Unified Model Documentation Paper X1

PORTABILITY CONSIDERATIONS

by

A. Dickinson

Version number 3 dated 24th February 1995

For version 3.4 of the Unified Model

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

Modification Record		
Document Version	**Author**	**Description**
2	A. Dickinson	Information about data representation added
3	P. Burton	Added information on changing dump times

Contents

1. Introduction

The Unified Model system has been gradually evolved into a form which makes the code easy to port onto any Unix environment supporting Fortran 77+, ANSI C and the IEEE format for data representation. This has been achieved without impacting on the efficiency of the model on the main Cray computers used for operational numerical weather prediction and climate modelling. It is planned that the portable nature of the model will be maintained from now on.

This achievement now opens up a range of possibilities for the further use and development of the model:

i) As a research tool for use by outside organisations: It is planned that data and software will be made available to collaborating institutes on tape or through a Met Office computer node freely attached to national and international networks. New releases of the model will also be made available, once they have been accepted by internal users.

ii) The portable nature of the model will enable easy transfer of the code to other large-scale scientific computers. The current code structure can then be tested and developed for efficient running on massively parallel computer systems. This should allow the code to be gradually evolved into a more parallel form, making the transition to future Met Office computer systems reasonably transparent.

iii) Of particular interest is the ability to run the model on workstations. This may represent a cost effective way of developing and running low resolution versions of the model in the future.

The original design of the model recognised the need for portability. Most occurrences of Cray specific code were identified and alternative code provided in standard Fortran via software switches in the source code management system. Nevertheless substantial changes have been necessary:

i) Random access input-output is used by the model. This is a non-standard feature of Cray Fortran. Alternative C routines have been written to provide the same functionality. These are described in UM Documentation Paper S5.

ii) Source code management relies on a Cray specific utility called nupdate. When used on non-Cray systems, this has been replaced by an equivalent facility which has been developed in-house. This utility is described in Section 3.

1. Introduction (cont)

iii) Memory management within the model uses a Fortran 77 extension to automatically allocate and free work space. This feature is supported by Cray Fortran and by the new Fortran 90 standard, but is not yet widely available on other computer systems. A utility was therefore written to identify the use of dynamic memory within the model and automatically replace the code by a call to standard C memory allocation routines. This utility is described in Section 4.

iv) The model control scripts have been developed so as to make those sections containing Cray specific commands, such as the compilation step, more modular. Alternative, machine specific modules may now be easily substituted.

v) The User Interface allows scientists to easily design and setup new experiments. It corresponds to over 200 menu driven panels and runs on the Hitachi Data Systems front-end to the Cray YMP8s. The conversion of the User Interface into a fully portable form will require further substantial effort which is now under way. In the meantime, some simple guidance is given in Section 5 on how to make changes to the basic functionality of the model which relies on the editing of a text file.

2. Fortran 77 Extensions Used

Certain widely supported extensions to the Fortran 77 standard are included in the code. All of these features are supported by the Fortran 90 standard.

DO ... ENDDO
DO WHILE ..... ENDDO
More than 19 continuation lines
Variable names longer than 6 characters
Variable names containing _ or $
Use of lower case letters
NAMELIST processing
Use of IMPLICIT NONE
COMMON variables initialised outside block data
Character variables equivalenced to non-character variables
Use of automatic arrays
Use of ! to indicate embedded comments

3. Source Code Management

A portable version of the Cray utility *nupdate* is provided as part of the UM suite. It supports all the functionality and options used within the UM system to modify and update the program library. As with the Cray version of *nupdate* each DECK and COMDECK is stored as a separate member of a directory. This facilitates browsing of the program library in a workstation environment. The portable *nupdate* does not support the retention of historical information about previous changes.

A full description of the utility is given in UM Documentation Paper X2.

4. Dynamic Memory Management

Automatic arrays are used to manage workspace throughout the Unified Model. Although this aspect of dynamic memory management is available under Cray Fortran, it is not part of the Fortran 77 standard. It is, however, part of the Fortran 90 standard.

A simple example of the use of automatic arrays is as follows. Only the length of the array to be allocated is passed to the subroutine. The space occupied by the array is release on exit from the subroutine.

```
PROGRAM MAIN
        INTEGER N
        N=10
        ..
        CALL SUB(N)
        ..
        RETURN
END

SUBROUTINE SUB(N)
        REAL ARRAY(N)
        ..
        ..
        ..
        RETURN
END
```

A utility to provide the features of automatic arrays from within Fortran 77 is provided as part of the UM suite. The utility analyses the source code for occurrences of automatic arrays and allocates the necessary memory via an intermediate C language routine. The utility, known as *lexcon*, acts as a preprocessor and produces as output a modified version of the Fortran source code and a file of C routines which carry out the allocation of dynamic memory. Details of how to call *lexcon* are given in UM Documentation Paper X3.

4. Dynamic Memory Management (cont)

On application of *lexcon* to the above example, the following code is produced:

Modified Fortran source code:

```
PROGRAM MAIN
        INTEGER N
        N=10
        ..
        CALL SUB(N)
        ..
        RETURN
END

SUBROUTINE SUB_(N,ARRAY)
        REAL ARRAY(N)
        ..
        ..
        ..
        RETURN
END
```

New C routine:

```
SUB(N,ARRAY)
int *N;
int *ARRAY;
{
        allocate(ARRAY,N);
        SUB_(N,ARRAY);
        free(ARRAY);
}
```

4. Dynamic Memory Management (cont)

Support for the use of lexcon means that certain coding styles cannot be supported within the UM source code. Two constructs in particular should be avoided:

(i) The use of COMMON to pass integer values which are then used to dimension automatic arrays is beyond the scope of lexcon.

(ii) Use of CHARACTER string declarations of the form

```
CHARACTER PHRASE*(*)
```

cannot be handled by the Fortran-C-Fortran interface used by *lexcon*. This is because Fortran compilers can generate a hidden argument which is lost by the use of *lexcon*. This problem can be avoided by coding an explicit length for the dimension.

5. Substitute for the User Interface

Until the User Interface is available in a portable form, the default experiments generated within the Met Office computer system and provided for external use will have to suffice. Fundamental changes to these experiments, such as additions to the list of diagnostics or the number of model levels, will be difficult to implement without the aid of the User Interface. If required new experiment libraries can be obtained from the Met Office.

Some changes to the details of an experiment can be made quite easily by editing the control files which form part of the experiment directory. These files are located in the directory $HOME/jobs/<experiment name>

The most useful file is the master control file which takes a name of the form *mctlfl#x* (where x can be any letter), which contains configuration data set out in NAMELISTs. Some useful examples are:

(i) Changing the number of timesteps

Search for RUN_TARGET_END. This is an array of six numbers ordered as year, month, day, hour, minute, second. Edit these numbers to give the length of the forecast. For example, a one day forecast would be 0,0,1,0,0,0. The actual number of timesteps is determined by the value of A_STEPS_PER_HR. For example, a value of 2 means a 30 minute timestep. The value of A_STEPS_PER_HR should not be changed.

(ii) Changing the initial conditions

Search for MODEL_BASIS_TIME. This is an array of six numbers ordered as year, month, day, hour, minute, second. Edit these numbers to give the date of start data. For example, if the forecast is to be run from midnight on 22/9/93, then enter 1993, 9, 22, 0, 0, 0.

The name of the file containing the initial data will also need to be changed. If reconfiguration is switched on then change AINITIAL. If the forecast is being started directly from a dump then change ASTART.

(iii) Controlling the writing of dumps

Dumps can be written out at regular intervals by setting the A_ARCHDUMP_FREQ and A_DUMPFREQ variables in the MCTLFL file. As their names suggest they give the frequency (in timesteps) at which dumps are written. Alternatively, the array A_DUMPTIMES can be used, this allows dumps to be output at the end of specified timesteps.

6. Data Representation

The data representation used by Cray assumes a 64-bit word length and uses a unique, non-standard representation for floating point data. In order to provide a more widely supported format for the initial and ancillary data used by the model, alternative files may be constructed which use the IEEE standard for representing floating point, integer and logical data types. These may be created in either 32-bit or 64-bit precision. The utilities for converting from Cray representation to IEEE representation are described in Documentation Paper F5.

On some computer systems the order of bytes in each IEEE word may be reversed. The default format for IEEE data is assumed to be big endian. A utility called *bigend* is available which reverses the byte order to use the little endian data format. This is descibed in Documentation Paper S5. Repeated application of *bigend* toggles the file format between big endian and little endian.