routine FLUSH_BUFFER.
3	P. Burton	Many new functions and Cray extensions added. General tidy up.
4	P. Burton	Added new 32 bit functions.

1. Introduction

This document describes a suite of utility routines written in C which are available for use within the Unified Model. The routines are designed to be called from Fortran, providing extensions to the Fortran standard which enhance the portable nature of the model. A full description of how to call the routines is provided in the Appendix.

By far the largest number of these routines are in support of random access input-output. Other routines provide standard interfaces to date and time functions and allow interaction with the operating environment.

2. Input-output

All data files used by the Unified Model, including dump, ancillary, fieldcos and acobs files, are unstructured and random access. This means that the files contain no Fortran style record blocks and a user program can arbitrarily position the file pointer to select the next address for a data transfer. If the program does not reset the pointer then, by default, the system automatically adjusts the file pointer after each read or write operation to point to the next sequential piece of data on the file.

The utility routines allow data to be transferred in blocks of bytes or words. The data type is transparent and blocks are transferred as an unmodified bit stream of data. Those utility routines relating specifically to the transfer of blocks of words emulate the Cray input-output extensions BUFFER IN and BUFFER OUT. A Fortran-like unit number needs to be specified when the file is opened. This must be in the range 1 - 199. The equivalent Fortran unit numbers can also be used at the same time to access other files, since there is no conflict between C and Fortran input-output, but it is not recommended because of the potential for confusion and error.

The names of the routines can depend on whether the code is being run in MPP (Massively Parallel Processor, distributed memory) mode (selected by nupdate `*DEF,MPP`) , or normal single processor mode. Some routines are specifically for high performance I/O on particular platforms (Cray), these require certain nupdate *DEF switches to function correctly.

The input-output routines available are shown in table 1. (N.B. All routine names listed here are in the format Uppercase, no trailing underscore. A discussion of the format of routine names appears in section 4.)

Routine Name	*DEF required	Brief Description
BUFFIN BUFFIN_SINGLE	MPP	Reads in a block of words from a file
BUFFOUT BUFFOUT_SINGLE	MPP	Writes out a block of words to a file
BUFFIN8		Reads a block of bytes from a file
BUFFOU8		Writes a block of bytes from a file
BUFFIN32		Reads a block of 32bit words from a file
BUFFO32		Writes a block of 32bit words to a file
CLOSE_ALL_FILES	CRI_FFIO	Close all files safely (Cray FFIO only)
FILE_OPEN OPEN_SINGLE	MPP	Opens a random access file
FILE_CLOSE CLOSE_SINGLE	MPP	Closes a random access file
FLUSH_ALL_FILES	CRI_FFIO	Flushes all files to disk (Cray FFIO only)
FLUSH_BUFFER		Flush a file to disk
GETPOS		Get current word address of file pointer
GETPOS8		Get current byte address of file pointer
GETPOS32		Get current 32bit word address of file pointer
GET_FILE		Get filename associated with unit number
SETPOS SETPOS_SINGLE	MPP	Set file pointer to new word address
SETPOS8		Set file pointer to new byte address
SETPOS32		Set file pointer to new 32bit word address
SET_DUMPFILE_ LENGTH	CRI_OPEN	Set preallocation file size (Cray FFIO only)

Table 1: Input/output routines

3. Exchanges with the environment

The routines shown in table 2 provide a Fortran interface to standard C functions, whose features are required by the Unified Model:

Routine Name	Brief Description
ABORT	Aborts a run of the model
DATE_TIME	Get current date and time
FORT_GET_ENV	Get the contents of an environment variable
SHELL	Execute UNIX command
WORD_LENGTH	Get the default word length used by the computer system

Table 2 : Fortran interface to useful C functions

4. Accessing C routines from Fortran (Portability issues)

The C routines are maintained under a single deck called PORTIO2A protected by *IF DEF,95_2A on the Unified Model program library.

4.1 Fortran / C calling interface

There are three common formats of writing C function names to make them callable from Fortran, which are shown in table 3. The appropriate format is selected by use of the nupdate DEF shown.

Example function	Format	nupdate *DEF	Computer
CFUNCTION	Upper case	<none> (Default)	Cray
cfunction	Lower case	C_LOW	HP
cfunction_	Lower case, postfixed underscore	C_LOW_U	SUN

Table 3 : C function name formats

4.2 Fortran / C argument type interface

Fortran routines pass three principal data types (or more correctly pointers to data) to the C functions, REAL, INTEGER and CHARACTER. The problem in interfacing between Fortran and C, is that the C type corresponding to a particular Fortran type is not the same on every machine.

i) REALs and INTEGERs

The C code contains a nupdate *DEF to control which C data type is used to map on to the Fortran REAL and INTEGER types:

nupdate *DEF	C type interfacing to Fortran REAL	C type interfacing to Fortran INTEGER
FRL8 (or CRAY)	double	long
<none> (Default)	float	int

ii) CHARACTER

By default the C code uses the C data type char to map to the Fortran CHARACTER data type. However, if nupdate *DEF CRAY is switched on, a special C type _fcd is used, which is converted into a C char type by the function _fcdtocp.

4.3 Access to C functions from non-UM code

Access to the C utility routines within the UM system is automatically supported by default. External use of the routines on a workstation may be achieved by the following lines of code. The environment variable PL points to the Unified Model program library and code.f contains the calling Fortran subroutines. (Note other *DEFs may be required for the correct Fortran / C interface, as discussed above).

```
cat > port.deck << \EOF
*ID MODS
*DEF 95_2A
*C PORTIO2A
EOF
update -p $PL -i port.deck -c port -a c
cc -c port.c
f77 code.f port.o
./a.out
```

APPENDIX A : Full Descriptions of C routines

ABORT

Call:

```
CALL ABORT()
CALL ABORT(MESSAGE)   (*DEF,CRAY_MSG)   (Cray systems only)
```

Arguments:

Variable	Type	Status	Description
MESSAGE	CHARACTER	IN	Abort message to be printed. Only valid on Cray systems when nupdate *DEF,CRAY_MSG has been set.

Description:

ABORT stops a run of the model, and ensures all files are closed properly, before the model exits. On Cray systems, if nupdate *DEF,CRAY_MSG has been set, a message can also be output, informing the user of the reason abort was called.

Return Codes:

None

BUFFIN
BUFFIN_SINGLE (*DEF,MPP)

Call:

```
CALL BUFFIN(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	INTEGER/REAL/ LOGICAL	OUT	Array to read data into
MAXLEN	INTEGER	IN	Number of words to be read
LENGTH	INTEGER	OUT	Number of words actually read
ERR	REAL	OUT	Return code

Description:

BUFFIN reads in MAXLEN words of data into array ARRAY from the random access file already opened on unit UNIT. Data is read in from the current position of the file pointer.

BUFFIN mimics the Cray extension BUFFER IN.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Read successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened
4.0	FFIO error - File not well formed (Cray systems with *DEF CRI_FFIO only)
5.0	Unspecified FFIO error (Cray systems with *DEF CRI_FFIO only)

BUFFOUT
BUFFOUT_SINGLE (*DEF,MPP)

Call:

```
CALL BUFFOUT(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	INTEGER/REAL/ LOGICAL	IN	Array to output
MAXLEN	INTEGER	IN	Number of words to be written
LENGTH	INTEGER	OUT	Number of words actually written
ERR	REAL	OUT	Return code

Description:

BUFFOUT writes out MAXLEN words of data from the array ARRAY to the random access file already opened on unit UNIT. Data is written from the current position of the file pointer.
BUFFOUT mimics the Cray extension BUFFER OUT.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Write successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened
4.0	FFIO error - File not well formed (Cray systems with *DEF CRI_FFIO only)
5.0	Unspecified FFIO error (Cray systems with *DEF CRI_FFIO only)

BUFFIN8

Call:

```
CALL BUFFIN8(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	CHARACTER	OUT	Array to read data into
MAXLEN	INTEGER	IN	Number of bytes to be read
LENGTH	INTEGER	OUT	Number of bytes actually read
ERR	REAL	OUT	Return code

Description:

BUFFIN8 reads in MAXLEN bytes of data into array ARRAY from the random access file already opened on unit UNIT. Data is read in from the current position of the file pointer. For portability reasons, it is essential that the Fortran array ARRAY is of type CHARACTER.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Read successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened

BUFFOU8

Call:

```
CALL BUFFOU8(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	CHARACTER	IN	Array to output
MAXLEN	INTEGER	IN	Number of bytes to be written
LENGTH	INTEGER	OUT	Number of bytes actually written
ERR	REAL	OUT	Return code

Description:

BUFFOU8 writes out MAXLEN bytes of data from the array ARRAY to the random access file already opened on unit UNIT. Data is written from the current position of the file pointer. For portability reasons, it is essential that the Fortran array ARRAY is of type CHARACTER.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Write successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened

BUFFIN32

Call:

```
CALL BUFFIN32(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	CHARACTER	OUT	Array to read data into
MAXLEN	INTEGER	IN	Number of 32bit words to be read
LENGTH	INTEGER	OUT	Number of 32bit words actually read
ERR	REAL	OUT	Return code

Description:

BUFFIN32 reads in MAXLEN 32bit words of data into array ARRAY from the random access file already opened on unit UNIT. Data is read in from the current position of the file pointer.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Read successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened

BUFFO32

Call:

```
CALL BUFFO32(UNIT,ARRAY,MAXLEN,LENGTH,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ARRAY(MAXLEN)	INTEGER*4/ REAL*4	IN	Array to output
MAXLEN	INTEGER	IN	Number of 32bit words to be written
LENGTH	INTEGER	OUT	Number of 32bit words actually written
ERR	REAL	OUT	Return code

Description:

BUFFO32 writes out MAXLEN 32bit words of data from the array ARRAY to the random access file already opened on unit UNIT. Data is written from the current position of the file pointer.

Return Codes:

The ERR variable contains a return code:

Value	Description
-1.0	Write successful
0.0	End of file
1.0	Unspecified error
2.0	FFIO error (Cray systems with *DEF CRI_FFIO only)
3.0	File not opened

CLOSE_ALL_FILES (*DEF CRI_FFIO) (Cray systems only)

Call:

```
CALL CLOSE_ALL_FILES()
```

Arguments:

None

Description:

Closes all units that have been opened using `FILE_OPEN`/`OPEN_SINGLE`. This routine only has any effect if nupdate *`DEF,CRI_FFIO` has been set.

Return Codes:

None

DATE_TIME

Call:

```
CALL DATE_TIME(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND)
```

Arguments:

Variable	Type	Status	Description
YEAR	INTEGER	OUT	Year with century
MONTH	INTEGER	OUT	Month (1-12)
DAY	INTEGER	OUT	Day (1-31)
HOUR	INTEGER	OUT	Hour (0-23)
MINUTE	INTEGER	OUT	Minute (0-59)
SECOND	INTEGER	OUT	Second (0-59)

Description:

DATE_TIME returns the current date and time.

Return Codes:

None

FILE_OPEN
OPEN_SINGLE (*DEF,MPP)

Call:

```
CALL FILE_OPEN(UNIT,FILE_NAME,CHAR_LEN,READ_WRITE,
               ENVIRON_VAR_FLAG,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
FILE_NAME	CHARACTER	IN	File name or Name of environment variable containing the file name.
CHAR_LEN	INTEGER	IN	Number of characters in FILE_NAME. Maximum of 80 characters.
READ_WRITE	INTEGER	IN	=0 : File opened for read only !=0 : opened for read/write
ENVIRON_VAR_FLAG	INTEGER	IN	=0 : FILE_NAME contains name of environment variable !=0 : FILE_NAME contains explicit file name
ERR	INTEGER	OUT	Return code

Description:

FILE_OPEN opens a random access file on unit UNIT. The file name may be specified explicitly or from a named environment variable.
If nupdate *DEF,CRI_OPEN (Cray machines only) has been set and READ_WRITE indicates a file opened for READ/WRITE and the file does not already exist, then FILE_OPEN will attempt to preallocate contiguous space for the file on the disk. The amount of space allocated can be controlled by the routine SET_DUMPFILE_LENGTH.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	File opened successfully
!=0	File not opened. See output for error message

FILE_CLOSE
CLOSE_SINGLE (*DEF,MPP)

Call:

```
CALL FILE_CLOSE(UNIT,FILE_NAME,CHAR_LEN,ENVIRON_VAR_FLAG,
                DELETE,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
FILE_NAME	CHARACTER	IN	File name or Name of environment variable containing the file name.
CHAR_LEN	INTEGER	IN	Number of characters in FILE_NAME. Maximum of 80 characters.
ENVIRON_VAR_FLAG	INTEGER	IN	=0 : FILE_NAME contains name of environment variable !=0 : FILE_NAME contains explicit file name
DELETE	INTEGER	IN	=0 : Do not delete file !=0 : Delete file
ERR	INTEGER	OUT	Return code

Description:

FILE_CLOSE closes the random access file on unit UNIT. The file name may be specified explicitly or from a named environment variable.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	File closed successfully
1	Error occurred during close. See output for error message

FLUSH_ALL_FILES (*DEF,CRI_FFIO) (Cray systems only)

Call:

```
CALL FLUSH_ALL_FILES()
```

Arguments:

None

Description:

FLUSH_ALL_FILES flushes the buffers for all open random access file units, ensuring the physical files on disk are up to date with all the I/O requests that have been made.

Return Codes:

None

FLUSH_BUFFER

Call:

```
CALL FLUSH_BUFFER(UNIT,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
ERR	INTEGER	OUT	Return code

Description:

FLUSH_BUFFER forces the last I/O buffer to be written to the random access file identified by UNIT, without waiting for the buffer to become full.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	Buffer flush successful
!=0	Buffer flush unsuccessful. Error code is return code from C function `fflush` and may be machine specific

FORT_GET_ENV

Call:

```
CALL FORT_GET_ENV(EV_NAME,EV_LEN,EV_CONTENTS,
                  EV_CONTENTS_LEN,ERR)
```

Arguments:

Variable	Type	Status	Description
EV_NAME	CHARACTER	IN	Name of environment variable
EV_LEN	INTEGER	IN	Length of environment variable's (EV_NAME) name
EV_CONTENTS	CHARACTER	OUT	Contents of environment variable EV_NAME
EV_CONTENTS_LEN	INTEGER	IN	Dimension of EV_CONTENTS
ERR	INTEGER	OUT	Return code

Description:

FORT_GET_ENV returns the contents of the environment variable whose name is contained within EV_NAME, into EV_CONTENTS.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	Successfully returned environment variable contents
-1	No environment variable with the name specified by EV_NAME was found

GETPOS

Call:

```
CALL GETPOS(UNIT,WORD_ADDRESS)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
WORD_ADDRESS	INTEGER	OUT	Number of words into file. Zero points to first word in the file

Description:

GETPOS returns the current position of the file pointer for the random access file identified by UNIT. The returned value is in terms of the number of words into the file.

Return Codes:

None

GETPOS8

Call:

```
CALL GETPOS8(UNIT,BYTE_ADDRESS)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
BYTE_ADDRESS	INTEGER	OUT	Number of bytes into file. Zero points to first byte in the file

Description:

GETPOS returns the current position of the file pointer for the random access file identified by UNIT. The returned value is in terms of the number of bytes into the file.

Return Codes:

None

GET_FILE

Call:

```
CALL GET_FILE(UNIT,FILENAME,FILE_LEN,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
FILENAME	CHARACTER	OUT	File name
FILE_LEN	INTEGER	IN	Dimension of FILENAME
ERR	INTEGER	OUT	Return code

Description:

GET_FILE returns the file name associated with unit UNIT. The contents of the environment variable UNIT??? is returned in the character string FILENAME. For unit numbers in the range 0-9, ??? is a string of the form "0?", unit numbers in the range 10-99 a string of form "??", and for unit numbers over 99, the string takes the form "???". For example, the following are valid environment variable names: UNIT08 UNIT22 UNIT105.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	Successfully retrieved file name
1	File name was too long to fit into variable FILENAME dimensioned with length FILE_LEN

SETPOS
SETPOS_SINGLE (*DEF,MPP)

Call:

```
CALL SETPOS(UNIT,WORD_ADDRESS,ERR)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
WORD_ADDRESS	INTEGER	IN	Number of words into file. Zero points to the first word in a file.
ERR	INTEGER	OUT	Return code

Description:

SETPOS sets the word address file pointer for the random access file identified by UNIT.

Return Codes:

The ERR variable contains a return code:

Value	Description
0	Successfully set file pointer
1	Unspecified error. File pointer was not set

SETPOS8

Call:

```
CALL SETPOS8(UNIT,BYTE_ADDRESS)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
WORD_ADDRESS	INTEGER	IN	Number of bytes into file. Zero points to the first byte in a file.

Description:

SETPOS sets the byte address file pointer for the random access file identified by UNIT.

Return Codes:

None

SET_DUMPFILE_LENGTH (*DEF CRI_OPEN) (Cray systems only)

Call:

```
CALL SET_DUMPFILE_LENGTH(UNIT,LENGTH)
```

Arguments:

Variable	Type	Status	Description
UNIT	INTEGER	IN	File unit number
LENGTH	INTEGER	IN	Expected size of file

Description:

Used in conjunction with FILE_OPEN and the nupdate *DEF,CRI_OPEN, SET_DUMPFILE_LENGTH allows the expected size of the file about to be created to be set. FILE_OPEN then uses this size to preallocate space for the file.
This functionality is only available on Cray systems, the routine does nothing on non-Cray platforms.

Return Codes:

None

SHELL

Call:

```
CALL SHELL(COMMAND,COMMAND_LEN)
```

Arguments:

Variable	Type	Status	Description
COMMAND	CHARACTER	IN	Command to be executed
COMMAND_LEN	INTEGER	IN	Dimension of COMMAND

Description:

SHELL passes the command COMMAND to the operating system for execution.

Return Codes:

None

WORD_LENGTH

Call:

```
CALL WORD_LENGTH(LENGTH)
```

Arguments:

Variable	Type	Status	Description
LENGTH	INTEGER	OUT	Word length used

Description:

`WORD_LENGTH` returns the number of bytes used in a "standard" word. The number returned is obtained via the C function `sizeof`, using the C datatype "`real`". "`real`" should have the same wordlength as the Fortran type `REAL`. The C "`real`" is defined to be C type `double` if nupdate `*DEF,FRL8` is set, otherwise C type `float`. Section 4.2 of this document discusses this issue further.

Return Codes:

None