The Met.Office

UNIFIED MODEL DOCUMENTATION PAPER NO X4

GUIDELINES FOR BUILDING THE UNIFIED MODEL ON EXTERNAL SYSTEMS

by

A. Van der Wal

Version 8

29 February 2000

Model version 4.5 (beta release)

Numerical Weather Prediction
The Met. Office
London Road
BRACKNELL
Berkshire
RG12 2SZ
United Kingdom

Modification Record		
Document Version	**Author**	**Description**
1	Tracey Smith	Original Version
2	Tracey Smith	i) Extra files umdata.tar, umpl3.2.tar, READ.ME and crayscripts.a added to initial archive (2.1) ii) Information about tape format (2.1.1) iii) List of machine specific scripts updated and information about lexcon and C_LOW added (3.1) iv) New interface to compilemodel3.2 explained (3.2) v) Reference made to bigend (5.) vi) Added sections 1, 3.2.1, 4.2, 6 and Appendix B
3	Tracey Smith	i) Removed all references to crayscripts.a ii) Changed READ.ME files to README iii) Removed section 4.2 iv) Re-wrote section 3 v) Updated diagrams in Appendix A vi) Explained new unpacking and compiling procedure
4	Tracey Smith & Nicola Farnon	i) Swapped sections 4 & 5 ii) Added sections 2.3.1, 4.1, 4.2, 5.2 & 7 iii) Re-wrote section 3.3 & subsections iv) Updated section 6 & Appendix A
5	Tracey Smith	i) Correct disk space requirement (1.2) ii) Added warning about incompatible byte order when reading QIC (2.2) iii) Amended diagram and text (2.3) iv) Included path of compilemodel (3) v) JOBID & RUNID clarified (4) vi) More information included (4.1) vii) Redundant tar extract deleted (4.2) viii) Include use of at (5) ix) Update instruction for zebra (6) x) Update known problems (7)
6	Mussadiq Ahmed	i) Major rewrite to take in account of changes to version 4.0
7	Anette Van der Wal	i) Major rewrite to include changes to version 4.4, especially new compile system.
8	Anette Van der Wal	i) Update for version 4.5, including adding brand name The Met. Office

Contents

1 Introduction

The Unified Model (UM) is one of the world’s leading numerical prediction models. When the original code was written, portability was not a big issue. It was more important to get fast efficient software for the platform of the time. Over the years, with advancements in computer technology, it has been necessary for The Met. Office to upgrade its supercomputer, which has meant changing the UM code. The decision to make the UM as portable as possible was taken both to enable a smoother transition between supercomputer upgrades within The Met. Office, and also to allow external use of the model. It was hoped that the latter would encourage development of scientific schemes which could be introduced into the UM to improve results.

Hence, the term "portable" is used in the sense of the UM having the *potential* to run on any Unix platform, rather than it actually doing so. Since there are only a limited number of platforms available for testing within The Met. Office, support is provided for these platforms only. This doesn’t mean that the UM can’t be built on other machines, but it is likely that some effort would be required by the user to do so.

It is intended that the UM should not be just a "black box" to its external users, and for this reason some knowledge of the UM system, and general Unix experience is essential before attempting to build the model. The code contains many thousands of lines, and requires large amounts of memory and disk space in order to run successfully.

The UM makes extensive use of the Unix "make" facility for compilation of its Fortran and C source code, and is controlled by top level Unix scripts, which take input from namelists provided by the UM User Interface (UMUI). The UMUI is an X windows application based on Tcl/Tk.

The UM can be compiled to use 32bit/IEEE numbers, or 64 bits if the platform supports this, and it is for the user to decide whether accuracy can be sacrificed for performance, and lower memory and disk usage. All platforms supported at this release have been tested with the UM running at 64 bits.

2 Resource Requirements

2.1 Preliminaries

Three experiments are supplied as standard. Table 1 gives some general details about them.

EXPT. ID (reference only)	**DIMENSIONS (columns x rows x levels)**	**RESOL-UTION (approx.)**	**AREA**	**FORMU-LATION**	**TIMESTEP**	**RUN-LENGTH**
a	96 x 73 x 19	280km	Global - climate	HadAM3	30 minute	40 days
b	92 x 92 x 31	17km	Regional - mesoscale	M20	5 minute	24 hours
c	98 x 73 x 20	280km (N-S) 417km (E-W)	Global - ocean	HadOM2	60 minute	10 days

Table 1

2.2 Memory

The amount of memory required by the UM depends on the choice of resolution and the size of the integration area. A further factor is the number of output diagnostics requested. Since, at each timestep, all of the model data is held in memory, the total requirement scales according to the number of grid points in the horizontal, and the number of levels. The addition of a new single-level diagnostic, such as accumulated precipitation, increases the memory requirement by the size of one horizontal field.

Some examples of typical memory usage for the atmosphere and ocean components of the UM are as shown in Table 2. This is for The Met. Office T3E where each PE is 16 Mwords (1Mword = 8 Mbytes).

EXPT. ID	**JOB**	**MEMORY USAGE (assuming 64-bit word)***
xaaa	c (MPP T3E)	83.3 Mwords (12 PE's)
xaab	c (MPP T3E)	78.5 Mwords (18 PE's)
xaac	c (MPP T3E)	78.0 Mwords (16 PE's)

* Maximum total memory usage over PEs

Table 2

On workstation systems it is advisable to ensure that enough physical memory is available to avoid the cost of data swaps by the virtual memory manager.

2.3 Disk Space

Table 3 shows the amount of disk space required to hold the UM, before it is unpacked.

Sub-package	Disk space (Mbytes)
data32	301
gcom	0.37
UM system	556
UM documentation system	150
UM graphical tools	28
UMUI	7

Table 3

Approximately 100 Mbytes of extra space is required after the UM system is unpacked and built. In addition, you should allow another 60 to 100 Mbytes for the compilation files, executable and reconfigured start data for each experiment you run. Diagnostic output files contain copies of each requested field at each time. The size of these files will therefore vary greatly depending on resolution, number of diagnostics, write frequency and length of integration. For the supplied mesoscale experiment, output files use about 125 Mbytes of disk space.

2.4 Timings

The CPU time needed for a forecast is proportional to the period of the forecast, i.e. the number of timesteps, and the total number of grid points in the model, i.e. rows x columns x levels. Increasing the horizontal resolution of the model requires a corresponding change in the number of timesteps per forecast hour, in order to satisfy the criterion for numerical stability. Hence, halving the horizontal grid size will demand 8 times the CPU cost for a forecast of the same period covering the same domain.

Examples of model CPU timings for the supplied experiments are as shown in Table 4. These were obtained from running on The Met. Office T3E (a T3E-900). All timings include compilation.

EXPT. ID	CPU TIME*	
	MPP	Non-MPP
xaaa	(#c) 137640 secs	(#a) 40588 secs
xaab	(#c) 1485 secs	(#a) 1030 secs
xaac	(#c) 42385 secs	(#a) 11597 secs

* Maximum memory usage over PEs

Table 4

Note that subsequent compilations require less time than the initial one.

The total run time also depends on the number of output requests, on I/O overheads and other traffic on the system. Times for other resolutions and domain sizes can be estimated approximately from the scaling described above, but costs also depend on factors such as the increased speed of vector processing with increased vector lengths, cache size, physical memory available, and number of PE’s if running a massively parallel version of the UM.

3 Building the Unified Model - A Summary

The process of building the UM on a platform external to The Met. Office is performed in 5 stages:

- Retrieving the files from tape or CD
- Unpacking the UM onto the computer using unpackmodel
- Setting the UM environmental variables by running setvars
- Compiling the small executables using configure_execs
- Compiling the model sections using configure_all_sects

The build process has been designed to be performed in several steps; these have purposely not been combined to be carried out by a single command. This is because it is more difficult to recover from an error in this situation. Also, it is felt that the UM should not just be a "black box" to it's users, and understanding of the build process is beneficial at a later stage when performing model runs.

Chapters 4 and 6 give a detailed description of how to unpack and build the UM. It is strongly advised that a new userid be set up for the UM, and that it is built using this userid. This ensures that accidental over-writing doesn't occur. Also, for example, by issuing the "touch" command, a user can update the date/time information of a file they own, even if they don't have write permission for that file. This can be very dangerous when using make.

Separate, but essential to the running of the UM is the UM User Interface (UMUI). This is an X windows application built using Tcl/Tk. Details of how to configure the UMUI can be found in the README files which are included with the installation.

For your information:

The UM version is 4.5 (VN=4.5)
The portable nupdate version is 0.4.9 (UPDATEVN=0.4.9)
The GHUI version is 2.0a3
The UMUI version is 2.0

The directory UM_HOME will generally be the HOME directory of the UM userid; the directory UMDIR will be $UM_HOME/um.

4 Unpacking the Unified Model

4.1 Retrieving Files from Tape/CD

If you haven't done so already, you should retrieve the file um_system.tar from the tape or CD. See UMDP X0, chapter 2 for instructions on how to do this. The file needs to be placed in the HOME directory of the userid which is being used to hold the UM. When um_system.tar has been untarred, the directory structure is as shown in Appendix A.1.

4.2 Determining Platform Specific Interfaces, Wordlengths & Data Representation

The UM is written in Fortran, but uses C routines to perform I/O. It is crucial that the correct Fortran/C interface is chosen, as it differs between platforms and compilers. The C routine names can take 3 forms:

- Upper case names
- Lower case names
- Lower case names with a trailing underscore

The UM passes Fortran variables to C routines whilst performing I/O, so it is essential for the C code to use variables with the same wordlength as used in the Fortran. It is assumed that Fortran INTEGER, REAL and LOGICAL variables all have identical wordlengths. Therefore, it is necessary to determine which C variable type (float or double) has the same wordlength as a Fortran REAL. If a C double has equivalent wordlength to a Fortran REAL, then generally, a C long will have equivalent wordlength to a Fortran INTEGER. An exception to this is the Linux PC, where a C long long is required.

The byte order of the data is also different between machines. It indicates whether the bytes are stored most significant to least significant (big-endian) or vice versa (little-endian). The concept is analogous to storing the digits in 100 as 1-0-0 or 0-0-1. The supplied data files are all big-endian so on platforms with little-endian data representation, the bytes have to be swapped. This is done using a simple C program called bigend.

All of the above can be determined by using the utility fc_test, which can be found in the directory $UM_HOME/um/vn4.5/utils/fc_test-1.1. It is compiled and run automatically when the UM is unpacked.

4.3 Choosing Between Cray and "Portable" Nupdate

4.3.1 General Information

The UM uses a source code control system called nupdate. The version used at The Met. Office is by Cray; it may already be installed on your system. Versions are available for Cray, Sun and SGI platforms and can be downloaded by anonymous ftp from ftp.cray.com in directory src/nmodex-nupdate. A portable version of the software has been written at The Met. Office. Portable nupdate has less functionality than the Cray original; therefore it is advisable to use the Cray version if possible.

The portable version of nupdate can be found in $UMDIR/vn4.5/utils/update0.4.9. There are 5 makefiles, each tailored for a different platform, with the suffixes cr (cray), da (DecAlpha), hp (HP), sun (Sun) and other (OTHER). There is also a makefile.general which uses the compiler options provided to build the UM, and which is used to automatically build the nupdate executable when the UM is unpacked.

A full description of portable nupdate is given in UMDP X3.

4.3.2 Changes to the UM Program and Script Libraries

In unpackmodel, you are required to specify which version of nupdate will be used – Cray or Portable. The choice here will determine which program and script libraries are used, since Cray nupdate requires them to be in a single file, in a format which is not browsable, and Portable nupdate expects the libraries to be in directory format; the files in here are browsable. Both sets of libraries are provided with the portable release. The file-format (Cray nupdate) libraries are called umpl.file and umsl.file; the directory-format (portable nupdate) ones are called umpl.dir and umsl.dir. On running unpackmodel, the required ones are renamed umpl and umsl.

If for any reason, a problem occurs during the renaming process, the user should manually do the renaming, before running unpackmodel again.

Since umpl and umsl are large, you may prefer to delete the set which you won't be using, to save some disk space.

4.4 Setting System Parameters

Some system parameters may need to be altered in order for the UM to run. The most common parameters that need to be increased are those that control the maximum number of arguments allowed to a UNIX command, and the maximum length of a UNIX command line. You may need to ask for assistance from your system administrator in order to change these. For example, these are changes we needed to make to an SGI Octane/Origin system:

The parameter ncargs, which determines the length of the argument lists allowed, needs to be set to 40960. To do this, you need to be root. At the prompt type

systune -i

and in systune type

ncargs 40960

It is often necessary to increase the stacksize; the method varies between platforms and it may be necessary for your system administrator to help you with this. It can involve a rebuild of the kernel. On some platforms, this can be done using the command *ulimit –s*. The system limits in Table 5 are taken from the supported platforms at The Met. Office, using *ulimit -a*:

	Dec Alpha	**SGI Octane**	**SGI Origin**	**Linux PC**
time(seconds)	unlimited	unlimited	unlimited	unlimited
file(blocks)	unlimited	unlimited	unlimited	unlimited
data(kbytes)	131072	2097152	unlimited	unlimited
stack(kbytes)	131072	65536	65536	8192
memory(kbytes)	251408	120464	524288	unlimited
coredump(blocks)	unlimited	unlimited	unlimited	unlimited
nofiles(descriptors)	4096	200	200	1024
vmemory(kbytes)	1048576	2097152	unlimited	unlimited
concurrency(threads)	N/A	N/A	1024	256

Table 5

4.5 Unpacking the Files

The UM Unix script unpackmodel can be found in the directory:

$UM_HOME/um/vn4.5/scripts/Install

along with a file, mach_defaults, containing default answers to the questions in unpackmodel.

Unpackmodel is used to:

- extract files from umsl (UM script library) and place them in the directory, and subdirectories of $UM_HOME/um/vn4.5/scripts
- create scripts for the small executable (utility) programs which are placed in $UM_HOME/um/vn4.5/utils
- create cross reference files for the small executables (exec_xref) and model sections (obj_xref) which are placed in $UM_HOME/um/vn4.5/source
- report the progress for unpacking the UM in $HOME/unpack_4.5.log
- create a file of customized answers in $UM_HOME/config_4.5.cache, which can be used in subsequent runs of unpackmodel to provide default answers.

For more information on running unpackmodel, see Appendix B.

4.6 Setting Environment Variables

4.6.1 General

Unpackmodel creates a file called $UM_HOME/setvars_4.5 which is linked to $UM_HOME/setvars and contains all the essential UM environment variables. These variables must be set up before attempting to create the small executables or compile the model sections. When running model experiments, the UM must have access to these variables, and it expects them to be in a file called $HOME/setvars. Therefore, $UM_HOME/setvars should be linked or copied to $HOME/setvars for each UM user. A line should be added to the login file so that $HOME/setvars is executed for all login sessions. Note that if unpackmodel is re-run, for instance to change the compiler flags, the user should re-source $UM_HOME/setvars either directly or by re-running the login file. Failure to do this will result in the changes not being picked up when compiling the UM.

The following sub-sections explain what to do if you use the Bourne, Korn or POSIX shell, or the C shell. The UM uses the Korn shell.

4.6.2 The Bourne, Korn & POSIX Shells

If you already use the Bourne, Korn or POSIX shell, you will need to change your .profile file by inserting the following line:

```
. /<full pathname>/setvars
```

Note that there is a space between the "." and the "/"; it is important that this space is not omitted.

To pick up the setvars variables for your entire session, it is advisable to log off and then log back on again. For the current window, you can type:

```
cd $HOME
. ./.profile
```

to pick up the changes in this window only.

4.6.3 C Shells

If you normally use the C shell you will need to change shell by typing the command:

```
sh
```

You should then execute the script that sets up your UM environment variables by typing:

```
. /<full pathname>/setvars
```

Note that there is a space between the "." and the "/"; it is important that this space is not omitted.

5 GCOM

5.1 General Details

GCOM is only required if you are running the UM in MPP mode. It is a library of Fortran routines, which interface to a message passing library of your choice. This allows the UM to be used with many different message passing libraries without having to change or recompile any of the UM code itself (just relink with a different version of the GCOM library).

The GCOM software should be unpacked into the directory $UMDIR/gcom. If you haven't done so already, retrieve the file gcom.tar from the tape/CD and place it in the directory $UMDIR. You should then untar this file by following the instructions in UMDP X0, chapter 2. The readme file gives a general useful background about the GCOM directory structure and how to build GCOM.

To build GCOM, you will need to go to the rel_1m1s5x4/build directory, and edit the Makefile to suit the particular message passing library and machine you wish to use. At the moment it is set up for SHMEM_NAM on the Cray T3E. You may find it useful to look at the header blocks of gc_com.F and gcg_com.F for further information on the cpp preprocessor flags used by the GCOM libraries.

Type

make

in this directory to create the gcom library (the exact name of which will depend on the configuration you have chosen). If you are happy with this library, copy it up a level to the gcom/rel_1m1s5x4 directory.

To ensure this library gets linked by your UM compiles, a hand-edit may be required.

On a T3E, for example, the file $UMDIR/vn4.5/source/compile_vars contains the following lines:

```
@load  LOAD_LIBS=-lgcom1m1s5x4_shmemnam
@load  LOAD_OPTS=-Wl"-Dpermok=yes;streams=on"
@load  LOAD_PATH=-L. -L$(UMDIR)/gcom/rel_1m1s5x4
```

The name of the library in the LOAD_LIBS line may be wrong if you have not chosen the SHMEM_NAM option, in which case change the name to the name of the library you have created. Any value in the compile_vars file can be changed for a model run using a compile override file, but it is nicer not to have to do this, so a hand-edit now is preferable.

5.2 SGI Specific Requirements

Some extra code was required to make GCOM (MPI flavour) work correctly on SGI MPP platforms (Origin) in 64 bit mode. The 64-bit MPI implementation required some arguments to be 32 bits. This problem was solved by adding some wrapper routines around the MPI, which can be found in extras/sgi_64_mpi; they convert the relevant arguments to 32 bits. In order to compile GCOM correctly for MPI on SGI platforms, the makefile Makefile.sgi64 should be used instead of the standard Makefile.

6 Compiling the Unified Model

6.1 An Overview

In earlier versions of the UM, all compilation was performed by executing one script. Although potentially simple to operate, this meant that the whole compilation system was a "black box" to the user, and after a failure, the user was often left unaware of what had failed, and how it should be corrected. At this release it is intended that the user should play a more interactive role in the compilation.

The following stages should be completed successfully before moving on to running the UM:

- Compile GCOM (MPP machines only), see chapter 5
- Create the small executables
- Create the pre-compiled model sections

6.2 Creating the Small Executables

6.2.1 The exec_xref File

The term "small executables" is used to describe any of the UM utilities, such as pumf and cumf, the reconfiguration program and the UM history executables. Each of the small executables to be built on your platform is listed in the file $UMDIR/vn4.5/source/exec_xref. A full list of those available can be found in $UMDIR/vn4.5/source/umsl/exec_xref_base. Note that some (such as convieee) require the use of Cray specific routines, whilst others (such as camdump) are specific to The Met. Office.

The exec_xref file is a useful point of reference as it gives a short description of the purpose of each small executable, along with the DEFS required to build it and a list of the Fortran and C decks which are used.

6.2.2 Submission Method

Before building the small executables, the user must decide what their submission method is. For platforms external to The Met. Office, all compilation is done interactively. If your platform supports batch submission then a hand-edit is required. To do this, edit the file $UMDIR/vn4.5/scripts/Install/configure_execs and near the bottom change the line

```
  # Default submission method...hand-edit if necessary
  $OUTDIR/$exec.script
```

to your preferred submission method. On The Met. Office T3E, the command

qsub $OUTDIR/$exec.script
is used.

6.2.3 Adding in Modifications

The UM program library (umpl) is the vn4.5 umpl used at The Met. Office, plus extra modifications (mods) to allow compilation to complete successfully on a non-T3E platform. If there are any platform specific mods which haven't been built in to umpl these will need to be passed to the nupdate step when the code is being extracted to build the small executables. This might happen if you are building on platform type "OTHER", for example.

To do this, edit the file $UMDIR/vn4.5/scripts/Install/configure_execs and specifically the call to qsconf. You can get help for qsconf by typing:

```
$UMDIR/vn4.5/scripts/qsconf -help
```

Add in Fortran mods using the *-femod* option, or C mods using the *-cemod* option. Different modsets should be comma-separated and not contain any spaces. Alternatively, all mods can be concatenated into one file.

So, for example, if there were 3 Fortran modsets to add, then the call to qsconf would look something like this:

```
$UMDIR/vn$VN/scripts/qsconf \\
  -femod $HOME/mods/mymod1,$HOME/mods/mymod2,\\
$HOME/mods/test/mymod3 \\
  -outdir ‘dirname $SRCDIR‘ \\
  -execs $exec 1> $OUTDIR/$exec.out 2>&1
```

You should edit the call to qsconf which is already in the code; do not add the above code as an extra call to qsconf. Note that the continuation line for the *-femod* option starts at the very beginning of the line, and there is no space between the "," and the "\\" on line 2, but there is a space before the "\\" on line 3.

You may like to include an environment variable which holds the path of the directory which contains your mods. In this case, you must protect the "$" to stop it from being interpreted by the shell, as follows:

```
MODDDIR=$HOME/mods
$UMDIR/vn$VN/scripts/qsconf \\
  -femod \$MODDIR/mymod1,\$MODDIR/mymod2,\\
\$MODDIR/test/mymod3 \\
  -outdir ‘dirname $SRCDIR‘ \\
  -execs $exec 1> $OUTDIR/$exec.out 2>&1
```

All mods can be found in the sub-directories of $UMDIR/PUM_Input/vn4.5/mods. Under the source directory, mods in clim, mes and ocean, are science modifications and are applied during a model run. Those in debug can be used for debugging purposes, during a model run, and those in general are not part of the standard supplied experiments, but may be required on some platforms. The directory builtin contains all mods which have already been included in umpl.

6.2.4 Running configure_execs

To build the small executables, type the following:

cd $UMDIR/vn4.5/scripts/Install
configure_execs

This creates the following directories:

- $UMDIR/vn4.5/build_execs_date_time
  This is a uniquely named directory, and each time configure_execs is run, a new directory is created. It contains the scripts which get submitted to create each small executable (e.g. cumf.script), the output from the script (e.g. cumf.out) and a an empty file which acts as a flag to immediately ascertain whether the build has worked (e.g. cumf.success) or failed (e.g. cumf.error).

- $UMDIR/vn4.5/exec_build
  This contains a sub-directory for each small executable which holds all the extracted Fortran and C source files plus a Makefile, and all the compiled object code for the executable. If you encounter an error in the build of one of the small executables, it is useful to check the source in here, and amend it until a successful build is achieved. It is then wise to add a modification via the call to qsconf as described in 6.2.3. Remember that changes to some of the decks will be required throughout the model. For instance, a change to PORTIO2A, the portable I/O C routines, will also be required when building the model sections.

- $UMDIR/vn4.5/exec
  When the small executables have been successfully made, they are automatically copied to this directory.

The script loops through all of the executables in the exec_xref file, but if the user wants to re-run configure_execs for a single executable, then the lines

```
for exec in ‘cat $EXECXREF| awk ’{print $1}’|\
            egrep -v ’#’ | egrep -v ’MACHINE’ | sort -u | flower‘
```

can be changed to
for exec in qxrecon_dump

for example, to build the reconfiguration program qxrecon_dump only.

6.2.5 Using the Small Executable Utilities

For ease of use of the utilities such as pumf and cumf, it may be desirable to add the directory $UMDIR/vn$VN/utils to your PATH, or link the utilities directly to $UMDIR/bin. This isn’t performed automatically as the UM is built due to possible clashes with different versions of the UM. This could be particularly dangerous in a multi-user environment.

6.3 Building the Model Sections

6.3.1 The obj_xref File

The model sections are pre-built for all standard or frequently used configurations of the UM. This saves time at each run of the UM. The sections to be pre-built on your platform are listed in $UMDIR/vn4.5/source/obj_xref. Only those sections and versions which contain the word "BUILD" will be built. So, for instance, in section A01, version 1B will be built but version 1A won’t. Some section versions have 2 or more "BUILD" lines which indicate that the version is to be built with different combinations of *DEFS. For instance, A10_1A can be built with no DEFS (G0_n) or DEF GLOBAL (G3_n).

The obj_xref file also gives a title for each section and shows which decks are required to build the section.

6.3.2 Submission Method

Before building the sections, the user must decide what their submission method is. For platforms external to The Met. Office, all compilation is done interactively. If your platform supports batch submission then a hand-edit is required. To do this, edit the file $UMDIR/vn4.5/scripts/Install/configure_all_sects and near the bottom change the line

```
  # Default submission method...hand-edit if necessary
  $OUTDIR/$sect.script
```

to your preferred submission method. On The Met. Office T3E, the command

qsub $OUTDIR/$sect.script

is used.

6.3.3 Adding in Modifications

This is very similar to the procedure for including mods in the small executables (section 6.2.3) except that the script is called:

$UMDIR/vn4.5/scripts/Install/configure_all_sects

and the options to qsconf are:

-fsmod for Fortran mods
and
-csmod for C mods.

6.3.4 Running configure_all_sects

To build the model sections, type the following:

cd $UMDIR/vn4.5/scripts/Install
configure_all_sects

This creates the following directories:

- $UMDIR/vn4.5/build_sects_date_time
  This is a uniquely named directory, so each time configure_all_sects is run, a new directory is created. It contains the scripts which get submitted to create each model section (e.g. a01_1b.script), the output from the script (e.g. a01_1b.out) and a an empty file which acts as a flag to immediately ascertain whether the build has worked (e.g. a01_1b.success), failed (e.g. a01_1b.error) or that this section has no combinations (e.g. a01_1b.nocomb).

- $UMDIR/vn4.5/src
  Under this directory is a sub-directory for each model section and version and these contain further sub-directories for each build combination (e.g. G0 or G1G3Gn). These hold Fortran and C source code files and a Makefile. There is an additional sub-directory called NORMAL and it is in here that the object files are placed.

- $UMDIR/vn4.5/obj
  Using a similar directory structure to the src directory, the object files from the "NORMAL" sub-directories of "src" are linked to here.

The script loops through all of the section versions in the obj_xref file, but if the user wants to re-run configure_all_sects for a single section version, then the lines

```
for sect in ‘cat $OBJXREF|\
 egrep -v \
 ’^COMP_GEN_OPTS|^DEF_SWITCHES|^COMPILE|^MACHINE|^SECTION|^#|^$’|\
 awk ’{print $1}’|\
 sort|uniq|flower‘
```

can be changed to

for sect in a01_1b

for example, to build version 1b of section a01 only.

6.4 The Bigend Utility

Section 3.3 described the procedure for determining some platform specific details, one of which was the byte ordering or "endian" of the machine. The UM data is in BigEndian format, so if you are using a LittleEndian platform (such as the DecAlpha) the data requires conversion, where the bytes are swapped. A utility called bigend is provided to do this.

To build bigend, issue the following commands:

cd $UMDIR/vn4.5/utils/bigend
compile

To see how to use this utility, type

bigend

The output from this command is:

```
bigend: usage: -32|-64 file1 file2
```

For 64-bit data you will need the flag -64 and 32-bit data requires the -32 flag; file1 is the input filename and file2 is the output filename.

Those datasets which require conversion are all the ones in the sub-directories of:

- $UMDIR/PUM_Input/dumps
- $UMDIR/PUM_Input/lbcs
- $UMDIR/vn4.5/ancil

6.5 Preliminary Tests

Before going any further it is useful to test whether the small executables and other utilities are working correctly. It is also useful to ensure that the start dump is readable. For details on usage and functionality of the utilities, see UMDP F5.

- **pumf** - **P**rints out the contents of a **UM F**ile
  By testing this utility out on a UM dump, the user can ensure that pumf is working and the UM dump is readable. It is very easy to forget to run bigend on some data files, so use pumf to check that bigend has worked successfully.

- **cumf** - **C**ompares **UM F**iles
  Check that cumf is working correctly by comparing a file with itself. You should see no differences.

- **convpp** - **Conv**erts UM files into **PP** format
  and
  pptoanc - Converts **PP** files **to anc**illary file format
  Take one of the UM ancillary files (from sub-directories of $UMDIR/vn4.5/ancil) and apply convpp. Then run the newly created pp file through pptoanc to convert back to ancillary format. See the documentation on pptoanc for instructions on how to do this. Use cumf to check that you have ended up with what you started with.

7 The UM User Interface

7.1 General Details

The UMUI is a self-contained package which is external to the UM. It consists of three parts:

- Tcl/Tk (Tool Command Language/Tool Kit)
- GHUI (Generic Hierarchical User Interface)
- UMUI (Unified Model User Interface)

If you haven't done so already, retrieve the files umui_package.tar and umui_templates.tar from the tape/CD. These can be placed anywhere in your directory structure, but preferably not under $UMDIR as the UMUI is not part of the UM system - $UM_HOME is a good place. Untar them following the instructions in UMDP X0, chapter 2.

The first step is to install Tcl/Tk, version 8.0, patch 2. This is supplied with the UMUI package, if it is not already available on your system. Tcl/Tk is 3rd party software which is freely available off the Internet; it is not supported by The Met. Office.
The README file included with the package gives full installation instructions.

Secondly, install the GHUI. This is a user interface design package which can be used to create user interfaces. Again, see the README file included in the GHUI installation for instructions. The version number of the job entry code doesn't match that of the UM, so during installation, when asked for the version, you should enter the default value (which is 1.1 in this release).

Finally, configure the UMUI. This stage doesn't require any compilation; the README file gives details.

Table 6 shows the compiler options used on various platforms.

	PLATFORM			
QUESTION	**SGI**	**DecAlpha**	**HP**	**Linux PC**
C compiler	cc	cc	cc	gcc
C compiler options	-n32 -ansi -O2	-std1 -O -shared	-Aa +O2 +z	-ansi -O2 -fpic -shared
Link editor	ld	cc	ld	ld
Link editor options	-shared -n32	-std1 -O -shared	-b	-shared
Remote shell command	rsh	rsh	remsh	rsh

Table 6

If you are running the UM on a supercomputer (T3E for example) it is common for the UMUI to be installed on a workstation, with the jobs being submitted remotely. In this case, it is necessary to have a .rhosts file on the remote machine, to allow the remote copy to work. The file should reside in your $HOME directory, and contain lines of the format:

hostname username

where hostname and username are those of the local machine. See "man rhosts" for more details.

The UMUI package makes use of shared libraries, a facility not available on the T3E. Therefore, if your requirement is to run the UMUI on a T3E, you will need to contact Portable UM Support, who may be able to supply a new version of the UMUI code.

The file umui_templates.tar contains template files for changing the appearance and personal preferences of the umui, and also a script called copy_stash which is used to copy stash from one job to another. When untarred, the files will be in a directory called umui_templates; it is not important where this resides as it not part of the UM or UMUI standard software.

7.2 UMUI Administration

Many users can utilize the same UMUI installation, but there must be someone in charge of administration. This will usually be the person who installed the application. It is the job of the administrator to start the umui server(s) initially, and also after a system crash or shutdown. This is done by executing the umui_admin script from the bin directory of the umui installation. Most installations of the umui will be "single-server" mode.

The following steps are required to start the server:

- Click on the button "Start Primary Server"
- Go to the menu bar and click on "Window" and then "Re-Draw"
  This will bring up a server log with 5 buttons underneath.
- Click on the button "Read Database"
  As you do this, the "Status" button will change from "(Empty)" to "(Paused)".
- Click on the button "Status"
  As you do this, the "Status" button will change from "(Paused)" to "(Active)".
- Go to the menu bar and click on "File" and then "Quit"

For more information about administrating the UMUI, read the help pages within umui_admin.

7.3 Starting the UMUI

Once the server has been started, the UMUI is activated from the bin directory of the umui installation by typing *umui*. It may be worthwhile linking the umui executable to your standard bin directory or adding the umui bin directory to your PATH variable in order for the umui to be started from anywhere.
For help on uploading and editing jobs see UMDP X0 or the UM User Guide, chapter 4, section 1.

7.4 UMUI Appearance

It is possible to change the appearance of the UMUI, which may be required for aesthetic purposes or technical reasons. On some low resolution monitors, the size of some of the windows is too large to fit on the screen and portions of the window can get omitted. Always check that the window is completed by ensuring that a row of buttons is visible along the bottom of it.

To change the appearance, you require a file called

$HOME/.umui_appearance

A template can be found in the umui_templates directory described in section 7.1. Change the values according to your own preference.

7.5 Personal Preferences

Users can have a file that specifies personal preferences. These will be set whenever a job is copied from another user or whenever a new job is created. This file can be used to specify items such as email addresses, TIC codes and queue preferences.

A file should be created called:

$HOME/.umui4.5rc

The format is:

Column 1: umui variable name
Column 2: new value

so, for example to ensure that the userid is always hadsm, the file will contain the following:

```
# Any comments start with a hash
USERID    hadsm
```

A template can be found in the umui_templates directory described in section 7.1. If, when copying a job, the template file fails due to an error in the file, then the job will be copied without any of the
preferences applied.

7.6 Copying STASH

A script which is external to the umui has been written that will copy sections from one downloaded job to another. The altered job can then be uploaded into a new UMUI job. The script is called copy_stash. Download the two jobs and run the script; a help option is available. The script can be used for copying any parts of jobs but since it is only likely to be used for copying STASH settings, explicit help has been provided only for this.

The script can be found in the umui_templates directory described in section 7.1.

Appendices

A : The UM Directory Structure

A.1. Structure of $UMDIR Before Unpacking the UM

```
um
 |
 └── vn4.5
        |
        ├── ancil
        |      |
        |      ├── atmos
        |      |      |
        |      |      ├ cl_9673
        |      |      |      |
        |      |      |      ├ AMIPII
        |      |      |      |
        |      |      |      └ hadcm3
        |      |      |
        |      |      └ ms_9292_uk
        |      |
        |      ├── ocean
        |      |      |
        |      |      └ cl_9673
        |      |
        |      └── slab
        |             |
        |             └ cl_9673
        |
        ├── ancil32 (as ancil but at 32-bits)
        |
        ├── ctldata
        |      |
        |      ├ STASHmaster
        |      |
        |      └ stasets
        |
        ├── scripts
        |      |
        |      └ Install
        |
        ├── source
        |      |
        |      ├ umpl.dir
        |      |
        |      └ umsl.dir
        |
        └── utils
               |
               ├ bigend-1.1
               |
               ├ fc_test-1.1
               |
├ figlet-2.2
│          │
│          └ fonts
│
└ update0.4.9
```

A.2. The Directory Structure of PUM_Input - Input for the PUM

```
um
 |
 └ PUM_Input
          |
          └ vn4.5
                   |
                   ├ ancil
                   |         |
                   |         ├ AMIPII
                   |         |
                   |         └ HadCM3
                   |
                   ├ ancil32 (as ancil but at 32-bits)
                   |
                   ├ dumps
                   |
                   ├ dumps32 (as dumps but at 32-bits)
                   |
                   ├ lbcs
                   |
                   ├ lbcs32 (as lbcs but at 32-bits)
                   |
                   ├── mods
                   |         |
                   |         ├── builtin
                   |         |         |
                   |         |         ├ scripts
                   |         |         |
                   |         |         └ source
                   |         |
                   |         ├── scripts
                   |         |         |
                   |         |         └ clim
                   |         |
                   |         └── source
                   |                   |
                   |                   ├── clim
                   |                   |         |
                   |                   |         ├── vn4.3
                   |                   |         |         |
                   |                   |         |         └── hadam3
                   |                   |         |                   |
                   |                   |         |                   └ AMIPII
                   |                   |         |
                   |                   |         ├── vn4.4
                   |                   |         |         |
                   |                   |         |         └── hadam3
                   |                   |         |                   |
                   |                   |         |                   └ AMIPII
                   |                   |         |
                   |                   |         └── vn4.5
                   |                   |                   |
                   |                   |                   ├── hadam3
                   |                   |                   |         |
│                   │                   │         └ AMIPII
│                   │                   │
│                   │                   ├── hadcm3
│                   │                   │
│                   │                   └── hadom3
│                   │
│                   ├── debug
│                   │
│                   ├── general
│                   │
│                   ├── mes
│                   │
│                   └── ocean
│                             │
│                             └── vn4.4
│                                       │
│                                       └── hadcm2
│
└ scripts
```

A.3. Brief Explanation of the Use and Contents of the UM Directories

- um/bin
  Contains executables such as nupdate and bigend (if required on your platform). The directory name is added to $PATH, so executables linked in to this directory will automatically be found.

- um/vn4.5/ancil
  Standard ancillary files for atmosphere, ocean and coupled models.

- um/vn4.5/ancil32
  As above but containing 32-bit data

- um/vn4.5/build_execs_ddmmyy_hhmm
  Contains the scripts for, and output from, building the small executables. There may be one or more of these directories depending on whether the first build was successful or not.

- um/vn4.5/build_sects_ddmmyy_hhmm
  Contains the scripts for, and output from, building the model sections. There may be one or more of these directories depending on whether the first build was successful or not.

- um/vn4.5/ctldata
  Contains the spectral namelists for longwave and shortwave radiation, along with the STASHmaster files and other STASH-related files.

- um/vn4.5/exec
  Contains the executable file for each UM small executable.

- um/vn4.5/exec_build
  Contains the build directories for each of the small executables. Each sub-directory holds the Fortran and C source files and the Makefile required for each small executable. The resulting object files are also present.

- um/vn4.5/obj
  Contains links to the object code of each of the model sections and versions.

- um/vn4.5/scripts
  Contains the “qs” type scripts which are using when running the model (they call the small executables).

- um/vn4.5/scripts/Install
  Contains scripts for installing and building the UM.

- um/vn4.5/scripts/functions
  Contains UM Unix functions which are called from within the UM scripts.

- um/vn4.5/source
  Contains a number of important reference files, the description of which are as follows:

• cdecks	list of control decks for the model
• comp_specific	compile options for specific decks
• compile_vars	compiler and loader names and options
• exec_xref	final cross reference file for the small executables
• exec_xref_base	initial cross reference file for the small executables
• makefileu_comp_in	generic makefile for the compile step of the model
• makefileu_link_in	generic makefile for the link step of the model
• obj_xref	final cross reference file for the model sections
• obj_xref_base	initial cross reference file for the model sections
• umpl.file	program library in file format: use with Cray nupdate
• umsl.file	script library in file format: Use with Cray nupdate

- um/vn4.5/source/umpl.dir
  The UM program library in directory (browsable) format containing the source decks and comdecks: use with portable nupdate.

- um/vn4.5/source/umsl.dir
  The UM script library in directory (browsable) format containing the script decks: use with portable nupdate.

- um/vn4.5/src
  Contains the extracted source code for each of the model sections and versions within its own sub-directory.

- um/vn4.5/utils
  Contains the scripts to run the utility programs such as pumf and cumf.

- um/vn4.5/utils/bigend-1.1
  Contains source for the bigend utility which is used to byte-swap data.

- um/vn4.5/utils/ fc_test-1.1
  Contains source code to build the fc_test utility which is used in unpackmodel to determine the Fortran/C interface, word-length and byte ordering of the platform.

- um/vn4.5/utils/figlet-2.2
  Contains freeware which converts normal characters into fancy fonts. It is used in place of the “banner” utility on some platforms.

- um/vn4.5/utils/update0.4.9
  Contains the source code for portable nupdate, the UM source code management utility.

B : Information on Running Unpackmodel

B.1. General Synopsis

This version of *unpackmodel* provides a default answer to every question, which appears in square brackets after the question. To select the default, the user should hit the <return> key. Each platform has its own set of default answers; these have been set and tested on platforms within The Met. Office. External users may find that their platforms differ slightly in some way from those on which the UM was tested, for example, have a different operating system version, or different compilers. On these occasions, the user may wish to supply an answer different to the default. For those questions where it is difficult to provide a default answer, for instance asking for a specific directory path, the default has been left blank, and the user should provide their own answer if applicable.

The default answers are provided in the read-only file *mach_defaults*. If your platform is not one of the following: Cray T3E, Cray PVP, HP, DecAlpha, SGI Origin or Linux x86, then you will be building the UM using the machine type “OTHER”. Because this can be used for a variety of machines, few of the questions have default answers. If you manage to successfully build and run the UM under “OTHER” for a non-supported platform, we will be happy to take your default answers and add them to *mach_defaults* for use by others. Extra care is needed when building MPP code on platforms where only the non-MPP version has been tested (and is supported), as some of the default answers will be incorrect.

As answers are entered for each question (either defaults or otherwise), they are saved in a temporary file. At the end of *unpackmodel*, before the UM is unpacked, the user is able to save these answers to a file called *config_$VN.cache* (hereafter referred to as *config.cache*). In subsequent executions of *unpackmodel*, the user can choose to have the default answers taken from *config.cache* instead of *mach_defaults*, which should reduce the time spent providing responses to questions, especially when unpacking on an unsupported platform. If a mistake is made whilst answering one of the questions in *unpackmodel*, it is worthwhile continuing with the remainder of the questions, and finally saving the responses in *config.cache*. The user can then re-run *unpackmodel*, hitting <return> at every question, but correcting the mistake when they come to it.

If a *config.cache* file exists, and the user chooses a different platform type for unpacking – this should never be the case, but could happen by mistake – the *config.cache* file is removed and answers revert to being taken from *mach_defaults*.

All answers are recorded in a log file, *unpack_$VN.log*, where $VN is the version number of the UM, but the experienced user can also take information from the *config.cache* file. If the log file already exists (from a previous run of *unpackmodel*), then the user is asked if they wish to overwrite it. A negative answer will cause *unpackmodel* to exit, enabling the user to rename the log file.

Answers of a logical type are checked for validity. A positive answer, which constitutes one of the following: y, yes, t, true, in upper or lower case, will be evaluated as “true”. Likewise, negative answers are “false”; q, quit, x or exit will cause the program to exit, but *config.cache* will not be updated.

B.2. Questions and Answers

The following is a list of questions and default answers, which can be used as a reference whilst running unpackmodel.

B.2.1. Stage 0 - Preliminaries

Step 0.1: Check Machine Defaults File Exists (all platforms)

No questions, just checks that the machine defaults file has been extracted from umsl.

Step 0.2: Choose Source of Defaults (all platforms)

If *config.cache* exists, then
Q: $HOME/config.4.5.cache exists. Use the defaults from here?
A: y

Step 0.3: Choose Quick or Slow Unpack (all platforms)

If using defaults from *config.cache*, then
Q: Unpack the UM, taking answers directly from *config.cache*?
A: n

If you choose to do a quick unpack, i.e. you have already been through *unpackmodel* at least once and the answers in *config.cache* are correct, then stages 1 to 6 inclusive will be skipped.

B.2.2. Stage 1 - Machine Type, Directory Structure, Mods and Paths

Step 1.1: Select Machine Type (all platforms)

Q: Please select a platform from the list of platforms below
1) Cray T3E (Met Office only)
2) Cray T3E (Generic)
3) Cray Parallel Vector Processor
4) Hewlett Packard Workstation
5) Digital Alpha Workstation
6) SGI Octane/Origin
7) Linux x86 using Fujitsu Fortran90 compiler
8) Other
x) Exit script

A: 1

Step 1.2: Build Directory Structure (non-Met. Office platforms only)

Q: Enter UM_HOME
A: $HOME (evaluated)

Q: Enter UMDIR
A: $UM_HOME/um (evaluated)

Q: Enter UMTEST
A: $UM_HOME/umtest (evaluated)

Q: Enter MY_UMHOME
A: $HOME (non-evaluated)

Q: Enter UM_TMP
A: $MY_UMHOME/tmp (non-evaluated)

Q: Enter MY_OUTPUT
A: $MY_UMHOME/umui_out (non-evaluated)

<u>Step 1.3: Script Modifications (non-Met. Office platforms only)</u>

Q: Enter SCRMODS, the file containing any mods for the scripts
A:

<u>Step 1.4: Utility PATHs (non-Met. Office platforms only)</u>

On some platforms, the utilities awk, sed and grep are not compatible with the UM. In these cases, different (gnu) versions can be used by supplying the relevant path of the executable (already installed on your system).

Q: Change the default location of awk?
A:

Platform Number	1	2	3	4	5	6	7	8
CHANGE_AWK	false	false	false	false	false	false	false	false

If CHANGE_AWK is true, then

Q: Enter UM_AWK, the path and name of the awk utility
A:

Q: Change the default location of sed?
A:

Platform Number	1	2	3	4	5	6	7	8
CHANGE_SED	false	false	false	false	false	false	false	false

If CHANGE_SED is true, then

Q: Enter UM_SED, the path and name of the sed utility
A:

Q: Change the default location of grep?
A:

Platform Number	1	2	3	4	5	6	7	8
CHANGE_GREP	false	false	false	false	false	false	false	false

If CHANGE_GREP is true, then

Q: Enter UM_GREP, the path and name of the grep utility
A:

B.2.3. Stage 2 - Compiler Options / Software Management

Step 2.1: Select Number of Parallel Compilations (all platforms)

Q: Please select the number of compilations that can run in parallel on your machine
A:

Platform Number	1	2	3	4	5	6	7	8
NPROC	8	4	4	1	1	1	1	1

Step 2.2: Compilers & Options (all platforms)

Q: Enter the name of your C compiler
A:

Platform Number	1	2	3	4	5	6	7	8
CCOM_CMD	cc	cc	cc	cc	cc	cc	gcc	

Q: Enter the C compiler options
A:

Platform Number	1	2	3	4	5	6	7	8
CCOM_OPTS					-g3	-ansi -64 -mips4 -r10000	-ansi	

Q: Enter the C compiler optimizations
A:

Platform Number	1	2	3	4	5	6	7	8
CCOM_OPTIM					-O	-O2		

Q: Enter the name of your Fortran compiler
A:

Platform Number	1	2	3	4	5	6	7	8
FCOM_CMD	f90	f90	cf77	f90	f90	f90	f90	

Q: Enter the Fortran compiler options
A:

Platform Number	LCOM_OPTS
1	
2	
3	-eh
4	
5	-g -i8 -r8 -automatic -align dcommons
6	-default64
7	-X9 -f s -Ccd4d8
8	

Q: Enter the Fortran compiler optimizations
A:

Platform Number	1	2	3	4	5	6	7	8
FCOM_OPTIM	-Oaggress -Oscalar3	As 1	-Oaggress		-O	-O2 -OPT:Olimit=0	-Kfast	

Q: Enter the name of your linker
A:

Platform Number	1	2	3	4	5	6	7	8
LCOM_CMD	f90	f90	segldr	f90	f90	f90	f90	

Q: Enter the linker options
A:

Platform Number	LCOM_OPTS
1	-Wl"-Dpermok=yes;streams=on"
2	-Wl"-Dpermok=yes;streams=on"
3	
4	
5	-warning_unresolved
6	-64 -Wl,-ignore_unresolved,-multigot
7	-noinhibit-exec -Bstatic -Wl,-warn-once
8	

Q: Enter any library path options
A:

Platform Number	LCOM_PATH
1	-L. -L\$(UMDIR)/gcom/rel_1m1s5x2
2	-L. -L\$(UMDIR)/gcom/rel_1m1s5x5
3	
4	
5	
6	-L. -L\$(UMDIR)/gcom/rel_1m1s5x5
7	
8	

Q: Enter any library options
A:

Platform Number	LCOM_LIBS
1	-lgcom1m1s5x2_shmemnam -l_vect -lgrib
2	-lgcom1m1s5x5_shmemnam
3	-lbnch -lmet
4	
5	
6	-lfastm -lgcom1m1s5x5_mpi -lmpi
7	
8	

Q: Enter the file (with full path) containing any compile options for specific decks
A: $UMDIR/vn$VN/source/comp_specific (non-evaluated)

Step 2.3: Fortran/C Interface and Wordlength (all platforms)

The Fortran/C interface (FC_LINK) specifies how C routine names are interpreted by Fortran. This can change between platforms and compilers; C_UP signifies the names are in upper case, C_LOW means lower case, and C_LOW_U means lower case with trailing underscore.

A false value for FRL8 indicates that a C float has equivalent wordlength to a Fortran real; true means that a C double is equivalent. The value will change with platform, and precision of the compilation.

INTLL is generally false, since we specify that real, integer and logical variables must all have the same wordlength (i.e. if FRL8 is false, then a C integer is equivalent to a Fortran integer, and if FRL8 is true, a C long will be equivalent). However, under Linux, we have found that when FRL8 is true, a C long long has equivalent wordlength to a Fortran integer.

FRL_SIZE is not used in the UM – it is just included in unpackmodel for your information, and gives the wordlength of the compilation.

BIGEND denotes whether the data representation is big-endian. A value of false indicates that the data needs to be operated on by the "bigend" utility. This is left for the user to do.

Q: The interface and wordlength can be calculated, or the defaults can be used. Do the calculations?
A: y

Defaults are:
A:

Platform Number	FC_LINK	FRL8	BIGEND	FRL_SIZE	INTLL
1	C_UP	true	true	64	false
2	C_UP	true	true	64	false
3	C_UP	true	true	64	false
4	C_LOW	false	true	32	false
5	C_LOW_U	true	false	64	false
6	C_LOW_U	true	true	64	false
7	C_LOW_U	true	false	64	true
8					

If BIGEND is false, then

Q: This platform requires data to be little-endian - the bigend utility is required.
Enter the C compiler flag for ANSI compliance
A:

Platform Number	1	2	3	4	5	6	7	8
ANSI_C					-std1		-ansi	

<u>Step 2.4: Nupdate Software Manager (all platforms)</u>

Q: There are 2 flavours of nupdate:-
(1) Cray - recommended for Cray
(2) Portable - not tested on Cray
You should use one or the other. Use Cray nupdate?

A:

Platform Number	1	2	3	4	5	6	7	8
USE_CRAY_NUPDATE	true	true	true	false	false	false	false	false

If USE_CRAY_NUPDATE is false then

Q: The supplied version of portable nupdate is 0.4.9. If you wish to use another version, enter it here.
A:

Q: Enter the directory containing the top level nupdate directory.
A: $UMDIR/vn$VN/utils

If USE_CRAY_NUPDATE is true then

Q: Enter the directory containing the nupdate executable
A:

Step 2.5: Banner Command (non-Met. Office platforms only)

If the banner command cannot be located on your platform, then the figlet program from $UMDIR/vn$VN/utils/figlet-1.1 is compiled and used to provide large lettering in the UM.

B.2.4. Stage 3 - Platform Definitions

Step 3.1: Execution Mode (all platforms)

Q: Build the MPP code?
A:

Platform Number	1	2	3	4	5	6	7	8
MPP	true	true	false	false	false	true	false	
MPPRECON	true	true	false	false	false	true	false	

Q: Build the non-MPP code?
A:

Platform Number	1	2	3	4	5	6	7	8
BLDNOMPP	false	false	true	true	true	false	true	

Step 3.2: Platform questions for machine OTHER (platform OTHER only)

There are no default answers for any question in this section

Q: Is this machine a CRAY supercomputer?
Q: Is this machine a T3E?
Q: Is this machine a T3D?
Q: Is this machine a Fujitsu VPP?
Q: Is this machine a SGI Origin?
Q: Is this machine a Sun?
Q: Is this machine an x86 Linux?

Step 3.3: Platform specifics (T3E & MPP machines only)

If T3E, then

Q: Some Cray T3E machines have a fast vector library called lib_vect.a. Would you like to use this library?
A:

Platform Number	1	2	3	4	5	6	7	8
VECTLIB	true	false	false	false	false	false	false	

If VECTLIB is true, then

Q: Please enter the pathname (no -L) of lib_vect.a if it is neither in a default library location nor in the list below:
A:

If MPP, then

Q: Some MPP machines have a labelled high memory PE. Enter the label for this PE.

A:

Platform Number	1	2	3	4	5	6	7	8
UM_PE_LABEL	HHIGHMEM							

B.2.5. Stage 4 - Directory Location

Step 4.1: Locate UMPL & UMSL (non-Met. Office platforms only)

No questions – moves the required versions of umpl and umsl (depending on whether portable or Cray nupdate is being used) to the correct locations. Cray nupdates expects umpl and umsl to be single files; portable nupdate requires directory format, containing the program and script decks.

Step 4.2: Build Portable Nupdate (non-Met. Office platforms only)

No questions – if required (USE_CRAY_NUPDATE=false), portable nupdate is built using the same compiler options as for the UM.

B.2.6. Stage 5 - Set up Nupdate *IF DEF Variables

Step 5.1: Set up PROGDEFS (all platforms)

No questions – the nupdate *DEFS for the source code are brought together in PROGDEFS

Step 5.2: Set up SCRDEFS (all platforms)

No questions – the nupdate *DEFS for the scripts are brought together in SCRDEFS

B.2.7. Stage 6 - File Creation

Step 6.1: Setvars File (non-Met. Office platforms only)

No questions – the file $UM_HOME/setvars_4.5 is created. It is linked to $UM_HOME/setvars, which is then run to provide the shell with environment variables. The file should be run each time the user logs on, by including some code to do this is the .profile or .kshrc. The setvars file should be linked or copied to the users $HOME directory.

Step 6.2: Setglobalvars File (all platforms)

No questions – the file $UMDIR/vn$VN/scripts/setglobalvars is created. This file is sourced each time the UM is run, from within the scripts produced by the UMUI.

Step 6.3: Config.cache File (all platforms)

Q: OK to overwrite config.cache file?
A: y

B.2.8. Stage 7 - Model Unpacking

Step 7.1: Script Extraction (all platforms)

Lists values of all parameters set in *unpackmodel*, and then

Q: OK to go ahead and unpack the UM with these options?
A: y

If answer to this question is "true", then, using *nupdate*, the script *mkscripts* is extracted from umsl, and then run. This in turn extracts all required scripts from umsl and locates them in the correct directory.

Step 7.2: Create cross reference files (all platforms)

No questions - creates *obj_xref*, the object cross-reference file, and *exec_xref*, the small executables cross-reference file

Step 7.3: Create compile_vars file (all platforms)

No questions – creates *compile_vars* which contains the general compile options and any deck specific compile options.

Step 7.4: Tidy up (all platforms)

No questions – tidy-up performed and reminder issued.

C : Unresolved External References

C.1. Unresolved Externals for the Global Climate Model (xaaa)

ocean_sizes_
iau_ctl_
iau_
clock_
date_
o_lbc_sizes_
rhcrit_calc_
var_umprocessing_
trbdry_
trsrce_
trac_vert_adv_
trac_adv_
set_trac_
stratq_
ac_
lwlkin_
swlkin_
var_umsetup_
ac_init_
stwvgt_
stocgt_
sootscav_
new2old_
sulphur_
gravsett_
swdkdi_
lwrad_
swrad_
vdif_ctl_
rad_moses_
area_cld_
coder_
ibm2cri_
cri2ibm_
strmov_
movbit_
tridiag_solver_up_
set_matrix_net_

C.2. Unresolved Externals for the Global Ocean Model (xaab)

ocean_sizes_
oa_zero_
oc_ac_ctl_
clock_
date_
init_emcorr_
init_hyd_
in_acctl_
remlnd_
oa_int_1d_
oa_int_lev_
stwvgt_
coder_
ibm2cri_
cri2ibm_
strmov_
movbit_
p_to_cv_
p_to_cu_
p_to_uv_
cnvstop_

C.3. Unresolved Externals for the Mesoscale Model (xaac)

ocean_sizes_
iau_ctl_
iau_
clock_
date_
init_emcorr_
o_lbc_sizes_
rhcrit_calc_
var_umprocessing_
add_eng_corr_
cal_eng_mass_corr_
eng_mass_diag_
stratq_
ac_
zonm_atm_
r2_lw_specin_
r2_sw_specin_
var_umsetup_
ac_init_
stwvgt_
stocgt_
sootscav_
new2old_
sulphur_
gravsett_
flux_diag_
r2_global_cloud_top_
gas_calc_
r2_lwrad_
r2_swrad_
tropin_
gwav_intctl_
vdif_ctl_
rad_moses_
rad_degrade_mask_
coeffs_degrade_
coder_
ibm2cri_
cri2ibm_
strmov_
movbit_

D : Modifications for the Portable UM

D.1. Source Modifications

The following mods have been used to create the new program library (umpl) which is part of the PUM release, and they can be found under $UM_HOME/PUM_Input/vn4.5/mods/builtin/source:

- C mods

portio2a.mod	Several fixes - fix warning/error for fname; make sure buffer is flushed; and allow long long type

- Fortran mods

blkdata.mod	Add extra data statements to the BLOCKDATA subprogram so they are only initialised once.
blokcntl.mod	Change .T. to .TRUE.
blokinit.mod	Removes warning messages about undefined variables
cgrelax.mod	Deletes empty lines at the end of the deck
compare1.mod	Corrects double comma in format statement
dfgrou1a.mod	Adds extra data value to make 61
flush.mod	A wrapper script to the flush intrinsic
genintf1.mod	Remove flush from makebc because its not required, and flush isn’t available on all platforms
hdppxref.mod	fix to interchange 2 lines to allow compilation under f77
inicmc.mod	Required * adding to read/write
lsqls2.mod	Make sure IRP1 is saved for all platforms
makebc.mod	Correct typo intf_ct1 (1 should be l)
meandia2.mod	Separates if tests into 2 bits
namelst.mod	Adds a delim='apostrophe' keyword to all opens of namelists
negq_reset.mod	Resets any negative values of q to zero
negzero.mod	Make sure this deck is only extracted on the T3E
oahor1a.mod	Bug fix (change DEF,-OADGHR2 to -DEF,OADGHR2 etc.)
oard1a.mod	Add comma to end of format statement
oasistep.mod	Required * adding to read/write
ofltcn2a.mod	Big code correction
oislesum.mod	Corrects variable names
packing.mod	Allows PACK21/EXPAND21 to be used on most platforms
pphead1a.mod	Bug fix (DEF,-CRAY to -DEF,CRAY)
prmch3a.mod	Allows machine tolerance to vary with machine precision
reorder.mod	Re-order declaration of arrays so that the array dimension variables are declared before the arrays
rmlines.mod	Removes lines in some decks which are outside the main *IF DEF
scontrol.mod	Declare NDATASETS before it is used to dimension array
timefn2a.mod	Changes for how various platforms handle time
timer1a.mod	Changes to stop writing to read-only variables

timer3a.mod	Make sure parvars is *CALLed for all MPP runs
udg1f406	Correct compile error in non-T3E branch
umshell.mod	Mainly code re-ordering for SGI
upbound.mod	Corrects IF test which might fall over if FLOOR_STEPSA=0
vectlib.mod	Add new DEF VECTLIB which controls use of Cray fast vector libraries
vertcld.mod	Remove comma after DO
wvcheck.mod	Correct double comma in format statement
zonmat1a.mod	Remove double comma from format statement

D.2. Script Modifications

The following mods have been used to create the new scripts library (umsl) provided with the PUM, and they can be found under $UM_HOME/PUM_Input/vn4.5/mods/builtin/scripts:

configure.mod	Adds the default submission method to minimise hand-edits
copyright.mod	Adds a copyright notice to the top of each script
flush.mod	Script changes to allow new wrapper subroutine to flush
glr1u406	Allows separate compile and link steps
makefile.mod	Re-orders the load line
metocray.mod	Removes DEF METOCRAY from around code which is to external users too
mkexecxref.mod	Allows for longer alldefs lines
mkobjxref.mod	Add in extra supported platforms
mkscripts.mod	Allows mods to be passed to the scripts before they are extracted. Extra platforms added to mkscripts
pptoanc.mod	Set default (general) values for EXEC
qsconf.mod	Remove files which aren’t required for each model section
qsconf2.mod	Remove zero length files
qsmain.mod	Assigns environment variable (ARG1) to value of $1 Changes /dev/null filename for LINUX due to error closing it Adds DEF SGI and associated code
qsmain2.mod	Remove zero length files and nupdate log files
qsprelim.mod	Control use of setlabel on reconfiguration by use UM_PE_LABEL
submitchk.mod	Use -f to force removal of file
unixutil.mod	Changes all occurances of awk, sed and grep $AWK, $SED and $GREP
unpackmodel_intll_fig.mod	Complete rewrite of unpackmodel
xref_intll_frl8.mod	Changes to add in values for new platforms