UNIFIED MODEL DOCUMENTATION PAPER Z52

UNIFIED MODEL COMPILATION SYSTEM

BY

R. HATCHER

Version 1

27 January 1999

Model version 4.5

Climate Research
Meteorological Office
London Road
BRACKNELL
Berkshire
RG12 2SY
United Kingdom

Modification Record		
Document version	**Author**	**Description.....**
1	R.Hatcher	First version

Introduction

This document aims to give a quick introduction to the Unified Model compile system for users doing runs at vn4.4 and beyond. It contains information about the system and practical examples of how to use some of the main features.

The sections describe the main sequence of actions performed by the compile system in a user run, how to understand messages in model output, the main variables a user needs to know about, how to set up compile override files, miscellaneous information on how to do various common tasks and finally how to build your own small executables.

Please see UMDP Z5 (System Maintenance Procedures) for more detailed information about the build system, which overlaps with some aspects of the compile system.

Actions performed in a User Run

The main functions of the compile system are listed below in the order in which they happen:

Reconfiguration

- Central and user compile options merged into one file
- Directory $URECONDIR created with subdirectory exec_build/qxrecon_dump
- nupdate used to extract source code for modified Fortran/C decks into a $TMPDIR directory
- In the reconfiguration compile directory ($URECONDIR/exec_build/qxrecon_dump) .f and .c source files are created for decks that have changed since the previous run or for all decks on the initial run
- Makefile automatically generated for all decks required
- *make* command is run to compile all uncompiled source code and link to make the executable
- Executable moved to $LOADRECON
- Label added to executable (UK Met. Office only)

Model compilation

- Central and user compile options merged into one file $UCOMPVARS
- nupdate used to extract source code for modified Fortran/C decks into a $TMPDIR directory. This includes:
  - All modified decks
  - Control decks from all relevant parts of $UMDIR/vn$VN/source/cdecks file
  - All decks required for section-version DEF combinations not pre-built.

  All deck source files are renamed as lower-case and .f/.c (Fortran/C) suffixes added.
- In compile directory $UCOMPDIR (the name of this directory should be unique to a runid eg. $DATAW/compile_$RUNID):
  - file links are set up in a previous compile of the same runid deleted
  - Source code (.f/.c) files in $TMPDIR nupdate directory compared with existing copies of the same files in compile directory. If different, newly-extracted deck replaces original deck (or used directly in initial run).
  - If using a section-version DEF combination not prebuilt, directory $REBUILD_DIR created. Relevant source files copied to subdirectories under here. Links created from compile directory to these files.
  - For other decks links are created to central copies of source code (.f/.c) files and object code (.o) files for precompiled code stored on a system-wide basis under the relevant $UMDIR/vn$VN/src/section/def-combination directory. The nupdate *DEF combinations are specified by codes eg. A1G1 for *DEFs ATMOS (A1) and MPP (G1). See top of $UMDIR/vn$VN/source/obj_xref file for *DEF code letters.
  - For decks whose compile options have changed since the previous run, old versions of the .o files are deleted to force recompilation.
- A makefile (called $UCOMPDIR/Makefile) is automatically generated for all decks required in run (unless previous version of makefile still valid)
- file links are set up in a previous compile of the same runid deleted
- *make* command is run to compile all source decks that have not been compiled or have been

modified since they were last compiled and also to run link step to make executable.
- Executable is run.

In second or subsequent runs it is possible to miss out the nupdate extraction and other script processing steps and go straight to the *make* step by setting SKIP_SRC_EXTRACT to true (see 'How To' section for more details). This is useful in conjunction with hand-editing the source files to speed up debugging.

The *make* utility works in the following way. To create an executable from a number of fortran files, the fortran files are compiled to create .o object files which are then linked to create the executable. To recreate the executable after just one or two fortran files are changed, only those altered files need to be recompiled before the executable is linked. In the UM compile system, the *make* utility works by comparing the dates of the files to determine what work needs to be done. So if one fortran file is changed, *make* sees that the .f file is newer than its .o file and so recompiles it. Then it will find that the .o file is newer than the executable and so it will relink the executable.

Browsing Compile Output and Error Messages

Order of information in job output files

When looking for information from the compile system first look at the script output from qsprelim and qsmain at the top of the model output file for messages as to whether the compile script processing and *make* steps completed successfully.

If not, then the detailed error messages can be found further down in the output. Search on the string **%QSMNCOMPILE** for the start of the output from the pre-processing compile scripts and on **%MAKE**. for the output from the *make* step itself (but note that these strings will also show up in the echo statements in SCRIPT if the job asks for a copy of SCRIPT to be listed in the output).

See the troubleshooting section of the User Guide for a description of the types of output in the model output file and the order they appear in.

See also the later section in this document giving an example of the output that appears after %QSMNCOMPILE and %MAKE, and an in-depth explanation of what all the messages and codes mean.

Note that at vn4.4 the reconfiguration compile output still appears in full near the top of the output file. At vn4.5 this will be moved further down the output file.

Understanding ERROR or WARNING messages

A run may fail during compilation. The message output from qsmain is.

```
qsmain(934): ***  Make has failed. Exiting
```

The actual error will be found near the bottom of the *make* output after %MAKE in the model output file. This should give an indication why it has failed. The most likely reason *make* might fail is due to a compilation error, but it can also fail for system reasons such as because there is not enough memory available. The error message may not be the last message in the output if there are several compilation processes running in parallel (due to NPROC being greater than 1) since *make* waits for the other current compilations to complete once an error has been detected.

Upon successful completion of the *make* command there may still be WARNING messages in the output. Often these can be ignored but sometimes they are significant. See the next few sections for some examples.

On Cray MPP machines the use of ‘‘-Dpermok=yes’’ in the load line allows the run executable to be created even if WARNING messages have been generated. This is necessary because there are always parts of the model not being used by a configuration for which the top routine calls are still there but the code is not included.

Missing externals

The main kind of WARNING, a T3E example of which is shown below, sometimes leads to failures in the model with missing externals.

e.g.

```
cld-404 cld: WARNING
The symbol ‘CHK_LOOK_BOUNDA’ referenced in relocatable object
‘inbound1.o:IN_BOUND’ is not defined.
```

It is worthwhile, if the model fails in this way after starting running, checking the relevant .f and .o files exist in the user compile directory and are not empty. (However, normally the compile system cleans up empty files caused by a system failure part way through a model compilation and this causes those routines to be recompiled.)

Occasionally when a sequence of changes are made and there are also system problems, a routine may be missing from the makefile when it is generated and hence is missing from the executable when made. This can be rectified by editing the Makefile manually and using SKIP_SRC_EXTRACT=true (see ’How to’ section) or by deleting all the files in the compile directory and recompiling again to get a clean compile.

Vector directives (Cray MPP only)

The UM when running on Cray MPP machines gives the following type of warnings:

```
CMIC$ DO ALL VECTOR SHARED(U_POINTS, ROW_LENGTH, U1, SU10, SV10, U_FIELD
      ^
cf90-801 f90: WARNING IMPL_CAL, File=implca2c.f, Line=1491, Column=7
  Unsupported compiler directive.
```

These kinds of warnings can be ignored. They arise because because compiler directives exist in the

code to enable the model to run on a vector parallel machine (e.g. an SGI/Cray C90).

Forcing *make* to continue through compilation errors

USE WITH CAUTION
By default *make* stops during compilation if there has been an error. An error code is returned and the run terminated. It is possible to force *make* to continue compilation after there has been an error. This means all errors can be identified in a single pass and object code will be created for decks which do not have any errors.

This is useful for debugging but unsafe in general because a success code is now returned from *make* regardless of whether there has been an error. It appears to *make* that compilation has been successful and the model run would be started after calling *make* in this way even though there were errors.

To force *make* to carry on past compile errors change the call from 'make' to 'make -k' in qsmain using a script mod and run with STEP=0 in the SUBMIT job library file (or set job to 'compile only' in the UMUI). At the UK Met. Office this change is available as fix mod $UMDIR/vn4.4/mods/fixes/fkr4u404 which contains:
*IDENT FKR4U404
*DECLARE qsmain
*D gex3u405.24
  make -k >>$OUTPUT 2>&1

Output Messages in more detail

The following is selected output from the compilation part of a UM run. This is not totally realistic as some of the messages will not appear under some conditions but is representative of what will generally appear. The first field indicates the script that is giving the message. A numeric value in parenthesis is used for system timing and indicates the number of seconds the current shell (ie UM run) has been in existance for. The information is split into eleven discrete sections. An explanation of each of these sections follows.

```
qsmncompile(191): ***   Compile vars diff information.
qsmncompile(193): ***     System differences.
qsmncompile(194): ***   End of compile vars info.

qsmncompile(222): ***   Removing symlinks in compilation dir.
qsmncompile(401): ***   Finished removing symlinks

qsmncompile(401): ***   Removing zero length files
qsmncompile(409): ***   Finished removing files

qsmncompile(414): ***   Removing previously modified files
qsmncompile(414): ***     This looks new. No files to remove
qsmncompile(414): ***   Finished removing modified files

qsmncompile(414): ***   STARTING control source comparison
qsmncompile(416): ***     abcalc1       (Same)  (F)
qsmncompile(416): ***     ac_ctl1       (Same)  (F)
```

```
qsmncompile(338): ***     chemctl1      (New)   (F)
qsmncompile(339): ***     cldctl1       (New)   (F)
qsmncompile(341): ***     conv_ct1      (New)   (F)
qsmncompile(351): ***     divcal1a      (Diff)  (F)
qsmncompile(352): ***     dosums1       (Diff)  (F)
qsmncompile(494): ***     st_dia11      (Same)  (F)     (Opts:new)
qsmncompile(518): ***     zonmctl1      (Same)  (F)
qsmncompile(518): ***   FINISHED control source comparison

qsmncompile(518): ***   Symlink-ing pre-compiled code
qsmncompile(576): ***     Linking /u/um1/vn4.4/obj/a01_1b/Gp/NORMAL
qsmncompile(577): ***       ftsa1a      (F:ok)  (O:ok)
qsmncompile(578): ***       solang1a    (F:ok)  (O:ok)
qsmncompile(579): ***       solpos1a    (F:ok)  (O:ok)
qsmncompile(579): ***       swclop1a    (F:ok)  (O:ok)
qsmncompile(584): ***     Linking /u/um1/vn4.4/obj/a02_1b/Gp/NORMAL
qsmncompile(584): ***       lwcld1a     (F:ok)  (O:ok)
qsmncompile(584): ***       lwdcsf1a    (F:ok)  (O:ok)
qsmncompile(657): ***     Linking /u/um1/vn4.4/obj/a15_1a/G1G3/NORMAL
qsmncompile(658): ***       calcp21a    (F:ok)  (O:ok)
qsmncompile(659): ***       calcpv1a    (F:ok)  (O:ok)
qsmncompile(660): ***       cat1a       (F:ok)  (O:ok)
qsmncompile(661): ***       dthdp1a     (F:ok)  (O:ok)
qsmncompile(662): ***       dyndia1a    (F:ok)  (O:com) (Opts:ok)
qsmncompile(676): ***       omegdi1a    (F:ok)  (O:ok)
qsmncompile(677): ***       pvpin1a     (F:ok)  (O:ok)
qsmncompile(678): ***       pvthin1a    (F:ok)  (O:ok)
qsmncompile(697): ***     Linking /u/um1/vn4.4/obj/c95_2a/G1GnP4/NORMAL
qsmncompile(697): ***       portio2a    (C:ok)  (O:ok)
qsmncompile(813): ***   Finished symlink-ing object code

qsmncompile(813): ***   Generating sed script
qsmncompile(814): ***   Sed script was generated okay
qsmncompile(814): ***   Running sed to create Makefile
qsmncompile(815): ***   Finished sed

qsmain(819): ***        Running make

%MAKE
        f90  -Oaggress -Oscalar3  -c abcalc1.f
        f90  -Oaggress -Oscalar3  -c ac_ctl1.f
        f90  -Oaggress -Oscalar3  -c acumps1.f
        f90  -Oaggress -Oscalar3  -c addrchk1.f
        f90   -c st_dia11.f
        f90   -c dyndia1a.f
        f90 -Wl"-Dpermok=yes;streams=on" -lgcom1m1s4x1_shmemnam
            -l_vect -lgrib -L. -L/u/um1/gcom/rel_1m1s4x1
            -o /mydir/me/aaxid/aaxid ac_ctl1.o acumps1.o \
            addres1.o addrln1.o atmdyn1.o atmphy1.o atmstep1.o \
            bl_ctl1.o boundva1.o chemctl1.o cldctl1.o \
            conv_ct1.o dervsize.o diag3a.o
.......etc..

cld-404 cld: WARNING
The symbol 'AC' referenced in relocatable object 'ac_ctl1.o:AC_CTL'
is not defined.
cld-404 cld: WARNING
The symbol 'STOCGT' referenced in relocatable object 'stwork1a.o:STWORK'
is not defined.
cld-404 cld: WARNING
  etc
```

```
qsmain(934): ***        Completed make
```

1. Extraction nupdate step to COMPDIR. All decks that are either specified in the cdecks file or modified by mods directly or indirectly, via comdecks, included in the UM configuration are extracted.
2. Rename extracted decks in COMPDIR. The decks are stored in the UMPL as uppercase without file type suffixes.
3. Remove any previous symlinks in UCOMPDIR. The symlinks are removed every time to enable decks that are newly modified to be moved into place.
4. Remove any zero length files in UCOMPDIR. A precaution. Some decks from previous compilations may have no effective source code or may not have produced complete object files due to compilation errors.
5. Remove any decks that are no longer modified wrt previous run, if necessary. This is done so that files can be symlinked in the symlink step.
6. Compare source in COMPDIR with any source in UCOMPDIR. Move it in if different from previous run code or new, leave it intact in UCOMPDIR if there has been no code change since previous run. Extract by-deck compiler options for later inclusion in Makefile. The following messages can occur under different conditions:

   (Same) This is a secondary run. The code for the deck found in the UCOMPDIR is the same as that extracted. The source is not updated. Re-compilation will not be needed, unless the compiler options have changed since the last run, for this deck.

   (New) This is either a primary run and no decks exist in the UCOMPDIR directory at all or a secondary run into which a new deck (proabably from a user mod) is being added.

   (Diff) A secondary run which has changed mods wrt to the previous run. The deck that has been extracted from the UMPL (including mod) into UCOMPDIR is different from the previous version in UCOMPDIR. The new version is moved into UCOMPDIR. This will force make(1) to re-compile at a later stage.

   (F) The deck has been identified as Fortran source code.

   (C) The deck has been identified as C source code.

   (Opts:new) This deck has specific compiler options defined in the UCOMPVARS file. These will be used instead of the default compiler options.

7. Produce sym-links to required pre-compiled source code and object files. Extract by-deck compiler options for later inclusion in Makefile, if necessary. Some decks exist in more than one section that may be required simultaneously. These decks contain identical code and are compiled in the same way, so the code and object from the first occurance of the deck are included. Subsequent occurances are skipped. If the relevent section-def combination is not found in the UM system pre-built directory and BUILDALL is set to ‘‘true’’ then script qsconf is called to extract the relevent code in to BUILDALL_DIR and then make(1) used to compile it. The files are then linked in the same way as for system pre-built code. The following messages can occur during this

stage.

Only one of the following messages should exist for each deck.

(F:ok) The Fortran source for the section-def combination has been linked into the UCOMPDIR directory from the system pre-built source directories.

(F:src) The Fortran source for this deck already exists in the UCOMPDIR directory. This is because the deck has been modified and extracted by the previous nupdate extract step in the current user run.

(F:Sec) The Fortran source for this deck already exists in the UCOMPDIR directory. This is because the deck was symlinked in a previous section-def combination during the current user run.

(F:OSN) ‘‘Outside System, New’’ - The Fortran source for this deck already exists in the UCOMPDIR directory. This is because the user has removed a symlink file and created a ‘‘real’’ file. The deck will gain a ‘‘rule’’ in the Makefile with compiler options from the UCOMPVARS file. NOTE: this source deck is completely under the users control. It will not be deleted by the UM build system. The user must delete it (and it’s object file) directly if the system default is to be used.

(F:OSO) ‘‘Outside System, Old’’ - The deck was in a previous section-def combination and has been labelled as an OSN file already. See previous item.

(C:ok) The C source for the section-def combination has been linked into the UCOMPDIR directory from the system pre-built source directories. Note: only the first source-object pair encountered in the pre-built section-def combinations is linked to.

(C:src) The C source for this deck already exists in the UCOMPDIR directory. This is because the deck has been modified and extracted by the previous nupdate extract step in the current user run.

(C:Sec) The C source for this deck already exists in the UCOMPDIR directory. This is because the deck was symlinked in a previous section def combination during the current user run.

(C:OSN) ‘‘Outside System, New’’ - The C source for this deck already exists in the UCOMPDIR directory. This is because the user has removed a symlink file and created a ‘‘real’’ file. The deck will gain an ‘‘rule’’ in the Makefile with compiler options from the UCOMPVARS file. NOTE: this source deck is completely under the users control. It will not be deleted by the UM build system. The user must delete it (and it’s object file) directly if the system default is to be used.

(C:OSO) ‘‘Outside System, Old’’ - The deck was in a previous section and has been labelled as an OSN file already. See previous item.

Only one of the following messages should exist for each deck.

(O:ok) The object file for the section-def-compile combination has been sym-linked into the UCOMPDIR directory from the system pre-built source directories. Note: only the first source-object pair encountered in the pre-built section-def combinations is linked to.

(O:rm/ok) An object file for the deck already exists in the UCOMPDIR directory from a previous run. This is removed and a sym-link into the UCOMPDIR from the section-def-compile combination object created.

(O:rm) The object already exists in UCOMPDIR, but needs re-compiling as the entry in UCOMPVARS has been changed, thus it is removed.

If the first of the following messages exists, then the second message may or may not exist, depending whether or not the UCOMPVARS file has changed.

(O:com) An entry in the makefile wil be generated so that the deck can be compiled.

(Opts:ok) This deck has specific compiler options defined in the UCOMPVARS file. These will be used instead of the default compiler options.

If these messages are given then the above messages will not be given.

WARNING The section-def combination that has been requested ahs not been pre-built by the UM system team.

BUILDALL The section-def will be extracted and built in BUILDALL_DIR.

8. Generate sed script then Makefile. The sed script contains information on how information on compilation of decks should be parsed into the Makefile. The Makefile contains rules that indicate what needs to be compiled/linked and dependencies indicating which files depend on other files. This enables object files to remain up to date wrt to source files.
   Run make(1) to generate any required object files and link the object files into a binary executable. The make(1) step compiles decks that have no object, recompiles decks that have changed since their object was created and leaves objects that are older than their source (ie the source has not changed). You can see here exactly how each object is being compiled.

Compile System Variables

Set in UMUI

Variable Name	Typical Setting	UMUI panel	Description
UCOMPDIR	$DATAW/compile_$RUNID	subindep_FileDir	Directory in which compilation will take place. Must be unique to a run identifier.
URECONDIR	$DATAW/recon_$RUNID	subindep_FileDir	Directory below which reconfiguration compilation will take place. Must be unique to a run identifier.
REBUILD_DIR	$DATAW/rebuild_$RUNID	subindep_FileDir	Directory below which compilations of section-version combinations that are not held centrally are done. Must be unique to a run identifier.
NPROC	5	subindep_Compile	Number of compilation processes that run in parallel under *make* (not applicable on all machines). If unset uses version-wide default in setglobalvars.

Hand-edit in job library

Variable Name	Typical Setting	Description
SKIP_SRC_EXTRACT	false	Set to true in SUBMIT to hand-edit .f files and run *make* directly.
BUILDSECT	false	Set to true in SUBMIT to allow section-version DEF combinations not prebuilt centrally to be built within run.
SKIP_COMPILE_TAR	false	(UK Met. Office only). Set to true in SUBMIT to prevent tar/gzip being used on compile files.

Overriding compiler options

The options used to compile source code are supplied in the central file $UMDIR/vn$VN/source/compile_vars for each version. Overrides to the options in this file can be specified in one or more compile override files.

Note that compile override files are only used by the compile system when it does the script pre-processing steps when SKIP_SRC_EXTRACT is false. When SKIP_SRC_EXTRACT is true it uses whatever compile options are specified in the Makefile.

Step by step guide to using overrides

Below are the steps you need to go through to specify your own compile options:

1. Create a directory on the machine where you are running the UM to hold compile override files e.g. $HOME/comp_override.
2. Look at the file $UMDIR/vn$VN/source/compile_vars to see the format of lines. The keywords (@fort, @load etc) and tags (FCOM_OPTS and LOAD_OPTS etc) are case-sensitive. The amount of white space around options does not matter.

   Note that apart from FCOM_CMD, CCOM_CMD and LOAD_CMD, which are used to specify the Fortran compiler, C compiler and link/loader to be used, other options are just concatenated together in each type. For example on a T3E f90 platform, all the options specified on the FCOM_OPTS, FCOM_ENABLE, FCOM_DISABLE, FCOM_OPTIM and FCOM_ROPTS lines are just concatenated together by the compile system in a run. Having more than one identifier just makes it easy to specify related options together if preferred. New categories will also be recognised provided they start with FCOM_. (At present there is no method for continuing options onto a second line with any of these identifiers. For changes to compile_vars for the next version of the UM there is a line length limit of 72 characters on these lines due to portable nupdate limitations. However this limit does not apply to lines within user compile override files.)

All options should be specified after the '=' exactly as they would be on the compile line (such as in a f90 command on the T3E) with no need for extra quotes.

3. Any line in the compile_vars file can be overridden. Default lines are those starting with @ eg. @fort, @load. Be careful if you change a compiler default line (@fort) as this will recompile THE WHOLE OF your model code with this new default. Also large numbers of single-deck compile options will slow down the scripts.

   When overriding load options remember to include the standard default options from compile_vars as well as additional options unless you definitely don't want to use the standard defaults.
4. Create a file or a number of files to hold the lines which are being overriden. These override files must then be made available to the UM scripts via the UMUI in window **subindep_Compile_User** UMUI_Help/subindep_Compile_User.help

   ```
   Sub-Model Independent
   -> Compilation and Modification
   --> User-defined compile option overrides
   ```

   or by hand-editing job library file COMP_OPTS. The scripts do the work to include the overrides in the model run.

   If a particular mod requires special compile options a compile override file can be used in conjunction with that mod. This is a convenient way for a run owner to organise groups of changes and associated compile options, but it is up to that person to keep track of which compile override files go with which mods (there is no link within the UMUI).

   Comments can be included in compile overide files using a # in the first column.

Some examples of compiler overrides

The first set of examples show how the global defaults, i.e. those applicable to the whole model, might be overriden.

- Set the f90 -g (debug) and -ei (enable i) options.

  Provide an override file with the line,

  ```
  @fort FCOM_OPTS=-g -ei
  ```

  to replace the default line ''@fort FCOM_OPTS='' (remembering that this recompiles all routines including precompiled ones)
- Force a recompile of the entire model without changing compile options.

  Provide an override file with the line,

  ```
  @fort FCOM_ENABLE=-eq
  ```

  to replace the default line ''@fort FCOM_ENABLE='' (the -q option is already enabled but not specified in the compile_vars file so having the effect that all routines including precompiled ones are recompiled but with null effect.) This can be useful for recompiling with a different version of the compiler to that used for the prebuilt code.
- Use load libraries ''-lgrib -lgcom_pvm_11 -lconv''

Provide an override file with the line,
`@load LOAD_LIBS=-lgrib -lgcom_pvm_11 -l_conv`
to replace the default line ‘‘@load LOAD_LIBS=-lgrib -lgcom_shmnam_13 -lvect’’

As you can see from browsing the compile_vars file it is possible to apply compiler options to individual decks.

Given below are some examples which show how to override existing decks specific options and how to add further compiler options for a given deck.

- Set f90 optimisation option ‘‘aggress’’ in deck ADVCTL1A

  Provide an override file with the line,
  `ADVCTL1A FCOM_OPTIM=-O aggress`
- Use machine default compile optimisation for deck HORINF1A

  Provide an override file with the line,
  `HORINF1A FCOM_OPTIM=`
  to replace the default line ‘‘HORINF1A FCOM_OPTIM=-Ounroll2 -Oaggress -Oscalar3’’

Applying overrides to the reconfiguration

These work in exactly the same way as the model compile overrides but may be different overrides and have a separate list of override files.

Miscellaneous "How to ..." questions

1. **Hand-edit source files and just rerun *make* in a run.**

   It is well worth becoming familiar with this way of speeding up debugging since it will save many hours of time compared to running the full compile system BUT it is only suitable for making small non-permanent changes (for example adding print statements) since mods cannot be generated from the changes. It involves making alterations to copies of the actual fortran code kept in the compile directory of your job rather than writing mods. It avoids the significant overhead in time of extracting all the relevant decks, applying mods and *DEFs, expanding comdecks and recompiling many decks. Only those decks that have been changed will be recompiled before linking to produce the executable. This method does not apply to the reconfiguration.

   The job must have been compiled in the normal way first. After the run has compiled, all the relevant source and object code is left in a compile directory as specified in the UMUI (typically $HOME/$RUNID/compile_$RUNID). On typing "ls -l" in this directory, a list of source (.f and .c) and object (.o) files or symbolic links to such files will be seen. (Follow the instructions in the later section on how to get at .f files bundled into tar files (UK Met. Office only).

Files without symbolic links are generally control code, ocean decks and modified decks. Many decks, however, have been precompiled a number of times with different combinations of *DEFs. The source and object code for these decks are held centrally; unmodified decks are thus included by making symbolic links to these files. See the warnings below before attempting to edit linked files.

Make alterations to copies of the actual Fortran code kept in the compile directory of your job rather than writing mods. Only the .f files that have been changed will be recompiled.

By using the following hand edit, the source extraction and checking of which decks/compile options have changed can be avoided and the files in the compile directory will be reused:

In SUBMIT, hand edit SKIP_SRC_EXTRACT from 'false' to 'true' (where SUBMIT is one of the job library files produced by the UMUI).

Then resubmit the run as normal.

WARNINGS:

- This system cannot be used for editing the contents of comdecks. These are already included inline in the appropriate routines. Another consequence of this is that the edited decks cannot be used with nmodex to create a modset.
- Many of the files in the compile directories are, in fact, links to the libraries; these cannot be edited. Before editing a deck find out whether it is linked: \

  ls -l windmx1a.f \

  Linked files are shown as:

  ```
  windmx1a.f -> /u/um1/vn4.4/src/a15_1a/A1G1G3/NORMAL/../windmx1a.f
  ```

  If you need to edit such a deck, make a note of the path to the file to which the link is pointing and delete the links to the .f and .o files. Then copy the .f file into your compile directory. For example:

  ```
  cd $DATAW/compile_$RUNID
  ls -l windmx1a.f  # Note path
  rm windmx1a.f
  cp /u/um1/vn4.4/src/a15_1a/A1G1G3/NORMAL/../windmx1a.f .
  ```

  It is important that the file is copied from the correct directory since there will be a number of alternative directories containing a deck but only the file to which the link is pointing will have had the correct *DEFS applied to it.

  Additionally, the system will not know how to compile such decks since previously it just picked up the precompiled .o file. Before running with SKIP_SRC_EXTRACT you will need to either:

  Compile the deck by hand:

  f90 -c -Oaggress -Oscalar3 windmx1a

Or edit the Makefile to add compile instructions. eg, near the top add the line:

FFLAGS = -c -Oaggress -Oscalar3

which sets the default compile options for any FORTRAN files that do not have options listed.

- Another warning: If you do handedit a linked deck and then subsequently do a normal full recompilation (ie. set SKIP_SRC_EXTRACT=false again) then your edited deck WILL be replaced with the linked deck unless you have now written a mod for it. The compile system will assume that you want to revert to the original version. This prevents an altered deck remaining and affecting subsequent runs in the future which you may otherwise forget about. If you wish to continue using the hand-edited version then you will need to set SKIP_SRC_EXTRACT=true.
- Using SKIP_SRC_EXTRACT with an incomplete compile directory.

  If you hand-edit a source file by extracting it from the tar file you must make sure that you put the hand-edited file back into the tar file and set SKIP_SRC_EXTRACT to true before resubmitting the run. The compile system assumes that if there is no Makefile in the compile directory that this is an initial run and will reset SKIP_SRC_EXTRACT to false and so perform a complete new compile thus overwriting the hand-edited files.
- If you need to copy the contents of one job's Compile directory to use in another job. You will need to alter the following line in the copied Makefile to reference the new load module.

```
# What will our executable be called
BINARY=/u/m20/data/mec/t20ro/aboia/aboia.exec
```

  If this change is not made then the original load module will be overwritten.
  Hand-edits to the files can be made as described above.

2. **Run without compiling from an existing executable.**

   Set **STEP=4** in the SUBMIT job library file on the system where you run the UMUI.

3. **Change just load options.**

   It is possible to change the load options using a compile override file, for example to use different libraries. However if the code has already been compiled and you just want to change the load options or pick up updated libraries you will end up doing alot of unnecessary script pre-processing work if you put the run back in with an added compile override file.

   Therefore it is more efficient to run with SKIP_SRC_EXTRACT set to true (see section on this below) which will just run the *make* step when the run is put back in.

   Note that *make* will not detect an updated library or load options changed by handediting the makefile. Therefore to force the makefile to run the load step you need to delete or move the model executable before running with SKIP_SRC_EXTRACT.

   It is also possible to run a load step interactively outside a run by doing:
   - Stoprun the run
   - Move the executable to a different name

- Go into your compile directory eg. for run aaumf:
  `cd ~userid/aaumf/Compile`
- Hand-edit the line similar to the following one in 'Makefile' to include the new library name if necessary:
  `LOADOPTS = -Wl"-Dpermok=yes;streams=on" -lgrib -lgcom1m1s4x1_shmemnam -l_vect -L.`
- Ensure you are running the *make* command under the same programming environment (and hence loader) as the UM version you are running at.
  For example: If you have '. prg_default' in your .profile and '. prg_3.0.2.1' in your SCRIPT job library file you will need to type:
  '`. prg_3.0.2.1`' at the command line
- Type '`make`'
- Resubmit job

4. **Recompile the whole model.**

   Changing a default Fortran compile option eg. ''@fort FCOM_OPTS'' will cause all the decks that your model uses to be recompiled. eg. on a T3E machine if the default is ''@fort FCOM_OPTS='' then it could be changed to
   `@fort FCOM_OPTS=-ea`
   This aborts compilation on finding the first error so is harmless when used with code that already compiles.
5. **Use a combination of *DEFs that hasn't been prebuilt for a section version.**

   Set the BUILDSECT variable in the SUBMIT job library file to true. Any DEF combinations which cannot be found will be built during the model run. The default, BUILDSECT=false, simply flags the missing section def combinations and causes the model to fail in the compile script pre-processing stage.

   Please report any commonly-used combinations that appear to be missing from the pre-built code (at the UK Met. Office only).
6. **Use a new *DEF in the UM system.**

   Set the BUILDSECT variable in the SUBMIT job library file to true. Make a copy of the obj_xref file defining a DEF_SWITCH for the new DEF at the top and adding identifier associated with the DEF to the 'DEFS' line of the relevant sections, and hand-edit your SCRIPT file to use this in your run (script mods to obj_xref do not work at present). Do not add anything to the 'BUILD hostname' lines. All the relevant sections will be recompiled with the new DEF. (A DEF_SWITCH is of the form identifier:defname e.g. A1:ATMOS.)
7. **Untar compile files to edit .f files (UK Met. Office only)**

   WARNING: If you manually unpack and unzip a tar file using the commands in your README file to hand-edit .f files you need to manually do the reverse process of packing them up before resubmitting the run. If you do not pack the files back up, then it may result in the hand-edited files being overwritten.

   At vn4.4 all compile directory (mainly .f and .o) files are placed in a single tar file called $DATAW/comp_$RUNID.tar at the end of each compile session and are unpacked at the beginning of the next run that compiles that experiment. This enables data migration on the T3E to

work more efficiently in migrating one file rather than hundreds of files each time. The tar file is also compressed using gzip to save space.

The code is controlled by a variable called SKIP_TAR_COMPDIR, which may be handedited to *true* in the SUBMIT job library file to avoid the 'tarring' being done at the end of the compile session. The default is *false*.

Irrespective of whether SKIP_TAR_COMPDIR=true/false, it first checks whether there are any .f files in the model compile directory. If there are it assumes that unpacking the tar file is not necessary and does not attempt to uncompress or untar.

If SKIP_TAR_COMPDIR=false, at the end of the run it creates the tar file, compresses it and then deletes all the files in the Compile directory. Each time a tar/gzip is done a README file is created in the $DATAW directory giving the commands required to manually untar files and edit them eg:

```
To unpack tar file:
cd $HOME/abaod/Compile # compile directory
gzip -d $HOME/abaod/comp_abaod.tar.gz # unzip tar file
tar -xf $HOME/abaod/comp_abaod.tar # extract all files
or
tar -xf $HOME/abaod/comp_abaod.tar atmstep1.f  # extract
                                               # individual
                                               # file(s)
To repack tar file:
cd $HOME/abaod/Compile # compile directory
tar -cf $HOME/abaod/comp_abaod.tar *  # tar whole directory
or
tar -uf $HOME/abaod/comp_abaod.tar atmstep1.f # update
                                              # individual
                                              # file(s) in
                                              # tar file
gzip $HOME/abaod/comp_abaod.tar # zip tar file
rm * # remove .f, .c files etc
```

In general it is recommended that people use the tar system
(ie. SKIP_TAR_COMPDIR=false) in case their compile files get migrated at some point in the future. However it adds about 5 minutes to a compile to untar, tar and remove all the compile files so when debugging compile errors by editing .f files using SKIP_SRC_EXTRACT=true it is more productive to turn off the use of tar files by also setting SKIP_TAR_COMPDIR=true.

The reconfiguration files are 'tar'ed up in the same way into a file called $DATAW/recon_$RUNID.tar, with similar instructions being placed in the README file.

Building a model section or small executable

Building personal copies of the small executables

To build one or more small execs with a mod:

- Copy the required version of configure_execs to $HOME.
  i.e. $UMDIR/vn**VN**/scripts/Install/configure_execs

Where **VN** is the version number.
NOTE: It is not possible to pass the VN variable into the script.

- Set environment variable:

```
export TOPDIR=$HOME
```

in either in configure_execs before it is used or before calling the script.

- Alter line:

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

to (for example):

```
for exec in qxcombine qxhistreport qxhistreset \
qxpickup qxsetup
```

where you just list the small execs you want to build.

- To add in a Fortran mod change:

```
$UMDIR/vn$VN/scripts/qsconf \\
-outdir ‘dirname $SRCDIR‘ \\
-execs $exec 1> $OUTDIR/$exec.out 2>&1
```

By adding 1 extra line with the -femod option to:

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

using your mod path instead of $HOME/mods/mymodname.

Use -cemod for C mods (Do ’$UMDIR/vn$VN/scripts/qsconf -help’ for more information on options).

- To add in a compile override:

1. Copy $UMDIR/vn$VN/scripts/allcompfiles to $HOME

   After the line

   ```
   export COMPSF TEMP
   ```

   Add:

   ```
   COMPVARS=${COMPVARS:-$UMDIR/vn$VN/source/compile_vars}
   ```

   Remove code for creation of UCOMPDIR:

   ```
   i.e        if test ! -d $UCOMPDIR
              then
                echo Creating directory $UCOMPDIR
   ```

```
  mkdir -p $UCOMPDIR
fi
```

2. In configure_execs:

   Before call to qsconf add:

   ```
   $HOME/allcompfiles override_filename $HOME update_compvars
   ```

   NOTE:- the file "override_filename" must contain the pathnames of the files containing the actual compile overrides. And in the above example is located under $HOME.

   Change:

   ```
   $UMDIR/vn$VN/scripts/qsconf \\
       -outdir `dirname $SRCDIR` \\
       -execs $exec 1> $OUTDIR/$exec.out 2>&1
   ```

   By adding 1 extra line with the -compvar option to:

   ```
   $UMDIR/vn$VN/scripts/qsconf \\
       -outdir `dirname $SRCDIR` \\
       -compvar $HOME/update_compvars \\
       -execs $exec 1> $OUTDIR/$exec.out 2>&1
   ```

   using the pathname to your "update_compvars" file output from allcompfiles.

- On Cray machines only: Edit the QSUB time, memory and queuename if appropriate. This will be used as a separate job for each small exec.
- Run your version of configure_execs:

  ```
  cd $HOME
  ./configure_execs
  ```

  This will create a directory called something like build_execs_010897_1513 containing a script to build each small exec which it will submit as QSUB jobs (on Cray machines).
- The code will be placed under a directory called exec_build with a subdirectory for each small exec. Each executable then needs to be moved to the directory where it will be used eg:

  ```
  mkdir $HOME/exec
  mv $HOME/exec_build/qxcombine_dir/qxcombine $HOME/exec
  mv $HOME/exec_build/qxhistreport_dir/qxhistreport $HOME/exec
  mv $HOME/exec_build/qxpickup_dir/qxpickup $HOME/exec
  mv $HOME/exec_build/qxhistreset_dir/qxhistreset $HOME/exec
  mv $HOME/exec_build/qxsetup_dir/qxsetup $HOME/exec
  ```

Using personal copies of the small executables within a model run

- Select the ‘‘Using test small executables’’ option via the UMUI in:
  `Sub-Model Independent`
  `-> Configuration and Control`

or alternatively:

Hand edit the variable TESTEXEC in SCRIPT from false to true.
Hand edit the variable EXECN in SCRIPT to make the path of your directory containing the test executables available e.g. `EXECN=~username/exec`