UNIFIED MODEL DOCUMENTATION PAPER NO C1

DYNAMIC ALLOCATION OF PRIMARY FIELDS

by

F Rawlins

Version 1.0 (formerly "Draft Version 1.0")

6th August 1993

Model Version 3.2

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

Modification Record		
Document version	**Author**	**Description.....**
1	F Rawlins	First version

1. INTRODUCTION

Versions of the model before 3.2 allocated space for the primary fields using dimension information obtained from PARAMETER blocks assigned by the compiler. Hence each configuration and each resolution, and each change that implied a difference in memory requirements, required a new compilation and executable file. Modifying accumulated or meaned diagnostic fields changes the memory allocation within the dump so that a small change in requested output needed a new executable file, with all the extra testing demanded by (CF) before implementation in an operational environment.

Hence it is an advantage to provide dimensions for model arrays at run-time rather than during compilation, allowing the same load module to be used for more applications. This is achieved by the dynamic allocation facility on the CRAY, passing sizes into the program at run-time and using these to supply dimensions in a lower level subroutine that controls the whole model. Arrays are then passed down to lower level routines by argument; dimensions are normally passed by COMMON block.

The principal advantage is that it is now possible to run with different output requests, i.e. different STASH files, using the same executable file. It is also a strong requirement from climate modellers that the integrity of model runs should be preserved over long periods of time, possibly including compiler changes. It is likely that keeping executable files for repeated running with different diagnostics will be more robust than keeping source code and relying on upward compatibility from manufacturers. There are also advantages in switching between different resolutions for testing or implementation - for example a change in the no. of levels should not by itself need a change in executable file.

The need for reconfiguration is reduced - reconfiguration is no longer required for continuation runs if the only changes are to output fields. Models at different resolutions can run from the same executable file, although set-up from the user interface is necessary to provide addressing within the control files.

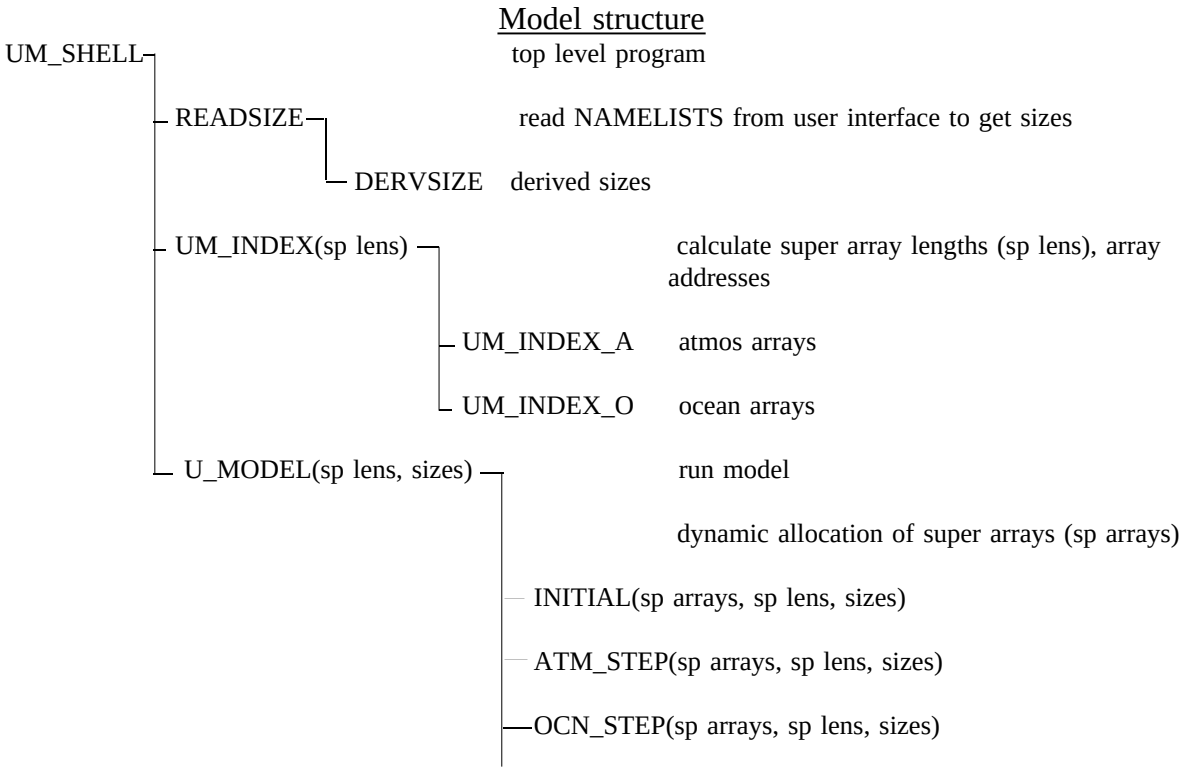
The introduction of a standard interface between control and plug compatible routines leads to a more modular structure, particularly for separating memory requirements for different sub-models.

2. METHOD

User Interface

Calculations of primary and diagnostic array sizes, and the addresses of the constituent fields, remain in the user interface. The user interface passes sizes via a configuration NAMELIST that is inserted into the STASH file dealing with output requests. This is appropriate since the STASH file contains configuration and resolution dependent addresses for processing output fields within the model that must be consistent with the model

dimensions. Similarly boundary file sizes are passed into the run-dependent NAMLST file for dimensioning, since this contains related information dealing with the generation of boundary files and is independent of STASH.



Sizes in READSIZE are read into a COMMON block to be passed back into UM_SHELL. This is done for ease of maintenance. Owing to the CRAY limit on release 5 of the CRAY compiler of a maximum of 255 arguments that can be pased by a subroutine, it is necessary to set up single-dimension super arrays for dynamic allocation. Each super array is composed of arrays in related areas of the model, such as STASH, dump headers, etc and these are defined separately for ocean and atmosphere sub-models, if sub-model dependent. The super arrays are dynamically allocated in U_MODEL and passed down to the next level of the model. The super arrays are then reconstituted into component arrays at the next level down, which interfaces with plug compatible routines.

Model sizes are passed by COMMON blocks through the control levels until the plug compatible routines are accessed where array sizes needed for dynamic allocation in any lower level routine are passed by argument. Where an exception to this arises and dynamic allocation is needed within a control routine an extra copy of an array size is passed by argument into the control routine. The extra copy is suffixed by "DA", e.g. P_FIELD becomes P_FIELDDA, and is used only in dimensioning - all other control functions refer to variables passed by COMMON block. This is an extra requirement arising from portability considerations. Portability is achieved by applying a pre-processor written in C that replaces

the non-portable (at FORTRAN77) dynamic allocation facility of the CRAY. Since the C routine cannot recognise COMMON blocks, super array sizes must be passed by argument into U_MODEL and subsequently down into the lower level routines. Declaring arrays dimensioned by argument puts those arrays on the CRAY heap.

Type declarations of model sizes and COMMON blocks are held in *COMDECK TYPESIZE which contain a complete list of sizes for all sub-models, with only some derived sizes protected by *DEF switches. The declarations must agree with namelists held in *COMDECK CNLSIZES; hence any new variable must be added to both. It is not possible to use the same *COMDECK with *DEF switches separating sub-models because namelist reads must be able to recognise all variables within the namelist and the user interface has to supply a full list, independent of *DEFs. Type declarations are of 3 kinds of variable:
i) sizes passed by the user interface, defining the model configuration; passed by namelist reads in STASH and NAMLST files;
ii) sizes implicit in the data structure, fixed for each version; passed by *COMDECK PARAMETER or assigned in DERVSIZE and passed back into READSIZE;
iii) sizes derived from i) and ii); calculated in DERVSIZE.

Passing sizes and arrays is simplified by the use of CRAY nupdate *COMDECKs, which allow related groups of variables to be defined and used globally. This has the advantage akin to COMMON blocks that changes can be made centrally, without the need to update scores of subroutine argument lists, that variable names can refer to the same fields throughout the model, and that logically connected subsets of variables can be grouped together. Hence it is good practice to use the *COMDECKs whenever large numbers of control variables need to be passed from one level to another. At the lowest levels, in particular where routines are plug-compatible, it is clearer if only those variables accessed within the routine are passed as arguments.

Hence an example of a control subroutine call that interfaces with plug compatible routines at a lower level has the form:

```
      CALL SUB(
     *CALL ARGSIZE
     *CALL ARTD1
     *CALL ARTDUMA
     &           ICODE,CMESSAGE)
```

where ARGSIZE is a dummy COMDECK [originally used to hold model sizes when these were passed by argument; this method has been superseded] and ARTD1, ARTDUMA are the top level super arrays, referring to the D1 array and a list of dump component arrays (atmosphere sub-model), e.g COMDECK ARTDUMA is:-

```
     & A_SPDUM(A_IXDUM(1)), A_SPDUM(A_IXDUM(2), ..... where
```

A_SPDUM is the super array holding dump components for the atmosphere sub-model and A_IXDUM contains addresses of each component array within the super array.

The subroutine definition has the form:

```
      SUBROUTINE SUB(
     *CALL ARGSIZE
     *CALL ARGD1
     *CALL ARGDUMA
     &                ICODE,CMESSAGE)

      IMPLICIT NONE
     *CALL TYPSIZE
     *CALL TYPD1
     *CALL TYPDUMA
```

where TYPSIZE, TYPD1, TYPDUMA are the declarations of array TYPE, and ARGD1, ARGDUMA refer to the component arrays of each super array, i.e. COMDECK ARGDUMA is :-

```
      & A_FIXHD, A_INTHD, .....
```

Hence subroutine SUB in this example forms the interface between the single dimension super array at top levels and the individual arrays input into plug compatible lower level routines. Note that it is essential that ART... and ARG... COMDECKs are matched exactly, with the same number of arguments under the same *DEF choices.

The main limitations of the method are: the FORTRAN limit of 99 continuation lines for any statement, and an upper bound on the number of variable names that can be passed by argument which appears to be machine and compiler specific. The former is exacerbated when variable names are excessively long. Hence control variables that are scalars (but not used for dimensioning - see above) or arrays with sizes that are not configuration dependent are passed by COMMON block within the *CALL TYP--- declaration where possible. Note that there is a practical limit to the scope of dynamic allocation since it is not necessary to have flexibility at run-time for fundamental sizes, such as the size of the fixed header, which would only be changed at a change of version. Care should be taken when dimension sizes are declared using PARAMETER values within a subroutine since this places the array on the CRAY stack in memory. For multitasking applications this implies that multiple copies of stack increments are taken, which can change the memory high water mark. Dimensioning with sizes passed by argument places arrays on the heap which is expanded only as needed.

*COMDECK definitions

MODEL SIZES

	All control routines except READSIZE:
*COMDECK TYPSIZE	declaration of model sizes TYPEs and COMMON blocks; *CALLs TYPOCPAR,COMOCBAS for ocean sub-model sizes.
*COMDECK ARGSIZE	null argument list; dummy *CALL only

Needed in routines which have expanded super arrays - TYPCONA,TYPINFA:

*COMDECK CMAXSIZE This contains maximum sizes for dimensioning arrays of model constants whose sizes are configuration dependent. This allows constants to be read in from a NAMELIST file but maintains the flexibility of dynamic allocation for primary variables. The maximum sizes should agree with the maximum sizes implicit in the front-end user interface.

*COMDECK CNLSIZES

Routine READSIZE only
*CALLs TYPSIZE ; lists of NAMELIST variables for reading user interface control files STASH# and NAMLST#

SUPER ARRAYS

Routines U_MODEL and levels above plug compatible routines

super array TYPE declarations:

*COMDECK TYPSPD1	SPD1 array D1 (and 2 copies for logical, integer)
*COMDECK TYPSPST	SPSTS STASH arrays
*COMDECK TYPSPPTA	A_SPPTR Pointers in D1 (atmosphere)
*COMDECK TYPSPPTO	O_SPPTR Pointers in D1 (ocean)
*COMDECK TYPSPBO	SPBND Input boundary arrays
*COMDECK TYPSPBOA	A_SPBND Input boundary arrays (atmosphere)
*COMDECK TYPSPBOO	O_SPBND Input boundary arrays (ocean)
*COMDECK TYPSPDUA	A_SPDUM Dump headers (atmosphere)
*COMDECK TYPSPDUO	O_SPDUM Dump headers (ocean)
*COMDECK TYPSPCOA	A_SPCON Derived constants (atmosphere)
*COMDECK TYPSPCOO	O_SPCON Derived constants (ocean)
*COMDECK TYPSPINA	A_SPINF Output interface arrays (atmosphere)
*COMDECK TYPSPINO	O_SPINF Output interface arrays (ocean)
*COMDECK TYPSPANA	A_SPANC Ancillary file (atmosphere)
*COMDECK TYPSPANO	O_SPANC Ancillary file (ocean)
*COMDECK TYPSPCPL	AO_SPCPL Coupling fields (atmosphere-ocean)

super array arguments:

*COMDECK ARGSP	super arrays not dependent on sub-model
*COMDECK ARGSPA	super arrays (atmosphere)
*COMDECK ARGSPO	super arrays (ocean)
*COMDECK ARGSPC	super arrays (atmosphere-ocean) coupled

Routines UM_INDEX, U_MODEL
and levels above plug compatible routines

*COMDECK TYPSZSP	lengths of super arrays not dependent on sub-model
*COMDECK TYPSZSPA	lengths of super arrays (atmosphere)
*COMDECK TYPSZSPO	lengths of super arrays (ocean)
*COMDECK TYPSZSPC	lengths of super arrays (atmosphere-ocean) coupled

super array length arguments

*COMDECK ARGSZSP	lengths of super arrays not dependent on sub-model
*COMDECK ARGSZSPA	lengths of super arrays (atmosphere)
*COMDECK ARGSZSPO	lengths of super arrays (ocean)
*COMDECK ARGSZSPC	lengths of super arrays (atmosphere-ocean) coupled

Routines UM_INDEX, U_MODEL
and levels above plug compatible routines

*COMDECK SPINDEX

Addresses of component arrays within super arrays

Routines above level interfacing with plug compatible routines

*COMDECK ARTD1	Argument lists of indexed super array
*COMDECK ARTSTS	which are called by routines interfacing with plug
*COMDECK ARTDUMA	compatible routines.
*COMDECK ARTDUMO	These are the analogues of ARGD1, ARGSTS argument
*COMDECK ARTPTRA	lists in the routine definition and must agree in
*COMDECK ARTPTRO	number of arguments.
*COMDECK ARTCONA	
*COMDECK ARTCONO	
*COMDECK ARTBND	calls ARTBNDA,ARTBNDO as temoprary fix until
*COMDECK ARTBNDA	ocean boundaries sorted out.
*COMDECK ARTBNDO	
*COMDECK ARTINF	
*COMDECK ARTINFA	
*COMDECK ARTINFO	
*COMDECK ARTANC	
*COMDECK ARTAOCPL	

Routines below U_MODEL interfacing with plug compatible routines

*COMDECK TYPD1	declaration of main D1 array
	REAL D1(LEN_TOT)
	LOGICAL LD1(LEN_TOT)
	INTEGER ID1(LEN_TOT)
*COMDECK ARGD1	& D1, LD1, ID1, ! argument D1
*COMDECK TYPSTS	STASH related arrays for describing output requests
*COMDECK ARGSTS	argument list
*COMDECK TYPDUMA	model arrays related to the dump, i.e. fixed headers
*COMDECK ARGDUMA	and lookup headers (atmosphere)

*COMDECK TYPDUMO	as TYPDUMA (ocean)
*COMDECK ARGDUMO	
*COMDECK TYPPTRA	pointers for model variables in D1 and dump
*COMDECK ARGPTRA	row/level dependent constants (atmosphere)
*COMDECK TYPPTRO	as TYPPTRA (ocean)
*COMDECK ARGPTRO	
*COMDECK TYPCONA	Derived model constants [CMAXSIZE should be *CALLed first]
*COMDECK ARGCONA	argument list of constants arrays
*COMDECK TYPCONO	as TYPCONA (ocean)
*COMDECK ARGCONO	[needs TYPSZO in TYPSIZO]
*COMDECK TYPBND	Input boundary conditions not sub-model dependent
*COMDECK ARGBND	argument list
*COMDECK TYPBNDA	Boundary condition arrays (atmosphere)
*COMDECK ARGBNDA	argument list
*COMDECK TYPBNDO	as TYPBNDA (ocean)
*COMDECK ARGBNDO	
*COMDECK TYPINF	Model interface output arrays not sub-model dependent
*COMDECK ARGINF	argument list
*COMDECK TYPINFA	Model interface output arrays (atmosphere) [CMAXSIZE should be *CALLed first]
*COMDECK ARGINFA	argument list
*COMDECK TYPINFO	as TYPINFA (ocean) *COMDECK ARGINFO
*COMDECK TYPANC	Ancillary file arrays (sub-models combined)
*COMDECK ARGANC	argument list
*COMDECK TYPAOCPL	Coupling (atmos-ocean) arrays
*COMDECK ARGAOCPL	argument list

Dump format

The model dump consists of a set of fixed headers followed by look-ups describing data fields and then the data itself. Before vn3.2 the reconfiguration program initialised space for look-ups and data required for the meaned diagnostic output fields. In order to avoid imposing a reconfiguration before each change of STASH file from vn3.2 on, only the prognostic fields and look-ups are read when starting a model run. However all fields must be accessed for a continuation run (CRUN) so that the model READDUMP routine operates

differently depending on the mode of run. An argument has been added to indicate the number of fields to be read. It is also necessary to initialise diagnostic lookups explicitly after the first dump read for a new run (NRUN) using routine INITHDRS so that lengths are correctly assigned.

Notes for changing from vn3.1 to vn3.2

A subroutine will need changing if it accesses the COMDECKs:

CDATA:	CSIZE	parameters independent of resolution
	CSIZEATM	configuration file for atmosphere
	CSIZEOCN	configuration file for ocean
	CDUMP	dump components
CADDRESS		addresses (pointers) + STASH arrays
CBOUND		headers, look-ups for boundary files
CCONSTS		derived constants array plus sizes
CINTF		model output interface arrays plus sizes.

These can be replaced at the top level by references to super arrays. These are passed down until a routine is called that will itself access plug compatible routines. Hence each super array can then be reconstituted into component arrays by referring to addresses within the super array - given by *CALL ART..., with corresponding *CALL ARG... in the argument lists defined within the control subroutines TYP--- will be needed within the routine. In general it is simplest to use *CALL ARGSIZ and ARGSIZA(O) to combine all sizes for atmosphere(ocean) routines and then a selection of the arrays as required. In the general case (for atmosphere routines)

*CALL CDATA is replaced by		*CALL ARGSIZE	
		*CALL ARGD1	
		*CALL ARGDUMA	
*CALL CADDRESS	by	*CALL ARGSIZE	
		*CALL ARGPTRA	if pointers
		*CALL ARGSTS	if STASH arrays
*CALL CBOUND	by	*CALL ARGSIZE	
		*CALL ARGBND	
*CALL CCONSTS	by	*CALL ARGSIZE	
		*CALL ARGCONA	
*CALL CINTF by		*CALL ARGSIZE	
		*CALL ARGINFA	

Notes for conversion from first try at dynamic allocation

1. A new library UM1.PRDY2302.MODS has been set up to hold changed mods. This contains DACDECKC, the list of COMDECKs, (equivalent to RR300393 earlier but I have added in boundary and coupling COMDECKs) where model sizes are passed by COMMON block [DACDECKS is an equivalent modset where model sizes are passed by argument]; DAINDEX, which is a new top level routine holding address, length calculations (new routine UM_INDEX, UM_INDEX_A,UM_INDEX_O). Other modsets should be adapted ones from UM1.PRDYN302.MODS.

2. COMDECKs have been split into atmos and ocean where possible, requiring some re-naming, so that

ARGST	becomes ARGSTS
ARGDU	becomes ARGDUMA or ARGDUMO or both
ARGPT	becomes ARGPTRA or ARGPTRO or both
ARGCON	becomes ARGCONA or ARGCONO or both
ARGBND	unchanged but ARGBNDA,ARGBNDO available if tidying possible
ARGINTF	replaced by ARGINF, ARGINFA, ARGINFO - some rationalisation probably needed.
ARGANC	unchanged, but again future (probably post 3.2) work needed to rationalise ancillary code areas.
ARGCPL	replaced by ARGAOCPL

3. TYP... corresponding to the above ARG... to be changed similarly

4. Other re-naming of model size COMDECKs is as follows:

TYPSIZ??	becomes TYPSZ?? for ?? = HI,DU,PP,ST,D1,BO and
TYPSIZ??	becomes TYPSZ??A and TYPSZ??O for atmosphere and ocean separately where ?? = DS,AN,IN,BO and
TYPSIZSM	becomes TYPSZA and TYPSZO for atmosphere and ocean separately

[NOTE: model sizes will be passed by COMMON block if possible:- to expedite this, simply define *CALL TYPSIZE for control routines, which will pass all model sizes, instead of TYPSZ?? components separately. *CALL ARGSIZE should be retained, but will be dummied out in the *COMDECK definition. This will allow simple transitions between argument calls and COMMON block calls of model sizes if this is needed.]

5. The ARG... are arguments in the subroutine reference i.e

```
      SUBROUTINE SUB(
*CALL ARG...
```

where subroutine SUB and below has references to plug compatible routines. The corresponding call from the control level above must be changed to

```
      CALL SUB(
*CALL ART...
```

in order to reference the super array. The routine calling SUB must also contain the super

array definitions and arguments TYPSP.., ARGSP.., etc and SPINDEX must be *CALLed to obtain addresses if a *CALL ART... is in the code as above.

5. Routines that access both control and plug compatible routines must be changed so that there is an interface routine is imposed above the plug compatible routine, since super arrays and their re-constituted component arrays cannot be mixed within the same routine without added complication. E.G. RR170693 has code to put an interface routine between INITIAL and its call to INIT_EMCORR, since INITIAL only passes super array; most other routines called by INITIAL are themselves control routines.

7. I am continuing to pass ARGSIZE t will be an advantage to pass these sizes by common block (defined within TYPSIZE) and this will be attempted for 3.2. At a later date the ARGSIZE will be removed, but it is easier to maintain them for the moment and dummy out as an intermediate test.

8. The dynamically allocated arrays for the ocean are probably best located in the derived constants super array O_SPCON; TYPSPCOO for passing into OCN_STEP.

9. A further problem has become apparent in separating atmos and ocean sub-models because of the structure of STASH. Currently, STASH calls STWORK for ocean or atmos or slab under a logical switch so that atmos and ocean arguments must always be present. This requires that ocean arrays need to be passed down to each STASH call in ATM_STEP and atmos arrays down through OCN_STEP even though neither are used. There is no difficulty if ocean or atmos configurations are run separately because the offending sections of code would be excluded by *DEFs. This is not the case for coupled runs. It is clearly desirable that the interdependence should be removed. This can be achieved either by introducing new *DEFs to control the argument lists (for ATM_STEP only or OCN_STEP only) or by creating separate STASH routines for each sub-model, i.e. STASH_A, STASH_O, STASH_S for atmos, ocean, slab. The latter choice seems preferable but will require a coding change for each STASH call. Each STASH_ would then call STWORK as before. This proposal requires some thought and could be left to vn3.3 provided that there is no problem with no. of arguments passed under the current scheme.

10. ARGD1,ARTD1 has been extended to include LD1 (logical 'equivalence' of D1 array).

11. Other points to be resolved:
Location of INVERT_OCEAN logical switch.
Duplicate variable name NI - in OCEAN TYPOCPAR and ST_DIAG1, ST_DIAG2, ATM_DYN, INIT_DIAG.

Rick Rawlins
21/06/93